aws

Guida per gli sviluppatori

AWS Lambda



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

AWS Lambda: Guida per gli sviluppatori

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

I marchi e l'immagine commerciale di Amazon non possono essere utilizzati in relazione a prodotti o servizi che non siano di Amazon, in una qualsiasi modalità che possa causare confusione tra i clienti o in una qualsiasi modalità che denigri o discrediti Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà dei rispettivi proprietari, che possono o meno essere affiliati, collegati o sponsorizzati da Amazon.

Table of Contents

Che cos'è AWS Lambda? 1
Come funziona Lambda 1
Funzionalità principali 2
Informazioni correlate 3
Come funziona 3
Funzioni Lambda e gestori di funzioni 4
Ambiente di esecuzione Lambda e runtime 5
Eventi e trigger 5
Autorizzazioni e ruoli Lambda 6
Esecuzione di codice 9
Creazione di architetture basate su eventi 25
Progettazione di un'applicazione 38
Crea la tua prima funzione 46
Prerequisiti 46
Creazione della funzione 48
Invocazione della funzione 54
Eliminazione 58
Fasi successive 58
App e modelli di esempio 60
Elaborazione dei file 60
Integrazione del database 60
Attività pianificate 60
Flussi di lavoro a lunga durata 61
Risorse aggiuntive 61
App di elaborazione file 62
Crea i file di codice sorgente 63
Distribuisci l'app 66
Esecuzione del test dell'app 78
Fasi successive 85
La manutenzione è programmata 86
Prerequisiti 87
Download dei file dell'app di esempio 88
Creazione e compilazione della tabella DynamoDB di esempio 98
Creazione dell'app per la manutenzione programmata 101

Test dell'app ... 105
Fasi successive ... 106
Creazione di un sistema di elaborazione degli ordini con Lambda Durable Functions ... 107
Prerequisiti ... 107
Creare i file del codice sorgente ... 107
Distribuire l'app ... 108
Eseguire il test dell'app ... 109
Fasi successive ... 109
Strumenti di sviluppo ... 111
Strumenti di sviluppo locali ... 111
Infrastructure as code (IaC) ... 112
Strumenti GitHub Actions ... 113
Utensili elettrici per AWS Lambda ... 113
Strumenti di gestione del flusso di lavoro e degli eventi ... 113
Sviluppo locale ... 114
Vantaggi principali dello sviluppo locale ... 114
Prerequisiti ... 114
Autenticazione e controllo degli accessi ... 115
Passaggio da console a sviluppo locale ... 118
Utilizzo delle funzioni a livello locale ... 118
Converti la tua funzione in un modello AWS SAM e usa gli strumenti IaC ... 121
Passaggi successivi ... 121
GitHub Actions ... 122
Flusso di lavoro di esempio ... 122
Risorse aggiuntive ... 123
Infrastructure as code (IaC) ... 123
Strumenti di IaC per Lambda ... 123
Utilizzo di AWS SAM e Infrastructure Composer ... 125
Usando AWS CDK ... 137
Utensili elettrici per AWS Lambda ... 147
Vantaggi principali di Powertools per AWS ... 147
Integrazione di Powertools con AWS ... 147
Passaggi successivi ... 121
Flussi di lavoro ed eventi ... 148
Orchestrazione incentrata sul codice con funzioni durevoli ... 149
Orchestrazione dei flussi di lavoro con Step Functions ... 151

Gestione degli eventi con Stand Scheduler EventBridge EventBridge 152
Funzioni durevoli Lambda 154
Vantaggi principali 154
Come funziona 154
Quando utilizzare le funzioni durevoli 156
Fasi successive 157
Concetti di base 157
Esecuzione durevole 158
DurableContext 158
Fasi 159
Stati di attesa 160
Richiamo di altre funzioni 161
Configurazione durevole delle funzioni 162
Creazione di funzioni durevoli 164
Prerequisiti 165
Crea la funzione duratura 166
Invoca la funzione durable 173
Eliminazione 175
Fasi successive 176
Usando AWS CLI 176
Utilizzo dell'infrastruttura come codice 183
Esempi 190
Processi tolleranti ai guasti di breve durata 190
Processi a esecuzione prolungata 195
Richiamazioni concatenate tra le funzioni 202
Schemi avanzati 206
Fasi successive 215
Sicurezza e autorizzazioni 215
Autorizzazioni del ruolo di esecuzione 215
Crittografia statale 217
CloudTrail registrazione 217
Considerazioni relative a più account 218
Funzionalità di sicurezza Lambda ereditate 218
SDK di esecuzione durevole 219
DurableContext 219
Cosa fa l'SDK 220

Come funziona il checkpoint ... 220
Comportamento di riproduzione ... 221
Operazioni durevoli disponibili ... 222
Come viene misurata la durata delle operazioni ... 229
Runtime supportati ... 232
Richiamo di funzioni durevoli ... 236
Metodi di invocazione ... 236
ARNs Requisito qualificato ... 237
Comprendere il ciclo di vita dell'esecuzione ... 239
Richiamo dal codice dell'applicazione ... 239
Invocazioni concatenate ... 241
Mappature delle sorgenti degli eventi ... 242
Tentativi ... 249
Idempotenza ... 257
Test di funzioni durevoli ... 262
Test locale ... 262
Test sul cloud ... 265
Cosa testare ... 266
Strategia di test ... 267
Errori di debug ... 268
Monitoraggio delle funzioni durature ... 268
CloudWatch metriche ... 268
EventBridge eventi ... 270
AWS X-Ray tracciamento ... 272
Best practice ... 273
Scrivi codice deterministico ... 273
Progettato per l'idempotenza ... 277
Gestisci lo stato in modo efficiente ... 279
Progetta passaggi efficaci ... 281
Usa le operazioni di attesa in modo efficiente ... 281
Ulteriori considerazioni ... 283
Risorse aggiuntive ... 283
Istanze gestite Lambda ... 284
Funzionalità chiave ... 284
Quando usare le istanze gestite Lambda ... 284
Come funziona ... 285

Modello di concorrenza 285
Locazione e isolamento 286
Comprensione delle istanze gestite 286
Prezzi 286
In che modo le istanze gestite Lambda differiscono dal tipo di calcolo Lambda (predefinito) 287
Fasi successive 288
Nozioni di base 288
Creazione di una funzione Lambda Managed Instance (console) 288
Creazione di una funzione Lambda Managed Instance (CLI AWS) 289
Concetti principali 296
fornitori di capacità 297
Dimensionamento 301
Sicurezza 304
Ruolo di operatore 306
Ambiente di esecuzione 309
Pubblicazione della versione 311
Runtime 312
Runtime supportati 312
Considerazioni specifiche sul runtime 312
Fasi successive 313
Runtime Java 313
Runtime Node.js 318
Runtime in Python 323
Runtime .NET 326
Rete 330
Opzioni di connettività 330
Sottorete pubblica con un gateway Internet 330
Endpoint VPC 332
Sottorete privata con gateway NAT 332
Scelta di un'opzione di connettività 333
Fasi successive 334
Monitoraggio 334
Metriche disponibili 334
Frequenza e ritenzione metriche 335
Visualizzazione delle metriche in CloudWatch 336
Utilizzo delle metriche per ottimizzare le prestazioni 336

Fasi successive ... 336
Quote ... 336
Quote di richieste API Lambda ... 337
Quote di risorse per Lambda Managed Instances ... 337
Quote di mappatura delle sorgenti degli eventi ... 337
Richiesta di aumento delle quote ... 338
Fasi successive ... 339
Best practice ... 339
Configurazione del provider di capacità ... 339
Selezione del tipo di istanza ... 339
Configurazione delle funzioni ... 340
Scaling configuration (Configurazione dimensionamento) ... 340
Sicurezza ... 341
Ottimizzazione dei costi ... 341
Monitoraggio e osservabilità ... 342
Considerazioni specifiche sul runtime ... 342
Fasi successive ... 342
risoluzione dei problemi ... 343
Problemi di limitazione e scalabilità ... 343
Problemi di concorrenza ... 344
Problemi di prestazioni ... 346
Problemi con i fornitori di capacità ... 347
Problemi di monitoraggio e osservabilità ... 348
Ottenere ulteriore assistenza ... 350
Fasi successive ... 350
Runtime Lambda ... 351
Runtime supportati ... 352
Nuovi rilasci di runtime ... 355
Policy di deprecazione del runtime ... 356
Modello di responsabilità condivisa ... 356
Utilizzo del runtime dopo la deprecazione ... 358
Ricezione di notifiche di ritiro del runtime ... 360
Runtime obsoleti ... 361
Aggiornamenti della versione di runtime ... 365
Compatibilità con le versioni precedenti ... 366
Modalità di aggiornamento del runtime ... 368

Rollout della versione runtime in due fasi 369
Configurazione della gestione del runtime 370
Rollback di una versione di runtime 371
Aggiornamenti della versione di runtime 373
Modello di responsabilità condivisa 375
Autorizzazioni 377
Ottenere dati sulle funzioni in base al runtime 378
Elenco delle versioni delle funzioni che utilizzano un determinato runtime 378
Identificazione delle funzioni richiamate più di frequente e più di recente 380
Modificazioni runtime 385
Variabili di ambiente specifiche della lingua 385
Script wrapper 385
API Runtime 389
Chiamata successiva 390
Risposta all'invocazione 391
Errore di inizializzazione 391
Errore della chiamata 393
Runtime solo per il sistema operativo 396
Compilazione di un runtime personalizzato 397
Tutorial runtime personalizzato 402
Archivi open source 411
Client di interfaccia Runtime 411
Librerie di eventi 412
Immagini di base del contenitore 412
Strumenti di sviluppo 413
Progetti di esempio 414
Configurazione delle funzioni 415
archivi di file .zip 417
Creazione della funzione 417
Utilizzo dell'editor di codice della console 419
Aggiornamento del codice della funzione 419
Modifica del runtime 420
Modifica dell'architettura 421
Utilizzo dell'API Lambda 421
Download del codice della funzione 421
CloudFormation 422

Encryption (Crittografia) ... 423
Immagini di container ... 430
Requisiti ... 432
Usare un'immagine di AWS base ... 432
Utilizzo di un'immagine di AWS base solo per il sistema operativo ... 433
Utilizzo di un'immagine non AWS di base ... 434
Client di interfaccia runtime ... 435
Autorizzazioni Amazon ECR ... 435
Ciclo di vita delle funzioni ... 438
Memoria ... 440
Quando aumentare la memoria ... 440
Utilizzo della console ... 441
Utilizzando il AWS CLI ... 441
Usando AWS SAM ... 442
Accettazione dei suggerimenti relativi alla memoria delle funzioni (console) ... 442
Archiviazione temporanea ... 444
Casi d’uso ... 444
Utilizzo della console ... 445
Utilizzando il AWS CLI ... 445
Usando AWS SAM ... 446
Set di istruzioni (ARM/x86) ... 447
Vantaggi dell'utilizzo dell'architettura arm64 ... 447
Requisiti per la migrazione all'architettura arm64 ... 448
Compatibilità del codice della funzione con l'architettura arm64 ... 448
Come eseguire la migrazione all'architettura arm64 ... 449
Configurazione dell'architettura del set di istruzioni ... 449
Timeout ... 451
Quando aumentare il timeout ... 451
Utilizzo della console ... 452
Usando il AWS CLI ... 452
Usando AWS SAM ... 452
Configura funzioni durevoli ... 453
Abilita un'esecuzione duratura ... 453
Autorizzazioni IAM per funzioni durevoli ... 454
Best practice per la configurazione ... 456
Variabili di ambiente ... 457

Creazione di variabili di ambiente 457
Scenario di esempio per le variabili di ambiente 461
Recupera le variabili di ambiente 463
Variabili di ambiente di runtime definite 464
Impostazione della sicurezza delle variabili d'ambiente 466
Collegamento delle funzioni a un VPC 470
Autorizzazioni IAM richieste 470
Collegamento di funzioni Lambda a un Amazon VPC nel tuo Account AWS 472
Accesso a Internet se collegato a un VPC 476
IPv6 supporto 476
Le migliori pratiche per l'utilizzo di Lambda con Amazon VPCs 477
Comprensione delle interfacce di rete elastiche Hyperplane () ENIs 479
Utilizzo dei tasti di condizione IAM per le impostazioni VPC 480
Tutorial VPC 485
Collegare funzioni alle risorse di un altro account 486
Prerequisiti 486
Creare un Amazon VPC nell'account della tua funzione 487
Concedere le autorizzazioni VPC al ruolo di esecuzione della funzione 487
Creare una richiesta di connessione peering VPC 488
Preparare l'account della tua risorsa 489
Aggiornare la configurazione VPC nell'account della tua funzione 490
Test della funzione 491
Accesso Internet per funzioni del VPC 493
Rete in entrata 518
Considerazioni per gli endpoint di interfaccia Lambda 518
Creazione di un endpoint di interfaccia per Lambda 519
Creazione di una policy degli endpoint di interfaccia per Lambda 521
File system 523
Regioni supportate 523
Autorizzazioni del ruolo di esecuzione e dell'utente 523
Configurazione di un file system e di un punto di accesso 524
Connessione a un file system (console) 525
Alias 527
Utilizzo di alias 529
Alias ponderati 530
Versioni 536

Creazione di versioni delle funzioni ... 537
Utilizzo delle versioni ... 538
Concessione di autorizzazioni ... 539
Tag ... 541
Autorizzazioni necessarie per lavorare con i tag ... 541
Utilizzo di tag con la console ... 541
Usare i tag con AWS CLI ... 543
Streaming delle risposte ... 545
Limiti di larghezza di banda per lo streaming delle risposte ... 546
Compatibilità VPC con streaming di risposta ... 546
Funzioni di scrittura ... 547
Richiamo di funzioni ... 549
Tutorial: creazione di una funzione di streaming delle risposte con un URL della funzione ... 551
Richiamo di funzioni ... 556
Richiamare una funzione in modo sincrono ... 558
Invocazione asincrona ... 562
Gestione degli errori ... 563
Configurazione ... 564
Conservazione di record ... 566
Invocazione ... 577
Limiti di invocazione sincrona ... 577
Richiamata asincrona per flussi di lavoro di lunga durata ... 577
Gestione dell'esecuzione APIs ... 578
Strumento di mappatura dell'origine degli eventi ... 580
Strumenti di mappatura dell'origine degli eventi e trigger ... 581
Comportamento di batching ... 581
Modalità provisioning ... 585
API della mappatura dell'origine eventi ... 587
Tag dello strumento di mappatura dell'origine degli eventi ... 588
Filtro eventi ... 593
Nozioni di base sul filtraggio di eventi ... 594
Gestione dei record che non soddisfano i criteri di filtraggio ... 596
Sintassi delle regole di filtro ... 597
Collegamento dei criteri di filtro a una mappatura dell'origine evento (console) ... 599
Collegamento dei criteri di filtro a una mappatura dell'origine evento (AWS CLI) ... 600
Collegamento dei criteri di filtro a una mappatura dell'origine evento (AWS SAM) ... 601

Crittografia dei criteri di filtro ... 602
Utilizzo di filtri con diversi Servizi AWS ... 608
Test nella console ... 610
Invocare funzioni con eventi di test ... 610
Creazione di eventi di test privati ... 611
Creazione di eventi di test condivisibili ... 611
Eliminare schemi di eventi di test condivisibili ... 613
Stati delle funzioni ... 614
Stati delle funzioni durante gli aggiornamenti ... 615
Tentativi ... 617
Rilevamento di un ciclo ricorsivo ... 619
Comprendere il rilevamento dei cicli ricorsivi ... 619
Supportato e Servizi AWS SDKs ... 621
Notifiche dei cicli ricorsivi ... 623
Risposta alle notifiche di rilevamento dei cicli ricorsivi ... 624
Consentire l'esecuzione di una funzione Lambda in un ciclo ricorsivo ... 625
Regioni supportate per il rilevamento ricorsivo del loop Lambda ... 627
Funzione URLs ... 628
Creazione di un URL della funzione (console) ... 629
Creazione di un URL di funzione (AWS CLI) ... 631
Aggiungere l'URL di una funzione a un CloudFormation modello ... 632
Cross-Origin Resource Sharing (CORS) ... 633
Funzione di limitazione URLs ... 635
Funzione di disattivazione URLs ... 635
Funzione di cancellazione URLs ... 635
Controllo accessi ... 636
Richiamo di URL di funzioni ... 647
Monitoraggio degli URL della funzione ... 659
Funzione URLs e Amazon API Gateway ... 660
Tutorial: Creazione di un endpoint webhook ... 666
Dimensionamento della funzione ... 680
Comprendere e visualizzare la simultaneità ... 680
Calcolo della simultaneità per una funzione ... 685
Informazioni sulla simultaneità riservata e simultaneità fornita ... 686
Simultaneità riservata ... 687
Simultaneità fornita ... 689

Come Lambda alloca la simultaneità fornita ... 694
Simultaneità riservata e simultaneità fornita di Lambda. ... 695
Informazioni sulla simultaneità e le richieste al secondo ... 696
Quote di simultaneità ... 698
Configurazione della simultaneità riservata ... 700
Configurazione della simultaneità riservata ... 701
Stima accurata della simultaneità riservata richiesta per una funzione ... 703
Configurazione della simultaneità fornita ... 705
Configurazione della simultaneità fornita ... 706
Stima accurata della simultaneità fornita richiesta per una funzione ... 708
Ottimizzazione del codice della funzione quando si utilizza la simultaneità fornita ... 709
Utilizzo di variabili di ambiente per visualizzare e controllare il comportamento della simultaneità fornita ... 710
Informazioni sul comportamento di registrazione e fatturazione con la simultaneità fornita ... 711
Utilizzo di Application Auto Scaling per automatizzare la gestione della simultaneità fornita . 711
Comportamento del dimensionamento ... 716
Velocità di dimensionamento della simultaneità ... 716
Monitoraggio della simultaneità ... 718
Parametri generici di simultaneità ... 718
Parametri della simultaneità fornita ... 718
Lavorare con il parametro `ClaimedAccountConcurrency` ... 721
Compilazione con Node.js ... 724
Versioni SDK incluse nel runtime ... 726
Utilizzo di keep-alive ... 726
Caricamento di certificati CA ... 726
Funzionalità sperimentali di Node.js ... 727
Gestore ... 729
Configurazione del progetto ... 729
Funzione di esempio ... 730
Moduli CommonJS ed ES ... 732
Inizializzazione di Node.js ... 734
Convenzioni di denominazione dei gestori ... 734
Oggetto evento di input ... 735
Modelli di gestione validi ... 735
Utilizzo dell'SDK per JavaScript ... 737
Accesso alle variabili d'ambiente ... 738

Utilizzo dello stato globale ... 739
Best practice ... 739
Implementazione di archivi di file .zip ... 741
Dipendenze di runtime in Node.js ... 741
Creazione di un pacchetto di implementazione .zip senza dipendenze ... 742
Creazione di un pacchetto di implementazione .zip con dipendenze ... 742
Creazione di un livello Node.js per dipendenze ... 744
Percorso di ricerca delle dipendenze e librerie incluse nel runtime ... 745
Creazione e aggiornamento delle funzioni Lambda di Node.js utilizzando file .zip ... 746
Implementazione di immagini di container ... 753
AWS immagini di base per Node.js ... 754
Utilizzo di un'immagine AWS di base ... 755
Utilizzo di un'immagine non di base AWS ... 761
Livelli ... 771
Crea un pacchetto per il contenuto del livello ... 771
Crea il layer in Lambda ... 776
Aggiungete il layer alla vostra funzione ... 777
App di esempio ... 778
Contesto ... 779
Registrazione dei log ... 781
Creazione di una funzione che restituisce i registri ... 781
Utilizzo dei controlli di registrazione avanzati di Lambda con Node.js ... 783
Visualizzazione dei log nella console Lambda ... 790
Visualizzazione dei log nella console CloudWatch ... 790
Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI ... 790
Eliminazione dei log ... 794
Tracciamento ... 795
Utilizzo di ADOT per strumentare le funzioni Node.js ... 796
Utilizzo dell'SDK X-Ray per strumentare le funzioni Node.js ... 796
Attivazione del tracciamento con la console Lambda ... 797
Attivazione del tracciamento con l'API Lambda ... 798
Attivazione del tracciamento con CloudFormation ... 798
Interpretazione di una traccia X-Ray ... 799
Memorizzazione delle dipendenze di runtime in un livello (SDK X-Ray) ... 802
Costruire con TypeScript ... 804
Ambiente di sviluppo ... 805

Definizioni dei tipi per Lambda 806
Gestore 808
Configurazione del progetto 808
Funzione di esempio 809
Moduli CommonJS ed ES 811
Inizializzazione di Node.js 812
Convenzioni di denominazione dei gestori 812
Oggetto evento di input 813
Modelli di gestione validi 814
Utilizzo dell'SDK per JavaScript 816
Accesso alle variabili d'ambiente 817
Utilizzo dello stato globale 818
Best practice 818
Implementare archivi di file .zip 820
Usando AWS SAM 820
Usando il AWS CDK 822
Utilizzo di ed esbuild AWS CLI 825
Implementazione di immagini di container 828
Utilizzo di un'immagine di base Node.js per creare e compilare il codice funzione TypeScript 828
Contesto 836
Registrazione dei log 838
Strumenti e librerie 838
Utilizzo di Powertools per AWS Lambda () e per la registrazione TypeScript strutturata AWS SAM 839
Utilizzo di Powertools for AWS Lambda (TypeScript) e the AWS CDK per la registrazione strutturata 841
Visualizzazione dei log nella console Lambda 845
Visualizzazione dei log nella console CloudWatch 846
Tracciamento 847
Utilizzo di Powertools per () e per il tracciamento AWS Lambda TypeScript AWS SAM 848
Usare Powertools for AWS Lambda (TypeScript) e the AWS CDK per tracciare 850
Interpretazione di una traccia X-Ray 854
Compilazione con Python 855
Versioni SDK incluse nel runtime 856
Funzionalità Python disattivate 857

Formato della risposta 857
Chiusura graduale per le estensioni 858
Gestore 859
Esempio di codice della funzione Lambda in Python 859
Convenzioni di denominazione dei gestori 861
L'oggetto evento Lambda: 862
Accesso e utilizzo dell'oggetto contestuale Lambda 863
Firme dell'handler valide per gli handler Python 864
Restituzione di un valore 864
Utilizzo della AWS SDK per Python (Boto3) versione v2 nell'handler 865
Accesso alle variabili d'ambiente 866
Best practice di codice per le funzioni Lambda con Python Lambda 866
Implementazione di archivi di file .zip 869
Dipendenze di runtime in Python 869
Creazione di un pacchetto di implementazione .zip senza dipendenze 870
Creazione di un pacchetto di implementazione .zip con dipendenze 871
Percorso di ricerca delle dipendenze e librerie incluse nel runtime 873
Utilizzo delle cartelle __pycache__ 875
Creazione di un pacchetto di implementazione .zip con librerie native 875
Creazione e aggiornamento delle funzioni Lambda di Python utilizzando file .zip 877
Implementazione di immagini di container 884
AWS immagini base per Python 885
Usare un'immagine di AWS base 887
Utilizzo di un'immagine non di base AWS 893
Livelli 903
Crea un pacchetto per il contenuto del livello 903
Crea il layer in Lambda 776
Aggiungi il layer alla tua funzione 909
App di esempio 910
Contesto 911
Registrazione dei log 913
Stampa nel log 913
Utilizzo di una libreria di log 914
Utilizzo dei controlli di registrazione avanzati di Lambda con Python 916
Visualizzazione dei log nella console Lambda 921
Visualizzazione dei log nella console CloudWatch 921

Visualizzazione dei log con AWS CLI ... 921
Eliminazione dei log ... 924
Strumenti e librerie ... 925
Utilizzo di Powertools per AWS Lambda (Python) AWS SAM e per la registrazione strutturata ... 925
Utilizzo di Powertools per AWS Lambda (Python) AWS CDK e per la registrazione strutturata ... 929
Test ... 936
Test delle applicazioni serverless ... 937
Tracciamento ... 939
Utilizzo di Powertools per AWS Lambda (Python) e di AWS SAM per il tracciamento ... 940
Utilizzo di Powertools per AWS Lambda (Python) e del AWS CDK per il tracciamento ... 943
Utilizzo di ADOT per strumentare le funzioni Python ... 948
Utilizzo dell'SDK X-Ray per strumentare le funzioni Python ... 948
Attivazione del tracciamento con la console Lambda ... 949
Attivazione del tracciamento con l'API Lambda ... 949
Attivazione del tracciamento con CloudFormation ... 950
Interpretazione di una traccia X-Ray ... 950
Memorizzazione delle dipendenze di runtime in un livello (SDK X-Ray) ... 953
Compilazione con Ruby ... 955
Versioni SDK incluse nel runtime ... 957
Abilitazione di Yet Another Ruby JIT (YJIT) ... 957
Gestore ... 958
Nozioni di base sull'handler Ruby ... 958
Best practice di codice per le funzioni Lambda con Ruby ... 959
Implementazione di archivi di file .zip ... 962
Dipendenze in Ruby ... 962
Creazione di un pacchetto di implementazione .zip senza dipendenze ... 963
Creazione di un pacchetto di implementazione .zip con dipendenze ... 963
Creazione di un livello Ruby per dipendenze ... 965
Creazione di un pacchetto di implementazione .zip con librerie native ... 965
Creazione e aggiornamento delle funzioni Lambda di Ruby utilizzando file .zip ... 967
Implementazione di immagini di container ... 974
AWS immagini di base per Ruby ... 975
Usare un'immagine di AWS base ... 975
Utilizzo di un'immagine non di base AWS ... 982

Livelli 992
Crea un pacchetto per il contenuto del livello 992
Crea il livello in Lambda. 776
Usare le gemme dei livelli in una funzione 999
Aggiungi il livello alla tua funzione. 1000
app di esempio 1001
Contesto 1002
Registrazione dei log 1003
Creazione di una funzione che restituisce i registri 1003
Visualizzazione dei log nella console Lambda 1005
Visualizzazione dei log nella console CloudWatch 1005
Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI 1005
Eliminazione dei log 1009
Utilizzo della libreria dei logger Ruby 1009
Tracciamento 1011
Abilitazione del tracciamento attivo con l'API Lambda 1016
Abilitazione del tracciamento attivo con CloudFormation 1017
Memorizzazione delle dipendenze di runtime in un layer 1018
Compilazione con Java 1019
Gestore 1023
Configurazione di Go Handler 1023
Esempio di codice della funzione Lambda 1024
Definizioni di classe valide per i gestori Java 1029
Convenzioni di denominazione dei gestori 1030
Definizione e accesso all'oggetto evento di input 1031
Accesso e utilizzo dell'oggetto contestuale Lambda 1032
Utilizzo della AWS versione v2 nell'handler 1033
Accesso alle variabili d'ambiente 1034
Utilizzo dello stato globale 1035
Best practice di codice per funzioni Lambda in Java 1035
Implementazione di archivi di file .zip 1038
Prerequisiti 1038
Strumenti e librerie 1039
Creazione di un pacchetto di distribuzione con Gradle 1040
Utilizzo dei livelli per le dipendenze 1041
Creazione di un pacchetto di distribuzione con Maven 1041

Caricamento di un pacchetto di implementazione con la console Lambda 1044
Caricamento di un pacchetto di distribuzione con AWS CLI 1045
Caricamento di un pacchetto di distribuzione con AWS SAM 1047
Implementazione di immagini di container 1049
AWS immagini di base per Java 1050
Utilizzando un'immagine AWS di base 1051
Utilizzo di un'immagine non di base AWS 1060
Livelli 1072
Crea un pacchetto per il contenuto del livello 1072
Crea il layer in Lambda 776
Aggiungete il layer alla vostra funzione 1076
Serializzazione personalizzata 1077
Quando usare la serializzazione personalizzata 1077
Implementazione della serializzazione personalizzata 1078
Test della serializzazione personalizzata 1079
Comportamento di avvio personalizzato 1080
Informazioni sulla variabile di ambiente JAVA_TOOL_OPTIONS 1080
Patch Log4j per Log4Shell 1083
Ahead-of-Time (AOT) e cache CDS 1083
Contesto 1084
Contesto nelle applicazioni di esempio 1086
Registrazione dei log 1088
Creazione di una funzione che restituisce i registri 1088
Utilizzo dei controlli di registrazione avanzati di Lambda con Java 1090
Implementazione della registrazione avanzata con Log4j2 e J SLF4 1093
Strumenti e librerie 1097
Utilizzo di Powertools per AWS Lambda (Java) e per la registrazione strutturata AWS SAM 1097
Visualizzazione dei log nella console Lambda 1102
Visualizzazione dei log nella console CloudWatch 1102
Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI 1102
Eliminazione dei log 1106
Codice di registrazione dei log di esempio 1106
Tracciamento 1107
Utilizzo di Powertools per (Java) e per il AWS Lambda tracciamento AWS SAM 1108
Utilizzo di Powertools per AWS Lambda (Java) e AWS CDK per il tracciamento 1110

Utilizzo di ADOT per strumentare le funzioni Java ... 1122
Utilizzo dell'SDK X-Ray per strumentare le funzioni Java ... 1122
Attivazione del tracciamento con la console Lambda ... 1123
Attivazione del tracciamento con l'API Lambda ... 1123
Attivazione del tracciamento con CloudFormation ... 1124
Interpretazione di una traccia X-Ray ... 1124
Memorizzazione delle dipendenze di runtime in un livello (SDK X-Ray) ... 1127
Tracciamento X-Ray in applicazioni di esempio (SDK X-Ray) ... 1128
App di esempio ... 1130
Compilazione con Go ... 1132
Supporto per il runtime Go ... 1132
Strumenti e librerie ... 1133
Gestore ... 1135
Configurazione di Go Handler ... 1135
Esempio di codice della funzione Lambda ... 1136
Convenzioni di denominazione dei gestori ... 1139
Definizione e accesso all'oggetto evento di input ... 1139
Accesso e utilizzo dell'oggetto contestuale Lambda ... 1140
Firme dell'handler valide per gli handler Go ... 1141
Utilizzo della AWS SDK per Go versione v2 nell'handler ... 1142
Accesso alle variabili d'ambiente ... 1143
Utilizzo dello stato globale ... 1144
Procedure consigliate di codice per le funzioni Go Lambda ... 1144
Contesto ... 1146
Variabili supportate, metodi e proprietà nell'oggetto contesto ... 1146
Accesso alle informazioni relative al contesto di invocazione ... 1147
Utilizzo del contesto nelle inizializzazioni e nelle AWS chiamate dei client SDK ... 1149
Implementazione di archivi di file .zip ... 1150
Creazione di un file .zip su macOS e Linux ... 1150
Creazione di un file .zip su Windows ... 1152
Creazione e aggiornamento delle funzioni Lambda di Go utilizzando file .zip ... 1155
Implementazione di immagini di container ... 1162
AWS immagini di base per l'implementazione delle funzioni Go ... 1162
Client di interfaccia di runtime per Go ... 1163
Utilizzo di un'immagine di AWS base solo per il sistema operativo ... 1163
Utilizzo di un'immagine non di base AWS ... 1171

Livelli 1180
Registrazione dei log 1181
Creazione di una funzione che restituisce i registri 1181
Visualizzazione dei log nella console Lambda 1183
Visualizzazione dei log nella console CloudWatch 1183
Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI 1184
Eliminazione dei log 1187
Tracciamento 1188
Utilizzo di ADOT per strumentare le funzioni Go 1189
Utilizzo dell'SDK X-Ray per strumentare le funzioni Go 1189
Attivazione del tracciamento con la console Lambda 1189
Attivazione del tracciamento con l'API Lambda 1190
Attivazione del tracciamento con CloudFormation 1190
Interpretazione di una traccia X-Ray 1191
Compilazione con C# 1195
Ambiente di sviluppo 1195
Installazione dei modelli di progetto .NET 1195
Installazione e aggiornamento degli strumenti della CLI 1196
Gestore 1197
Configurazione del progetto del gestore C# 1197
Esempio di codice della funzione Lambda C# 1199
Gestori di librerie di classi 1202
Gestori di assembly eseguibili 1203
Firme del gestore valide per le funzioni C# 1204
Convenzioni di denominazione dei gestori 1204
Serializzazione nelle funzioni Lambda C# 1205
Funzioni basate su file 1208
Accesso e utilizzo dell'oggetto contestuale Lambda 1208
Usando la SDK per .NET v3 nel tuo gestore 1209
Accesso alle variabili d'ambiente 1210
Utilizzo dello stato globale 1211
Semplificazione del codice delle funzioni con il framework Lambda Annotations 1211
Best practice di codice per le funzioni Lambda con C# 1212
Pacchetto di implementazione 1215
CLI globale Lambda .NET 1216
AWS SAM 1222

AWS CDK ... 1225
ASP.NET ... 1229
Livelli ... 1235
Implementazione di immagini di container ... 1236
AWS immagini di base per.NET ... 1237
Utilizzando un'immagine AWS di base ... 1237
Utilizzo AWS di un'immagine non di base ... 1240
Compilazione AOT nativa ... 1244
Runtime Lambda ... 1244
Prerequisiti ... 1245
Nozioni di base ... 1245
Serializzazione ... 1248
Rifinitura ... 1249
Risoluzione dei problemi ... 1250
Contesto ... 1251
Registrazione dei log ... 1254
Creazione di una funzione che restituisce i registri ... 1254
Utilizzo dei controlli di registrazione avanzati di Lambda con .NET ... 1255
Strumenti e librerie ... 1263
Utilizzo di Powertools per AWS Lambda (.NET) e per la registrazione strutturata AWS SAM ... 1263
Visualizzazione dei log nella console Lambda ... 1266
Visualizzazione dei log nella console CloudWatch ... 1266
Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI ... 1267
Eliminazione dei log ... 1270
Tracciamento ... 1271
Utilizzo di Powertools per AWS Lambda (.NET) e di AWS SAM per il tracciamento ... 1272
Utilizzo dell'SDK X-Ray per strumentare le funzioni .NET ... 1275
Attivazione del tracciamento con la console Lambda ... 1276
Attivazione del tracciamento con l'API Lambda ... 1277
Attivazione del tracciamento con CloudFormation ... 1277
Interpretazione di una traccia X-Ray ... 1278
Test ... 1281
Test delle applicazioni serverless ... 1282
Costruire con PowerShell ... 1285
Ambiente di sviluppo ... 1287

Pacchetto di implementazione 1288
Creazione di una funzione Lambda 1288
Gestore 1291
Restituzione dei dati 1292
Contesto 1293
Registrazione dei log 1294
Creazione di una funzione che restituisce i registri 1294
Visualizzazione dei log nella console Lambda 1296
Visualizzazione dei log nella console CloudWatch 1296
Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI 1297
Eliminazione dei log 1300
Compilazione con Rust 1301
Gestore 1303
Configurazione del progetto Rust Handler 1303
Esempio di codice della funzione Rust Lambda 1304
Definizioni di classe valide per i gestori Rust 1307
Convenzioni di denominazione dei gestori 1308
Definizione e accesso all'oggetto evento di input 1309
Accesso e utilizzo dell'oggetto contestuale Lambda 1310
Usando il nel tuo gestore AWS SDK per Rust 1310
Accesso alle variabili d'ambiente 1311
Utilizzo dello stato condiviso 1311
Best practice di codice per funzioni Lambda in Rust 1312
Contesto 1314
Accesso alle informazioni relative al contesto di invocazione 1314
Eventi HTTP 1316
Implementazione di archivi di file .zip 1319
Prerequisiti 1319
Compilazione della funzione 1319
Implementazione della funzione 1320
Richiamo della funzione 1322
Livelli 1323
Registrazione dei log 1324
Creazione di una funzione che scrive i log 1324
Registrazione avanzata con la cassa Tracing 1324
Best practice 1327

Codice della funzione ... 1327
Configurazione delle funzioni ... 1329
Scalabilità delle funzioni ... 1330
Parametri e allarmi ... 1331
Utilizzo di flussi ... 1331
Best practice di sicurezza ... 1333
Test di funzioni serverless ... 1334
Obiettivi aziendali specifici ... 1335
Cosa testare ... 1335
Come testare le soluzioni serverless ... 1336
Tecniche di test ... 1337
Test nel cloud ... 1338
Test con mock ... 1340
Test localmente utilizzando AWS SAMCLI ... 1342
Test con emulazione ... 1342
Best practice ... 1343
Dai priorità ai test nel cloud ... 1344
Struttura il tuo codice per la testabilità ... 1344
Accelera i cicli di feedback sullo sviluppo ... 1344
Concentrati sui test di integrazione ... 1344
Crea ambienti di test isolati ... 1345
Usa i mock per una logica aziendale isolata ... 1346
Utilizza gli emulatori con parsimonia ... 1347
Problematiche legate ai test locali ... 1347
Esempio: la funzione Lambda crea un bucket S3 ... 1348
Esempio: una funzione Lambda elabora i messaggi da una coda Amazon SQS ... 1348
Domande frequenti ... 1349
Risorse e passaggi successivi ... 1350
Lambda SnapStart ... 1352
Casi d'uso ... 1353
Funzionalità e limitazioni supportate ... 1353
Regioni supportate ... 1354
Considerazioni sulla compatibilità ... 1354
Prezzi ... 1355
Attivazione SnapStart ... 1357
Attivazione SnapStart (console) ... 1357

Attivazione SnapStart ()AWS CLI 1358
Attivazione SnapStart (API) 1360
Stati delle funzioni 1361
Aggiornamento di uno snapshot 1361
Utilizzo con SnapStart AWS SDKs 1362
Utilizzando SnapStart with CloudFormation, e AWS SAMAWS CDK 1362
Eliminazione di snapshot 1362
Gestione dell'unicità 1364
Evitare di salvare lo stato 1364
Uso di CSPRNG 1366
Strumento di scansione (Java) 1369
Hook di runtime 1370
Java 1370
Python 1374
.NET 1376
Monitoraggio 1379
Log CloudWatch 1379
AWS X-Ray 1380
API di telemetria 1380
Parametri di Gateway API e della funzione URL 1381
Modello di sicurezza 1382
Best practice 1383
Ottimizzazione prestazioni 1383
Best practice per la rete 1387
Risoluzione dei problemi 1389
SnapStartNotReadyException 1389
SnapStartTimeoutException 1389
Errore interno del servizio 500 1390
401 - Autorizzazione negata 1390
UnknownHostException (Java) 1390
Errori di creazione snapshot 1391
Frequenza di creazione degli snapshot 1391
Isolamento degli inquilini 1393
Quando utilizzare la modalità di isolamento dei tenant 1394
Funzionalità e limitazioni supportate 1394
Supportato Regioni AWS 1394

Considerazioni 1394
Prezzi 1395
Modalità di isolamento 1395
Abilitare l'isolamento dei tenant 1397
Console 1397
AWS CLI 1397
"Hello, World!" 1398
CloudFormation 1398
Richiamo di funzioni con isolamento del tenant 1400
AWS CLI 1400
"Hello, World!" 1400
Gateway API 1401
SDK 1402
Accesso tramite codice 1405
Accesso 1405
Modelli di utilizzo 1406
Monitoraggio 1407
Monitoraggio 1409
Comprendere la registrazione per la modalità isolata dei tenant 1409
risoluzione dei problemi 1411
InvalidParameterValueException 1411
TooManyRequestsException 1411
Integrazione con altri servizi 1412
Creazione di un trigger 1412
Elenco dei servizi 1413
Apache Kafka 1415
MSK 1417
Apache Kafka gestito dal cliente 1467
Ridimensionamento dei poller degli eventi 1486
Posizioni di polling e flussi 1494
Gruppo di consumer 1495
Filtro eventi 1496
Registri di schemi con sorgenti di eventi 1501
Apache Kafka a bassa latenza 1530
Riprova le configurazioni 1532
Conserva le chiamate non riuscite 1538

Destinazione Kafka in caso di errore 1546
Risoluzione dei problemi 1551
Gateway API 1554
Scelta di un tipo di API 1555
Aggiunta di un endpoint alla funzione Lambda 1556
Integrazione proxy 1556
Formato dell'evento 1557
Formato della risposta 1557
Permissions 1558
Applicazione di esempio 1560
Il gestore di eventi di Powertools for Lambda AWS 1561
Tutorial 1562
Errori 1579
API Gateway vs funzione URLs 1580
Infrastructure Composer 1585
Esportazione di una funzione Lambda in Infrastructure Composer 1586
Altre risorse 1588
CloudFormation 1589
Amazon DocumentDB 1592
Esempio di evento Amazon DocumentDB 1593
Prerequisiti e autorizzazioni 1594
Configurare la sicurezza della rete 1596
Creazione di una mappatura dell'origine degli eventi Amazon DocumentDB (console) 1599
Creazione di una mappatura dell'origine degli eventi Amazon DocumentDB (SDK o CLI) .. 1601
Posizioni di partenza di polling e flussi 1603
Monitoraggio dell'origine degli eventi di Amazon DocumentDB 1604
Tutorial 1604
DynamoDB 1631
Flussi di polling e batching 1631
Posizioni di partenza di polling e flussi 1633
Lettori simultanei 1633
Esempio di evento 1633
Creare lo strumento di mappatura 1635
Errori degli elementi del batch 1638
Gestione degli errori 1655
Elaborazione stateful 1662

Parametri ... 1667
Filtro eventi ... 1669
Tutorial ... 1678
EC2 ... 1695
Concessione di autorizzazioni a EventBridge (Eventi CloudWatch) ... 1696
Elastic Load Balancer (Application Load Balancer) ... 1697
Gestore di eventi di Powertools per Lambda AWS ... 1699
Richiamo tramite un pianificatore EventBridge ... 1700
Configurare il ruolo di esecuzione ... 1700
Creare una pianificazione. ... 1700
Risorse correlate ... 1705
IoT ... 1706
Flussi di dati Kinesis ... 1708
Flussi di polling e batching ... 1708
Esempio di evento ... 1711
Creare lo strumento di mappatura ... 1712
Errori degli elementi del batch ... 1717
Gestione degli errori ... 1736
Elaborazione stateful ... 1743
Parametri ... 1746
Filtro eventi ... 1749
Tutorial ... 1753
Kubernetes ... 1770
Controller AWS per Kubernetes (ACK) ... 1770
Crossplane ... 1771
MQ ... 1772
Informazioni sul gruppo di consumatori Lambda per Amazon MQ ... 1774
Configurare un'origine eventi ... 1778
Parametri ... 1785
Filtro eventi ... 1786
Risoluzione dei problemi ... 1792
RDS ... 1794
Configurazione della funzione per l'utilizzo con le risorse RDS ... 1794
Connessione a un database Amazon RDS in una funzione Lambda ... 1801
Elaborazione di notifiche di eventi da Amazon RDS ... 1820
Completare il tutorial su Lambda e Amazon RDS ... 1821

Amazon RDS e DynamoDB 1821
S3 1826
Tutorial: uso di un trigger S3 1828
Tutorial: Uso di un trigger Amazon S3 per creare miniature 1853
Secrets Manager 1882
Scelta di un approccio 1882
Quando usare Secrets Manager 1883
Utilizzo dei AWS parametri e dei segreti (estensione Lambda) 1883
Utilizzo dell'utilità di parametrizzazione di Powertools per AWS Lambda 1894
SQS 1901
Informazioni sul comportamento di polling e batch per gli strumenti di mappatura dell'origine degli eventi di Amazon SQS 1902
Utilizzo della modalità provisioning con le mappature delle sorgenti degli eventi di Amazon SQS 1903
Configurazione della modalità di provisioning per la mappatura delle sorgenti degli eventi Amazon SQS 1904
Esempio di evento con messaggio di coda standard 1906
Esempio di evento di messaggio di coda FIFO 1907
Creare lo strumento di mappatura 1908
Comportamento del dimensionamento 1913
Gestione degli errori 1916
Parameters 1931
Filtro eventi 1933
Tutorial 1938
Tutorial su SQS tra più account 1956
Step Functions 1963
Quando usare Step Functions 1963
Quando non usare Step Functions 1969
Batch S3 1971
Chiamata di funzioni Lambda dalle operazioni in batch Amazon S3 1972
SNS 1974
Utilità di idempotenza di Powertools per Lambda AWS 1974
Aggiunta di un trigger di argomento Amazon SNS per una funzione Lambda utilizzando la console 1975
Aggiunta manuale di un trigger di argomento Amazon SNS per una funzione Lambda 1975
Esempio di forma evento SNS 1976

Tutorial 1977
Autorizzazioni Lambda 1997
Ruolo di esecuzione (autorizzazioni per le funzioni per accedere ad altre risorse) 1999
Creazione di un ruolo di esecuzione nella console di IAM 1999
Creazione e gestione dei ruoli con AWS CLI 2000
Garantisci l'accesso minimo ai privilegi per il tuo ruolo di esecuzione Lambda 2002
Aggiornare un ruolo di esecuzione 2002
AWS politiche gestite 2004
ARN della funzione di origine 2007
Autorizzazioni di accesso (autorizzazioni per altre entità di accedere alle tue funzioni) 2012
Policy basate sull'identità 2012
Policy basate sulle risorse 2019
Controllo dell'accesso basato sugli attributi 2028
Risorse e condizioni 2035
Sicurezza, governance e conformità 2043
Protezione dei dati 2044
Crittografia in transito 2045
Crittografia dei dati a riposo 2045
Uso di ruoli collegati ai servizi 2050
Autorizzazioni di ruolo collegate ai servizi per Lambda 2051
Creazione di un ruolo collegato ai servizi per Lambda 2051
Modifica di un ruolo collegato al servizio per Lambda 2052
Eliminazione di un ruolo collegato al servizio per Lambda 2052
Regioni supportate per i ruoli collegati ai servizi Lambda 2053
Identity and Access Management 2054
Destinatari 2055
Autenticazione con identità 2055
Gestione dell'accesso con l'utilizzo delle policy 2056
Funzionamento di AWS Lambda con IAM 2058
Esempi di policy basate sull'identità 2064
AWS politiche gestite 2067
Risoluzione dei problemi 2076
Governance 2078
Controlli proattivi con Guard 2081
Controlli proattivi con AWS Config 2085
Controlli di rilevamento con AWS Config 2093

Registrazione del codice 2098
Scansione del codice 2101
Osservabilità 2106
Convalida della conformità 2114
Resilienza 2114
Sicurezza dell'infrastruttura 2115
Protezione dei carichi di lavoro con endpoint pubblici 2116
Autenticazione e autorizzazione 2116
Protezione degli endpoint API 2117
Firma del codice 2118
Convalida della firma 2118
Creazione di una configurazione 2119
Permissions 2121
Tag di configurazione della firma del codice 2123
Funzioni di monitoraggio e debug 2127
Prezzi 2127
Parametri di funzione 2128
Visualizzare i parametri delle funzioni 2128
Tipi di parametri 2129
Log delle funzioni 2138
Scelta della destinazione del servizio a cui inviare i log 2138
Configurazione delle destinazioni dei log 2139
Configurazione dei controlli di registrazione avanzati per le funzioni Lambda 2139
Formati di registro 2140
Filtraggio a livello di log 2147
Registrazione con CloudWatch Logs 2153
Accedi con Firehose 2172
Uso di con Amazon S3 2174
Log di CloudTrail 2179
Eventi di dati Lambda in CloudTrail 2180
Eventi di gestione Lambda in CloudTrail 2182
Utilizzo di CloudTrail per risolvere i problemi relativi alle origini eventi Lambda disabilitate . 2184
Esempi di eventi Lambda 2185
AWS X-Ray 2188
Informazioni sui monitoraggi di X-Ray 2189
Comportamento di tracciamento predefinito in Lambda 2194

Autorizzazioni del ruolo di esecuzione ... 2195
Abilitazione del Active tracciamento con l'API Lambda ... 2195
Abilitazione del tracciamento con ActiveCloudFormation ... 2196
Informazioni sulle funzioni ... 2197
Come funziona ... 2197
Prezzi ... 2198
Runtime supportati ... 2198
Attivazione di Lambda Insights nella console ... 2198
Attivazione di Lambda Insights a livello di programmazione ... 2198
Uso del pannello di controllo di Lambda Insights ... 2199
Rilevamento di anomalie di funzione ... 2201
Risoluzione dei problemi di una funzione ... 2203
Fasi successive ... 2205
Visualizzazione dei parametri dell'applicazione ... 2206
Application Signals ... 2209
In che modo Application Signals si integra con Lambda ... 2209
Prezzi ... 2210
Runtime supportati ... 2210
Abilitazione di Application Signals nella console Lambda ... 2210
Utilizzo del pannello di controllo di Application Signals ... 2211
Esecuzione del debug con VS Code ... 2213
Runtime supportati ... 2213
Sicurezza e debug da remoto ... 2213
Prerequisiti ... 2214
Debug da remoto delle funzioni Lambda ... 2214
Disabilitazione de debug da remoto ... 2215
Informazioni aggiuntive ... 2216
Livelli Lambda ... 2217
Come usare i livelli ... 2219
Livelli e versioni di livelli ... 2219
Creazione di pacchetti dei livelli ... 2221
Percorsi dei livelli per ciascun runtime Lambda ... 2221
Creazione ed eliminazione di livelli ... 2225
Creazione di un livello ... 2225
Eliminazione della versione di un livello ... 2226
Aggiunta di livelli ... 2227

Ricerca di informazioni sul livello ... 2229
Livelli con CloudFormation ... 2231
Livelli con AWS SAM ... 2232
Estensioni Lambda ... 2233
Ambiente di esecuzione ... 2234
Impatto su prestazioni e risorse ... 2235
Autorizzazioni ... 2236
Configurazione delle estensioni ... 2237
Configurazione delle estensioni (archivio di file .zip) ... 2237
Utilizzo delle estensioni nelle immagini di container ... 2237
Passaggi successivi ... 2238
Partner con estensioni ... 2239
AWSEstensioni gestite da ... 2240
API estensioni ... 2241
Ciclo di vita dell'ambiente di esecuzione Lambda ... 2242
Riferimento all'API delle estensioni ... 2252
API di telemetria ... 2259
Creazione di estensioni tramite l'API di telemetria ... 2261
Registrazione delle estensioni ... 2262
Creazione di un listener di telemetria ... 2263
Specifica di un protocollo di destinazione ... 2264
Configurazione dell'utilizzo della memoria e del buffering ... 2265
Invio di una richiesta di sottoscrizione all'API di telemetria ... 2267
Messaggi dell'API di telemetria in entrata ... 2268
Guida di riferimento alle API ... 2271
Riferimento allo schema `Event` ... 2275
Conversione di eventi in intervalli OTel ... 2297
Log API ... 2303
Risoluzione dei problemi ... 2316
Configurazione ... 2316
Configurazioni della memoria ... 2317
Configurazioni collegate alla CPU ... 2317
Timeout ... 2317
Perdita di memoria tra invocazioni ... 2318
Risultati asincroni restituiti a una invocazione successiva ... 2321
Implementazione ... 2325

Generale: autorizzazione negata/Impossibile caricare tale file 2326
Generale: si verifica un errore quando si chiama il UpdateFunctionCode 2327
Amazon S3: codice di errore. PermanentRedirect 2327
Generale: impossibile trovare, impossibile caricare, impossibile importare, classe non trovata, file o directory non trovati 2328
Generale: handler di metodi non definito 2328
Generale: il limite di memorizzazione del codice Lambda è stato superato 2329
Lambda: conversione dei livelli non riuscita 2330
Lambda: o InvalidParameterValueException RequestEntityTooLargeException 2330
Lambda: InvalidParameterValueException 2331
Lambda: quote di simultaneità e memoria 2331
Lambda: configurazione alias non valida per la concorrenza fornita 2332
Invocazione 2333
Lambda: timeout della funzione durante la fase di inizializzazione (Sandbox.Timedout) 2334
IAM: lambda: InvokeFunction non autorizzato 2334
Lambda: impossibile trovare un bootstrap (Runtime) valido. InvalidEntrypoint) 2335
Lambda: l'operazione non può essere eseguita ResourceConflictException 2335
Lambda: la funzione è bloccata in sospeso 2335
Lambda: una funzione sta usando tutta la simultaneità 2336
Generale: impossibile richiamare la funzione con altri account o servizi 2336
Generale: il richiamo della funzione è in loop 2336
Lambda: routing alias con simultaneità fornita 2336
Lambda: avvii a freddo con simultaneità fornita 2337
Lambda: avvii a freddo con nuove versioni 2338
Lambda: uscita imprevista di Node.js in fase di esecuzione (Runtime). NodejsExit) 2338
EFS: la funzione non è in grado di montare il file system EFS 2339
EFS: la funzione non è in grado di connettersi al file system EFS 2339
EFS: la funzione non è in grado di montare il file system EFS a causa del timeout 2339
Lambda: Lambda ha rilevato un processo IO che stava impiegando troppo tempo 2339
Contenitore: errori CodeArtifactUserException 2340
Contenitore: InvalidEntrypoint errori 2340
Esecuzione 2340
Lambda: debug remoto con Visual Studio Code 2342
Lambda: l'esecuzione richiede troppo tempo 2342
Lambda: payload per eventi imprevisti 2342
Lambda: dimensioni del carico utile inaspettatamente grandi 2343

Lambda: errori di codifica e decodifica JSON 2344
Lambda: i log o le tracce non vengono visualizzati 2344
Lambda: non vengono visualizzati tutti i log della mia funzione 2345
Lambda: la funzione restituisce prima del termine dell'esecuzione 2346
Lambda: esecuzione di una versione o di un alias di funzione non intenzionali 2346
Lambda: rilevamento di loop infiniti 2347
Generale: indisponibilità del servizio downstream 2348
AWS SDK: versioni e aggiornamenti 2349
Python: caricamento librerie errato 2350
Java: la funzione impiega più tempo per elaborare gli eventi dopo l'aggiornamento a Java 17 da Java 11 2351
Kafka: gestione degli errori e nuovi tentativi di configurazione 2351
Mappatura delle origini eventi 2352
Identificazione e gestione della limitazione 2352
Errori nella funzione di elaborazione 2354
Identificazione e gestione della contropressione 2356
Rete 2357
VPC: la funzione perde l'accesso a Internet o scade 2357
VPC: la connessione TCP o UDP fallisce a intermittenza 2358
VPC: è necessario accedere alla funzione Servizi AWS senza utilizzare Internet 2358
VPC: raggiunto il limite dell'interfaccia di rete elastica 2359
EC2: Interfaccia di rete elastica con tipo «lambda» 2359
DNS: impossibile connettersi agli host con UNKNOWNHOSTEXCEPTION 2359
Applicazioni di esempio 2360
Lavorare con AWS SDKs 2363
Esempi di codice 2365
Nozioni di base 2367
Hello Lambda 2368
Informazioni di base 2378
Azioni 2515
Scenari 2652
Confermare automaticamente gli utenti noti con una funzione Lambda 2653
Migrare automaticamente gli utenti noti con una funzione Lambda 2693
Creazione di una REST API per monitorare i dati COVID-19 2717
Creazione di una REST API per la libreria di prestiti 2718
Creazione di un’applicazione di messaggistica 2719

Creazione di un'applicazione serverless per gestire foto 2720
Creazione di un'applicazione di chat websocket 2724
Crea un'applicazione per analizzare il feedback dei clienti 2725
Invocazione di una funzione Lambda da un browser 2731
Trasformare i dati con S3 Object Lambda 2732
Utilizzo di Gateway API per invocare una funzione Lambda 2732
Utilizzo di Step Functions per invocare le funzioni Lambda 2734
Utilizzo degli eventi pianificati per invocare una funzione Lambda 2735
Utilizzare l'API Neptune per eseguire query sui dati del grafico 2737
Scrivere i dati di attività personalizzate con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito tramite un SDK 2738
Esempi serverless 2761
Connessione a un database Amazon RDS in una funzione Lambda 2761
Invocare una funzione Lambda da un trigger Kinesis 2780
Invocare una funzione Lambda da un trigger DynamoDB 2791
Invocare una funzione Lambda da un trigger Amazon DocumentDB 2800
Invocare una funzione Lambda da un trigger Amazon MSK 2813
Invocazione di una funzione Lambda da un trigger Amazon S3 2822
Richiamo di una funzione Lambda da un trigger Amazon SNS 2834
Invocazione di una funzione Lambda da un trigger Amazon SQS 2843
Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis 2852
Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB ... 2865
Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS 2876
AWS contributi della comunità 2886
Creare e testare un'applicazione serverless 2886
Quote di Lambda 2889
Calcolo e archiviazione 2890
Configurazione, implementazione ed esecuzione della funzione 2892
Richieste API Lambda 2894
Altri servizi 2896
Cronologia dei documenti 2897
Aggiornamenti precedenti 2927
..... mmcmxxxiv

Che cos'è AWS Lambda?

AWS Lambda è un servizio di elaborazione che esegue codice senza la necessità di gestire server. Il codice viene eseguito, scalabile automaticamente verso l'alto e verso il basso, con pay-per-use prezzi. Per iniziare, consulta Crea la tua prima funzione.

Puoi utilizzare Lambda per:

- Elaborazione di file: elabora automaticamente i file quando vengono caricati su Amazon Simple Storage Service. Per ulteriori informazioni, consulta gli esempi di elaborazione file.
- Flussi di lavoro a lunga durata: utilizza funzioni Lambda durevoli per creare flussi di lavoro in più fasi con stato che possono durare fino a un anno. Perfetto per l'elaborazione degli ordini, i flussi di lavoro di approvazione, human-in-the-loop i processi e le pipeline di dati complesse che devono ricordare i propri progressi.
- Esempi di operazioni e integrazione del database: rispondi alle modifiche del database e automatizza i flussi di lavoro dei dati. Per ulteriori informazioni, consulta gli esempi di database.
- Attività pianificate e periodiche: esegui operazioni automatizzate secondo una pianificazione regolare utilizzando. EventBridge Per ulteriori informazioni, consulta gli esempi di attività pianificate.
- Elaborazione di flussi: elabora flussi di dati in tempo reale per analisi e monitoraggio. Per ulteriori informazioni, consulta Flussi di dati Kinesis.
- Applicazioni Web: crea app Web scalabili che si adattano automaticamente alla domanda.
- Backend mobili: crea backend API sicuri per applicazioni mobili e Web.
- backend IoT: gestisci richieste di API Web, dispositivi mobili, IoT e di terze parti. Per ulteriori informazioni, consulta IoT.

Per informazioni sui prezzi, consultare Prezzi di AWS Lambda.

Come funziona Lambda

Quando si utilizza Lambda, si è responsabili solo del proprio codice. Lambda esegue il codice su un'infrastruttura di elaborazione ad alta disponibilità e gestisce tutte le risorse di calcolo, compresa la manutenzione del server e del sistema operativo, il provisioning della capacità, il dimensionamento automatico e la registrazione di log.

Poiché Lambda è un servizio di elaborazione serverless basato sugli eventi, utilizza un paradigma di programmazione diverso rispetto alle applicazioni Web tradizionali. Nel modello seguente viene illustrato il funzionamento di Lambda:

1. Scrivi e organizzi il codice in funzioni Lambda, che sono gli elementi costitutivi di base che usi per creare un'applicazione Lambda.
2. Puoi controllare la sicurezza e l'accesso tramite le autorizzazioni Lambda, utilizzando i ruoli di esecuzione per gestire i AWS servizi con cui le tue funzioni possono interagire e quali politiche delle risorse possono interagire con il tuo codice.
3. Le sorgenti e AWS i servizi degli eventi attivano le funzioni Lambda, trasmettendo i dati degli eventi in formato JSON, che le funzioni elaborano (incluse le mappature delle sorgenti degli eventi).
4. Lambda esegue il codice con runtime specifici del linguaggio (come Node.js e Python) in ambienti di esecuzione che raggruppano runtime, livelli ed estensioni.

> **Tip**
>
> Per scoprire come creare soluzioni serverless, consulta la Guida allo sviluppo serverless.

Funzionalità principali

Configura, controlla e implementa applicazioni sicure:

- Variabili di ambiente modifica il comportamento delle applicazioni senza nuove implementazioni di codice.
- Versioni testa nuove funzionalità in modo sicuro mantenendo ambienti di produzione stabili.
- Livelli Lambda ottimizza il riutilizzo e la manutenzione del codice condividendo componenti comuni tra più funzioni.
- Firma del codice applica la conformità alla sicurezza assicurando che solo il codice approvato raggiunga i sistemi di produzione.

Scalabilità e prestazioni affidabili:

- I controlli di simultaneità e scalabilità gestiscono con precisione la reattività delle applicazioni e l'utilizzo delle risorse durante i picchi di traffico.

- Lambda SnapStart riduce significativamente i tempi di avvio a freddo. Lambda SnapStart può fornire prestazioni di avvio anche inferiori al secondo, in genere senza modifiche al codice della funzione.
- Streaming delle risposte ottimizza le prestazioni delle funzioni fornendo payload di grandi dimensioni in modo incrementale per l'elaborazione in tempo reale.
- Immagini di container impacchetta le funzioni con dipendenze complesse utilizzando flussi di lavoro container.

Collega e integra senza problemi:

- Le reti VPC proteggono risorse sensibili e servizi interni.
- Integrazione File system che condivide dati persistenti e gestisce le operazioni con stato durante le invocazioni di funzione.
- Funzione URLscrea endpoint APIs e rivolti al pubblico senza servizi aggiuntivi.
- Estensioni Lambda aumenta le funzioni con strumenti operativi, di monitoraggio e di sicurezza.

Informazioni correlate

- Per ulteriori informazioni sul funzionamento di Lambda, consulta Come funziona Lambda.
- Per iniziare a utilizzare Lambda, consulta Crea la tua prima funzione Lambda.
- Per un elenco di applicazioni di esempio, vedere Guida introduttiva ad applicazioni e modelli di esempio.

Come funziona Lambda

Le funzioni Lambda sono gli elementi costitutivi di base utilizzati per creare applicazioni Lambda. Per scrivere funzioni, è essenziale comprendere i concetti e i componenti fondamentali che costituiscono il modello di programmazione Lambda. Questa sezione ti guiderà attraverso gli elementi fondamentali che devi conoscere per iniziare a creare applicazioni serverless con Lambda.

- Funzioni Lambda e gestori di funzioni- Una funzione Lambda è un piccolo blocco di codice che viene eseguito in risposta agli eventi. Le funzioni possono essere standard (fino a 15 minuti) o durevoli (fino a un anno). Le funzioni sono gli elementi costitutivi di base utilizzati per creare applicazioni. I gestori di funzioni sono il punto di ingresso per gli oggetti evento elaborati dal codice funzione Lambda.

- Ambiente di esecuzione Lambda e runtime: gli ambienti di esecuzione Lambda gestiscono le risorse necessarie per eseguire la funzione. Per funzioni durevoli, l'ambiente di esecuzione include funzionalità automatiche di gestione dello stato e checkpoint. I runtime sono gli ambienti specifici del linguaggio in cui vengono eseguite le funzioni.
- Eventi e trigger- Altri Servizi AWS possono richiamare le tue funzioni in risposta a eventi specifici. Per quanto riguarda le funzioni durevoli, gli eventi possono anche innescare la ripresa dei flussi di lavoro in pausa.
- Autorizzazioni e ruoli Lambda- Controlla chi può accedere alle tue funzioni e con quali altre Servizi AWS funzioni possono interagire. Le funzioni durevoli richiedono autorizzazioni aggiuntive per la gestione dello stato e l'esecuzione estesa.

> **Tip**
>
> Se vuoi iniziare a comprendere lo sviluppo serverless più in generale, consulta Informazioni sulla differenza tra sviluppo tradizionale e serverless nella Guida per gi sviluppatori serverless AWS .

Funzioni Lambda e gestori di funzioni

In Lambda, le funzioni sono gli elementi costitutivi fondamentali utilizzati per creare applicazioni. Una funzione Lambda è una parte di codice che viene eseguita in risposta a eventi, ad esempio un utente che fa clic su un pulsante in un sito Web o un file che viene caricato in un bucket Amazon Simple Storage Service (Amazon S3). Con le funzioni durevoli, il codice può mettere in pausa l'esecuzione tra i passaggi, mantenendo automaticamente lo stato, il che le rende ideali per flussi di lavoro di lunga durata come l'elaborazione degli ordini o la moderazione dei contenuti. Si può pensare a una funzione come a una sorta di programma autonomo con le seguenti proprietà.

Un gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando una funzione viene eseguita in risposta a un evento, Lambda esegue il gestore delle funzioni. I dati sull'evento che ha causato l'esecuzione della funzione vengono passati direttamente al gestore. Sebbene il codice di una funzione Lambda possa contenere più di un metodo o una funzione, le funzioni Lambda possono avere un solo gestore.

Per creare una funzione Lambda, crea un pacchetto di implementazione contenente il codice funzione e le rispettive dipendenze. Lambda supporta due tipi di pacchetti di implementazione: archivi di file .zip e immagini di container.

- Una funzione ha un lavoro o uno scopo specifico
- Vengono eseguite solo quando necessario in risposta a eventi specifici
- Smettono automaticamente di funzionare al termine

Ambiente di esecuzione Lambda e runtime

Le funzioni Lambda vengono eseguite all'interno di un ambiente di esecuzione sicuro e isolato che Lambda gestisce per te. Per funzioni durevoli, l'ambiente di esecuzione include componenti aggiuntivi per la gestione dello stato e il coordinamento del flusso di lavoro. L'ambiente di esecuzione gestisce i processi e le risorse necessari per eseguire la funzione. Quando una funzione viene invocata per la prima volta, Lambda crea un nuovo ambiente di esecuzione in cui eseguire la funzione. Al termine dell'esecuzione della funzione, Lambda non interrompe immediatamente l'ambiente di esecuzione; se la funzione viene invocata nuovamente, Lambda può riutilizzare l'ambiente di esecuzione esistente.

L'ambiente di esecuzione Lambda contiene inoltre un runtime, un ambiente specifico al linguaggio che inoltra le informazioni sugli eventi e le risposte tra Lambda e la funzione. Lambda offre una serie di runtime gestiti per i linguaggi di programmazione più diffusi, oppure puoi crearne di tuoi.

Per i runtime festiti, Lambda applica automaticamente aggiornamenti e patch alle funzioni usando il runtime.

Eventi e trigger

Puoi anche richiamare una funzione Lambda direttamente utilizzando la console Lambda o uno dei AWS Software AWS CLIDevelopment Kit (). SDKs In un'applicazione di produzione è più comune che una funzione venga richiamata da un'altra Servizio AWS in risposta a un particolare evento. Ad esempio, potresti voler eseguire una funzione ogni volta che un elemento viene aggiunto a una tabella Amazon DynamoDB.

Per fare in modo che la tua funzione risponda agli eventi, configura un trigger. Un trigger collega la funzione a un'origine eventi; la funzione può avere più trigger. Quando si verifica un evento, Lambda riceve i dati dell'evento come documento JSON e li converte in un oggetto che il codice può elaborare. Puoi definire il seguente formato JSON per il tuo evento; il runtime Lambda converte questo JSON in un oggetto prima di passarlo al gestore della funzione.

Example evento Lambda personalizzato

```
{
```

```
  "Location": "SEA",
  "WeatherData":{
    "TemperaturesF":{
      "MinTempF": 22,
      "MaxTempF": 78
    },
    "PressuresHPa":{
      "MinPressureHPa": 1015,
      "MaxPressureHPa": 1027
    }
  }
}
```

Con servizi di streaming e coda come Amazon Kinesis o Amazon SQS, Lambda utilizza uno strumento di mappatura dell'origine degli eventi anziché un trigger standard. Gli strumenti di mappatura dell'origine degli eventi analizzano l'origine alla ricerca di dati nuovi, raggruppano i record in batch e invocano quindi la funzione con gli eventi in batch. Per ulteriori informazioni, consulta In che modo gli strumenti di mappatura dell'origine degli eventi differiscono dai trigger diretti.

Per capire come funziona un trigger, inizia completando il tutorial Uso di un trigger Amazon S3 oppure, per una panoramica generale sull'uso dei trigger e le istruzioni sulla creazione di un trigger utilizzando la console Lambda, consulta Integrazione con altri servizi.

Autorizzazioni e ruoli Lambda

Per Lambda, è necessario configurare due tipi principali di autorizzazioni:

- Autorizzazioni di cui la tua funzione ha bisogno per accedere ad altre Servizi AWS
- Autorizzazioni di cui hanno Servizi AWS bisogno gli altri utenti per accedere alla tua funzione

Le sezioni seguenti descrivono entrambi questi tipi di autorizzazione e illustrano le best practice per applicare le autorizzazioni di privilegio minimo.

Autorizzazioni per le funzioni di accesso ad altre risorse AWS

Le funzioni Lambda spesso devono accedere ad altre AWS risorse ed eseguire azioni su di esse. Ad esempio, una funzione può leggere elementi da una tabella DynamoDB, archiviare un oggetto in un bucket S3 o scrivere in una coda Amazon SQS. Per concedere alle funzioni le autorizzazioni necessarie per eseguire queste azioni, puoi utilizzare un ruolo di esecuzione.

Un ruolo di esecuzione Lambda è un tipo speciale di ruolo AWS Identity and Access Management (IAM), un'identità che crei nel tuo account a cui sono associate autorizzazioni specifiche definite in una policy.

Ogni funzione Lambda deve avere un ruolo di esecuzione e un singolo ruolo può essere utilizzato da più di una funzione. Quando viene invocata una funzione, Lambda assume il ruolo di esecuzione della funzione e riceve l'autorizzazione a intraprendere le azioni definite nella policy del ruolo.

Quando crei una funzione nella console Lambda, Lambda crea automaticamente un ruolo di esecuzione per la funzione. La politica del ruolo fornisce alla tua funzione le autorizzazioni di base per scrivere output di log su Amazon CloudWatch Logs. Per autorizzare la funzione a eseguire azioni su altre AWS risorse, devi modificare il ruolo per aggiungere autorizzazioni aggiuntive. Il modo più semplice per aggiungere autorizzazioni consiste nell'utilizzare una policy AWS gestita. Le policy gestite vengono create e amministrate da AWS e forniscono autorizzazioni per molti casi d'uso comuni. Ad esempio, se la tua funzione esegue operazioni CRUD su una tabella DynamoDB, puoi aggiungere AmazonDynamoDBFullla politica di accesso al tuo ruolo.

Autorizzazioni per altri utenti e risorse per accedere alla tua funzione

Per concedere altre Servizio AWS autorizzazioni per accedere alla funzione Lambda, si utilizza una politica basata sulle risorse. In IAM, le policy basate sulle risorse sono collegate a una risorsa (in questo caso, la funzione Lambda) e definiscono chi può accedere alla risorsa e quali azioni può eseguire.

Affinché un'altra Servizio AWS persona possa richiamare la tua funzione tramite un trigger, la politica basata sulle risorse della funzione deve concedere al servizio l'autorizzazione a utilizzare l'azione. `lambda:InvokeFunction` Se crei il trigger utilizzando la console, Lambda aggiunge automaticamente questa autorizzazione per te.

Per concedere l'autorizzazione ad altri AWS utenti ad accedere alla tua funzione, puoi definirla nella politica basata sulle risorse della tua funzione esattamente come per un'altra o risorsa. Servizio AWS Puoi anche utilizzare una policy basata sull'identità associata all'utente.

Best practice per le autorizzazioni Lambda

Quando imposti le autorizzazioni usando le policy IAM, la best practice di sicurezza consiste nel concedere le autorizzazioni richieste per eseguire un'attività. Ciò è noto come principio di privilegio minimo. Per iniziare a concedere le autorizzazioni per la tua funzione, puoi scegliere di utilizzare una politica gestita. AWS Le policy gestite possono essere il modo più rapido e semplice per concedere

le autorizzazioni per eseguire un'attività, ma possono includere anche altre autorizzazioni non necessarie. Man mano che passi dallo sviluppo iniziale al test e alla produzione, ti consigliamo di ridurre le autorizzazioni solo a quelle necessarie definendo policy personalizzate gestite dal cliente.

Lo stesso principio si applica quando si concedono autorizzazioni per accedere alla tua funzione utilizzando una ploicy basata sulle risorse. Ad esempio, se desideri autorizzare Amazon S3 a richiamare la tua funzione, è consigliabile limitare l'accesso ai singoli bucket, o ai bucket in particolare Account AWS, anziché concedere autorizzazioni generali al servizio S3.

Esecuzione di codice con Lambda

Quando scrivi una funzione Lambda, crei codice che verrà eseguito in un ambiente serverless unico. Comprendere come Lambda esegue effettivamente il codice implica due aspetti chiave: il modello di programmazione che definisce il modo in cui il codice interagisce con Lambda e il ciclo di vita dell'ambiente di esecuzione che determina il modo in cui Lambda gestisce l'ambiente di runtime del codice.

Il modello di programmazione Lambda

Il modello di programmazione funziona come un insieme comune di regole per il modo in cui Lambda interagisce con il tuo codice, indipendentemente dal fatto che tu stia scrivendo in Python, Java o in qualsiasi altro linguaggio supportato. Il modello di programmazione include il runtime e il gestore.

Per le funzioni standard:

1. Lambda riceve un evento.
2. Lambda utilizza il runtime per preparare l'evento in un formato utilizzabile dal codice.
3. Il runtime invia l'evento formattato al gestore.
4. Il tuo gestore elabora l'evento utilizzando il codice che hai scritto.

Per funzioni durevoli:

1. Lambda riceve un evento
2. Il runtime prepara sia l'evento che DurableContext
3. Il tuo gestore può:
   - Fasi del processo con checkpoint automatico
   - Metti in pausa l'esecuzione senza consumare risorse
   - Riprendi dall'ultimo checkpoint riuscito
   - Mantieni lo stato tra i passaggi

Essenziale per questo modello è il gestore al quale cui Lambda invia gli eventi che devono essere elaborati dal codice. Consideralo come il punto di ingresso per il tuo codice. Quando Lambda riceve un evento, lo trasmette al gestore insieme ad alcune informazioni di contesto. Il gestore esegue quindi il codice per elaborare questi eventi: ad esempio, potrebbe leggere un file quando viene

caricato su Amazon S3, analizzare un'immagine o aggiornare un database. Una volta che il codice ha terminato l'elaborazione di un evento, il gestore è pronto per elaborare quello successivo.

Il modello di esecuzione Lambda

Mentre il modello di programmazione definisce il modo in cui Lambda interagisce con il codice, l'ambiente di esecuzione è il luogo in cui Lambda esegue effettivamente la tua funzione: è uno spazio di calcolo sicuro e isolato creato appositamente per la tua funzione.

Ogni ambiente segue un ciclo di vita che varia tra funzioni standard e durevoli:

Funzioni standard (fino a 15 minuti):

1. Inizializzazione: configurazione dell'ambiente e caricamento del codice
2. Invocazione: esecuzione singola del codice della funzione
3. Arresto: pulizia dell'ambiente

Funzioni durevoli (fino a 1 anno):

1. Inizializzazione: configurazione dell'ambiente e dello stato durevole
2. Invocazione: passaggi multipli con checkpoint automatico
3. Stati di attesa: sospende l'esecuzione senza consumo di risorse
4. Riprendi: riavvia dall'ultimo checkpoint
5. Arresto: pulizia dello stato durevole

Questo ambiente gestisce aspetti importanti dell'esecuzione della funzione. Fornisce memoria alla funzione, insieme a una directory `/tmp` per l'archiviazione temporanea. Per Durable Functions, gestisce anche:

- Persistenza automatica dello stato tra i passaggi
- Archiviazione e ripristino dei checkpoint
- Attendi il coordinamento statale
- Monitoraggio dei progressi nelle esecuzioni di lunga durata

Comprendere il modello di programmazione Lambda

Lambda offre due modelli di programmazione: funzioni standard che durano fino a 15 minuti e Durable Functions che possono funzionare fino a un anno. Sebbene entrambi condividano i concetti fondamentali, Durable Functions aggiunge funzionalità per flussi di lavoro di lunga durata e caratterizzati da uno stato.

Lambda fornisce un modello di programmazione comune a tutti i tempi di esecuzione. Il modello di programmazione definisce l'interfaccia tra il codice e il sistema Lambda. Comunica a Lambda il punto di ingresso alla funzione definendo un gestore nella configurazione della funzione. Il runtime trasferisce al gestore gli oggetti che contengono l'evento di chiamata e il contesto, ad esempio il nome della funzione e l'ID della richiesta.

Per Durable Functions, il gestore riceve anche un oggetto che fornisce: DurableContext

- Funzionalità di checkpoint tramite step ()
- Gestione dello stato di attesa tramite wait () e waitForCallback ()
- Persistenza automatica dello stato tra le chiamate

Quando l'handler termina l'elaborazione del primo evento, il runtime ne invia un altro. Per Durable Functions, il gestore può sospendere l'esecuzione tra i passaggi e Lambda salverà e ripristinerà automaticamente lo stato quando la funzione riprende. La classe della funzione rimane in memoria, quindi i client e le variabili dichiarate al di fuori del metodo del gestore nel codice di inizializzazione possono essere riutilizzati. Per risparmiare tempo di elaborazione sugli eventi successivi, crea risorse riutilizzabili come i client SDK AWS durante l'inizializzazione. Una volta inizializzata, ogni istanza della tua funzione può elaborare migliaia di richieste.

La funzione ha anche accesso allo spazio di archiviazione locale nella directory `/tmp`, una cache transitoria che può essere utilizzata per più invocazioni. Per ulteriori informazioni, consulta Ambiente di esecuzione.

Quando la traccia AWS X-Ray è abilitata, il runtime registra sottosegmenti separati per l'inizializzazione e l'esecuzione.

Il runtime acquisisce l'output di registrazione dalla tua funzione e lo invia ad Amazon CloudWatch Logs. Oltre a registrare l'output della funzione, il runtime registra anche le voci all'avvio e alla fine dell'invocazione. Questo include un log del report con l'ID della richiesta, la durata fatturata, la durata di inizializzazione e altri dettagli. Se la funzione genera un errore, il runtime restituisce tale errore all'invoker.

> **Note**
>
> La registrazione è soggetta alle quote di log. CloudWatch È possibile perdere i dati di log a causa del throttling o, in alcuni casi, quando un'istanza della funzione viene interrotta.

Principali differenze per le funzioni durevoli:

- Lo stato viene mantenuto automaticamente tra i passaggi
- Le funzioni possono sospendere l'esecuzione senza consumare risorse
- I passaggi vengono ritentati automaticamente in caso di errore
- I progressi vengono tracciati tramite checkpoint

Lambda dimensiona la funzione eseguendone altre istanze in base all'aumento della domanda e interrompendo le istanze in base alla diminuzione della domanda. Questo modello porta a variazioni nell'architettura delle applicazioni, come ad esempio:

- Salvo diversa indicazione, le richieste in entrata possono essere elaborate fuori ordine o simultaneamente.
- Non fare affidamento sulla durata delle istanze della funzione, ma archivia lo stato dell'applicazione in altri servizi.
- Utilizza lo storage locale e gli oggetti a livello di classe per migliorare le prestazioni, mantenendo al minimo le dimensioni del pacchetto di distribuzione e la quantità di dati trasferiti nell'ambiente di esecuzione.

Per un'introduzione pratica al modello di programmazione preferito, consulta i seguenti capitoli.

- Compilazione di funzioni Lambda con Node.js
- Compilazione di funzioni Lambda con Python
- Compilazione di funzioni Lambda con Ruby
- Compilazione di funzioni Lambda con Java
- Compilazione di funzioni Lambda con Go
- Compilazione di funzioni Lambda con C#
- Creazione di funzioni Lambda con PowerShell

Comprendere il ciclo di vita dell'ambiente di esecuzione Lambda

Gli ambienti di esecuzione Lambda supportano sia funzioni standard (fino a 15 minuti) che funzioni durevoli (fino a un anno). Sebbene entrambi condividano lo stesso ciclo di vita di base, Durable Functions aggiunge funzionalità di gestione dello stato per flussi di lavoro di lunga durata.

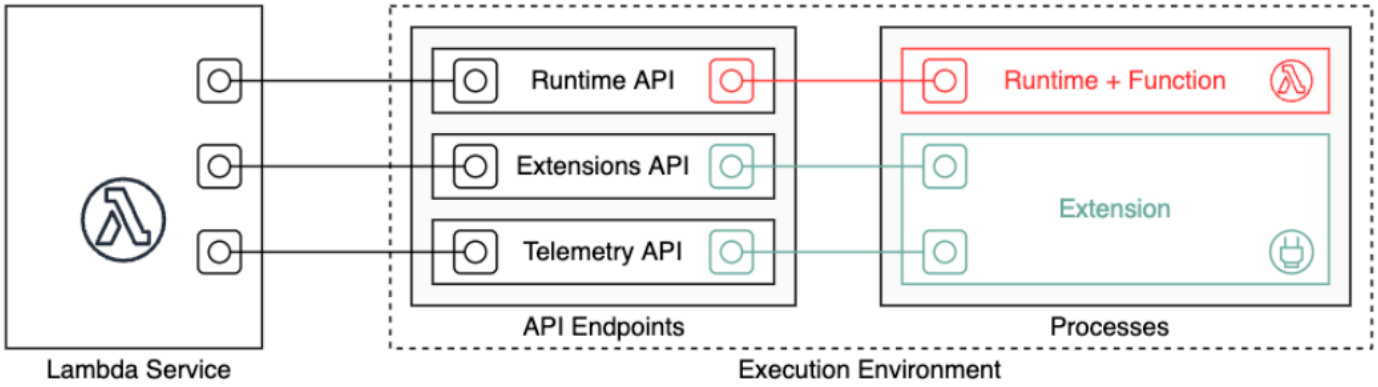
Lambda richiama la funzione in un ambiente di esecuzione, che fornisce un ambiente di runtime sicuro e isolato. L'ambiente di esecuzione gestisce le risorse necessarie per eseguire la funzione. L'ambiente di esecuzione fornisce inoltre il supporto del ciclo di vita per il runtime della funzione e per tutte le estensioni esterne associate alla funzione.

Per Durable Functions, l'ambiente di esecuzione include componenti aggiuntivi per:

- Persistenza dello stato tra i passaggi
- Gestione dei checkpoint
- Aspetta il coordinamento dello stato
- Monitoraggio dei progressi

> **Ambiente di esecuzione Lambda Managed Instances**
>
> Se si utilizzano istanze gestite Lambda, l'ambiente di esecuzione presenta differenze importanti rispetto alle funzioni Lambda (predefinite). Le istanze gestite supportano chiamate simultanee, utilizzano un modello di ciclo di vita diverso e vengono eseguite su un'infrastruttura di proprietà del cliente. Per informazioni dettagliate sull'ambiente di esecuzione delle istanze gestite, vedere. Comprendere l'ambiente di esecuzione di Lambda Managed Instances

Il runtime della funzione comunica con Lambda e usa l'API Runtime. Le estensioni comunicano con Lambda utilizzando le API estensioni. Le estensioni possono anche ricevere messaggi di log e altra telemetria dalla funzione tramite l'API di telemetria.



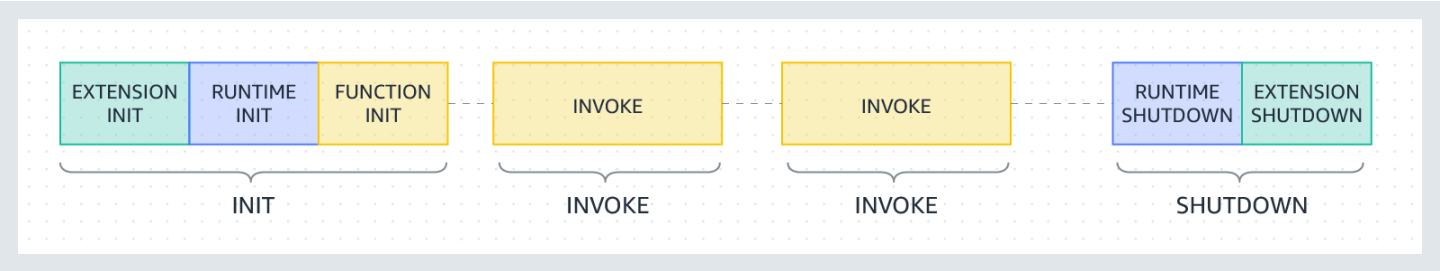
Quando si crea la funzione Lambda, si specificano le informazioni di configurazione, ad esempio la quantità di memoria disponibile e il tempo massimo di esecuzione consentito per la funzione. Lambda utilizza queste informazioni per impostare l'ambiente di esecuzione.

Il runtime della funzione e ogni estensione esterna sono processi eseguiti all'interno dell'ambiente di esecuzione. Le autorizzazioni, le risorse, le credenziali e le variabili di ambiente sono condivise tra la funzione e le estensioni.

Argomenti

- Ciclo di vita dell'ambiente di esecuzione Lambda
- Avvii a freddo e latenza
- Riduzione degli avvii a freddo con la simultaneità fornita
- Ottimizzazione dell'inizializzazione statica

Ciclo di vita dell'ambiente di esecuzione Lambda



Ogni fase inizia con un evento che Lambda invia al runtime e a tutte le estensioni registrate. Il runtime e ogni estensione indicano il completamento inviando una richiesta API `Next`. Lambda congela l'ambiente di esecuzione quando il runtime e ogni estensione sono stati completati e non ci sono eventi in sospeso.

Le fasi del ciclo di vita di Durable Functions includono:

- Init: inizializzazione standard più configurazione durevole dello stato
- Invoke: può includere esecuzioni in più fasi con checkpoint automatico
- Attendi: la funzione può mettere in pausa l'esecuzione senza consumare risorse
- Riprendi: la funzione viene riavviata dall'ultimo checkpoint
- Chiusura: ripulitura dello stato e delle risorse durevoli

Argomenti

- Fase di init
- Errori durante la fase di inizializzazione
- Fase di ripristino (solo Lambda SnapStart)
- Invoca fase
- Errori durante la fase di richiamo
- Fase di arresto

Fase di init

Nella fase `Init`, Lambda esegue tre incarichi:

- Avvia tutte le estensioni (`Extension init`)
- Esegue il bootstrap del runtime (`Runtime init`)
- Esegue il codice statico della funzione (`Function init`)
- Esegui qualsiasi hook di runtime before-checkpoint (solo Lambda) SnapStart

La fase `Init` termina quando il runtime e tutte le estensioni segnalano che sono pronti inviando una richiesta `Next` API. La fase `Init` è limitata a 10 secondi. Se tutte e tre le attività non vengono completate entro 10 secondi, Lambda ritenta la fase `Init` al momento della prima chiamata di funzione con timeout della funzione configurata.

Quando Lambda SnapStart è attivato, la fase `Init` si verifica quando si pubblica una versione della funzione. Lambda salva uno snapshot della memoria e dello stato del disco dell'ambiente di esecuzione inizializzato, mantiene lo snapshot crittografato e lo memorizza nella cache per l'accesso

a bassa latenza. Se disponi di un hook di runtime prima del checkpoint, il codice viene eseguito alla fine della fase `Init`.

> **Note**
>
> Il timeout di 10 secondi non si applica alle funzioni che utilizzano la concorrenza fornita SnapStart o le istanze gestite Lambda. Per le funzioni di provisioning, concorrenza e istanze gestite SnapStart, il codice di inizializzazione può essere eseguito per un massimo di 15 minuti. Il limite di tempo è 130 secondi o il timeout della funzione configurato (massimo 900 secondi), a seconda di quale dei due valori sia più elevato.

Quando utilizzi la simultaneità fornita, Lambda inizializza l'ambiente di esecuzione quando configuri le impostazioni del PC per una funzione. Lambda, inoltre, garantisce che gli ambienti di esecuzione inizializzati siano sempre disponibili prima delle invocazioni. Potrebbero verificarsi lacune tra l'invocazione della funzione e le fasi di inizializzazione. A seconda del runtime della funzione e della configurazione della memoria, può anche verificarsi una latenza delle variabili nella prima invocazione in un ambiente di esecuzione inizializzato.

Per le funzioni che utilizzano la simultaneità on demand, Lambda può inizializzare occasionalmente gli ambienti di esecuzione prima delle richieste di invocazione. Quando ciò si verifica, puoi riscontrare una lacuna temporale tra le fasi di inizializzazione e invocazione della funzione. È preferibile non acquisire dipendenza da questo comportamento.

Errori durante la fase di inizializzazione

Se una funzione si arresta in modo anomalo o si verifica un timeout durante la fase `Init`, Lambda emette informazioni sull'errore nel log `INIT_REPORT`.

Example — Log INIT_REPORT per il timeout

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

Example — Log INIT_REPORT per gli errori di estensione

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

Se la `Init` fase ha esito positivo, Lambda non emette il `INIT_REPORT` registro a meno che non SnapStartsia abilitata la concorrenza fornita. SnapStart e le funzioni di concorrenza predisposte

vengono sempre emesse. `INIT_REPORT` Per ulteriori informazioni, consulta Monitoraggio per Lambda SnapStart.

Fase di ripristino (solo Lambda SnapStart)

Quando richiami una SnapStartfunzione per la prima volta e man mano che la funzione aumenta, Lambda riprende i nuovi ambienti di esecuzione dall'istantanea persistente invece di inizializzare la funzione da zero. Se disponi di un hook di runtime dopo il ripristino, il codice viene eseguito alla fine della fase `Restore`. Ti sarà addebitata la durata degli hook di runtime dopo il ripristino. Il runtime deve essere caricato e gli hook di runtime dopo il ripristino devono essere completati entro il limite di timeout (10 secondi). Altrimenti, otterrai un. SnapStartTimeoutException Al termine della fase `Restore`, Lambda chiama il gestore della funzione (Invoca fase).

Errori durante la fase di ripristino

Se la fase `Restore` fallisce, Lambda emette informazioni sull'errore nel log `RESTORE_REPORT`.

Example — Log RESTORE_REPORT per il timeout

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

Example — Log RESTORE_REPORT per gli errori dell'hook di runtime

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

Per ulteriori informazioni sul log `RESTORE_REPORT`, consulta la pagina Monitoraggio per Lambda SnapStart.

Invoca fase

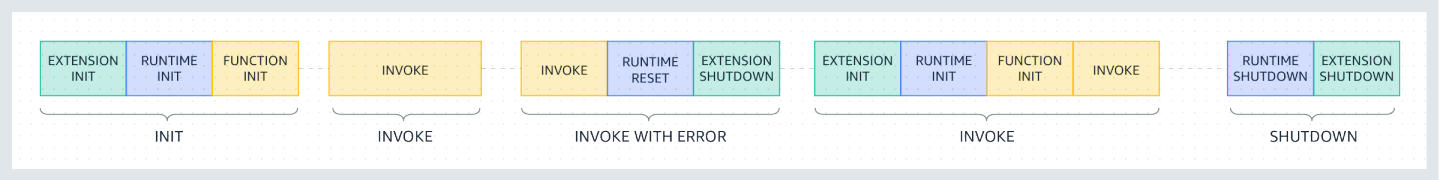
Quando una funzione Lambda viene richiamata in risposta a una richiesta API `Next`, Lambda invia un evento `Invoke` al runtime e a ogni estensione.

L'impostazione di timeout della funzione limita la durata dell'intera fase `Invoke`. Ad esempio, se si imposta il timeout della funzione come 360 secondi, la funzione e tutte le estensioni devono essere completate entro 360 secondi. Si noti che non esiste una fase post-richiamo indipendente. La durata è la somma di tutto il tempo di chiamata (runtime + estensioni) e non viene calcolata fino a quando la funzione e tutte le estensioni non hanno terminato l'esecuzione.

La fase di richiamo termina dopo il runtime e tutte le estensioni segnalano che vengono eseguite inviando una richiesta `Next` API.

Errori durante la fase di richiamo

Se la funzione Lambda si arresta in modo anomalo o si verifica un timeout durante la fase `Invoke`, Lambda reimposta l'ambiente di esecuzione. Il diagramma seguente mostra il comportamento dell'ambiente di esecuzione Lambda in caso di errore di invocazione:



Nel diagramma precedente:

- La prima è la fase INIT e viene eseguita senza errori.
- La seconda è la fase INVOKE e viene eseguita senza errori.
- A un certo punto, supponiamo che la funzione riscontri un errore di richiamo (le cause più comuni includono timeout delle funzioni, errori di runtime, esaurimento della memoria, problemi di connettività VPC, errori di autorizzazione, limiti di concorrenza e vari problemi di configurazione). Per un elenco completo dei possibili errori di chiamata, vedere. the section called “Invocazione” La terza fase, denominata INVOKE WITH ERROR, mostra questo scenario. In questo caso, il servizio Lambda esegue un ripristino. Il reset si comporta come un evento `Shutdown`. Innanzitutto Lambda chiude il runtime, poi invia un evento `Shutdown` a ogni estensione esterna registrata. L'evento include il motivo dell'arresto. Se questo ambiente viene utilizzato per una nuova chiamata, Lambda re-inizializza l'estensione e il runtime insieme alla chiamata successiva.

  Tieni presente che il reset Lambda non cancella il contenuto della directory `/tmp` prima della fase di init successiva. Questo comportamento è coerente con la normale fase di arresto.

  > **Note**
  >
  > AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS.
  > Se la configurazione del registro di sistema della funzione è impostata su testo semplice, questa modifica influisce sui messaggi di registro acquisiti in CloudWatch Logs quando la funzione riscontra un errore di richiamo. Gli esempi seguenti mostrano gli output dei log nei formati vecchi e nuovi.

Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regioni, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia di nuovo formato.

Example CloudWatch Registra l'output dei log (arresto anomalo del runtime o dell'estensione): vecchio stile

```
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 Registra l'output del log (timeout della funzione) - vecchio stile

```
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
```

Il nuovo formato per CloudWatch i log include un `status` campo aggiuntivo nella riga. `REPORT` In caso di arresto anomalo del runtime o dell'estensione, la riga `REPORT` include anche un campo `ErrorType`.

Example CloudWatch Registra l'output dei log (runtime o crash dell'estensione): nuovo stile

```
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 Registra l'output del log (timeout della funzione) - nuovo stile

```
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
```

- La quarta fase rappresenta la fase INVOKE, immediatamente successiva a un errore di chiamata. In questo caso Lambda inizializza nuovamente l'ambiente eseguendo nuovamente la fase INIT. Questa operazione è chiamata inizializzazione soppressa. Quando si verificano init soppressi, Lambda non riporta esplicitamente una fase INIT aggiuntiva nei log. CloudWatch Infatti potresti notare che la durata nella riga REPORT include una durata INIT aggiuntiva + la durata INVOKE. Ad esempio, supponiamo di visualizzare i seguenti log in: 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
  ```

  In questo esempio, la differenza tra i timestamp REPORT e START è di 2,5 secondi. Ciò non corrisponde alla durata riportata di 3022,91 millisecondi, perché la durata della INIT (inizializzazione soppressa) aggiuntiva eseguita da Lambda non viene tenuta in conto. Da questo esempio puoi capire che la fase INVOKE effettiva ha richiesto 2,5 secondi.

  Per ulteriori informazioni su questo comportamento, puoi utilizzare il Accesso ai dati di telemetria in tempo reale per le estensioni tramite l'API Telemetry. L'API di telemetria produce gli eventi `INIT_START`, `INIT_RUNTIME_DONE` e `INIT_REPORT` insieme a `phase=invoke` ogni volta che le inizializzazioni vengono soppresse tra le fasi di invocazione.
- La quinta fase rappresenta la fase SHUTDOWN, che viene eseguita senza errori.

Fase di arresto

Quando Lambda sta per chiudere il runtime, invia un evento `Shutdown` a ciascuna estensione esterna registrata. Le estensioni possono utilizzare questo tempo per le attività di pulizia finali. L'evento `Shutdown` è una risposta a una richiesta API `Next`.

Limite di durata: la durata massima della fase `Shutdown` dipende dalla configurazione delle estensioni registrate:

- 0 ms: funzione senza estensioni registrate
- 500 ms: funzione con estensione interna registrata
- 2.000 ms: funzione con una o più estensioni esterne registrate

Se il runtime o un'estensione non risponde all'evento `Shutdown` entro il limite, Lambda termina il processo utilizzando un segnale `SIGKILL`.

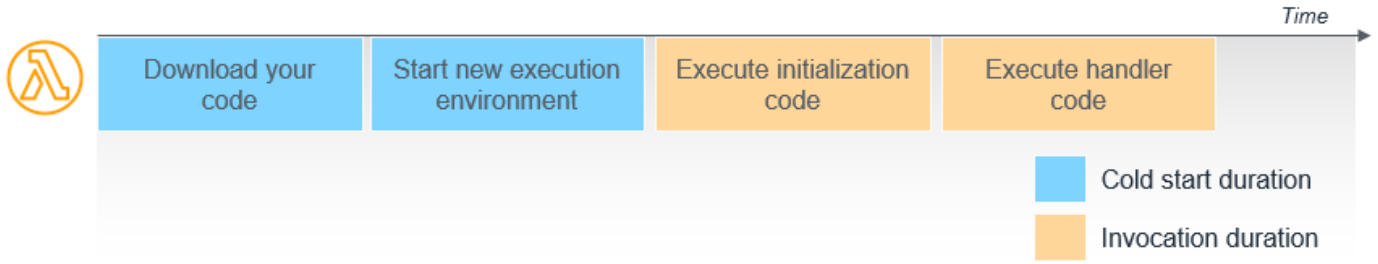
Dopo che la funzione e tutte le estensioni sono state completate, Lambda mantiene l'ambiente di esecuzione per qualche tempo in previsione di un'altra chiamata di funzione. Tuttavia, Lambda termina gli ambienti di esecuzione ogni poche ore per consentire gli aggiornamenti e la manutenzione del runtime, anche per le funzioni che vengono richiamate continuamente. Non si deve dare per scontato che l'ambiente di esecuzione persista all'infinito. Per ulteriori informazioni, consulta Implementazione stateless nelle funzioni.

Quando la funzione viene richiamata nuovamente, Lambda sblocca l'ambiente per il riutilizzo. Il riutilizzo dell'ambiente di esecuzione ha le seguenti implicazioni:

- Gli oggetti dichiarati al di fuori del metodo del gestore della funzione rimangono inizializzati, fornendo ulteriore ottimizzazione quando la funzione viene nuovamente invocata. Ad esempio, se la funzione Lambda stabilisce la connessione a un database, nelle invocazioni successive viene utilizzata la connessione originaria anziché stabilirne un'altra. È consigliabile aggiungere al codice una logica per verificare l'esistenza di una connessione prima che ne venga stabilita una nuova.
- Ogni ambiente di esecuzione fornisce da 512 MB a 10.240 MB con incrementi di 1 MB di spazio su disco nella directory `/tmp`. Il contenuto della directory rimane quando il contesto di esecuzione è bloccato, fornendo così una cache transitoria utilizzabile per più invocazioni. È possibile aggiungere ulteriore codice per verificare se la cache contiene i dati memorizzati. Per ulteriori informazioni sui limiti delle dimensioni della distribuzione, vedere Quote di Lambda.
- Processi in background o callback che sono stati avviati dalla funzione Lambda e che non sono stati completati quando la funzione è terminata; riprendere se Lambda riutilizza l'ambiente di esecuzione. È necessario accertarsi che tutti i processi in background o le callback nel codice vengano completate prima che il codice sia terminato.

Avvii a freddo e latenza

Quando Lambda riceve una richiesta di esecuzione di una funzione tramite l'API Lambda, il servizio prepara innanzitutto un ambiente di esecuzione. Durante questa fase di inizializzazione, il servizio scarica il codice, avvia l'ambiente ed esegue qualsiasi codice di inizializzazione al di fuori del gestore principale. Infine, Lambda esegue il codice del gestore.



In questo diagramma, i primi due passaggi del download del codice e della configurazione dell'ambiente vengono spesso definiti «avvio a freddo». Questo tempo ti viene addebitato e ciò aggiunge latenza alla durata complessiva della chiamata.

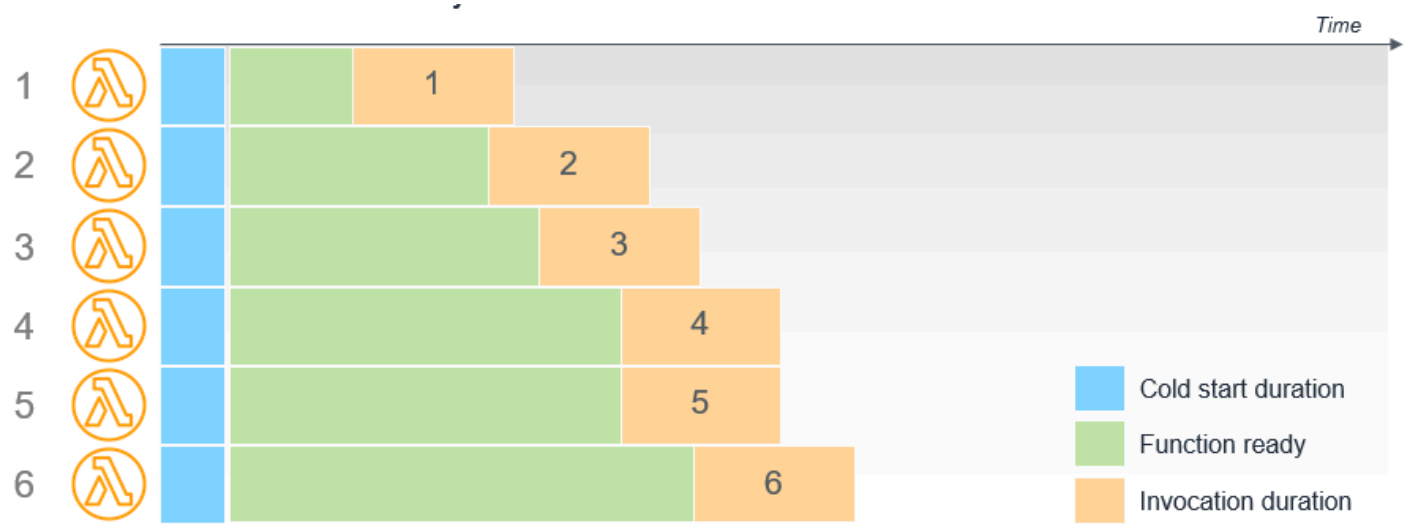
Una volta completata la chiamata, l'ambiente di esecuzione viene bloccato. Per migliorare la gestione delle risorse e le prestazioni, Lambda conserva l'ambiente di esecuzione per un periodo di tempo. Durante questo periodo, se arriva un'altra richiesta per la stessa funzione, Lambda può riutilizzare l'ambiente. Questa seconda richiesta in genere termina più rapidamente, poiché l'ambiente di esecuzione è già completamente configurato. Questo procedimento è chiamato "avvio a caldo".

Gli avviamenti a freddo si verificano in genere in meno dell'1% delle invocazioni. La durata di un avvio a freddo varia da meno di 100 ms a oltre 1 secondo. In generale, gli avviamenti a freddo sono in genere più comuni nelle funzioni di sviluppo e test rispetto ai carichi di lavoro di produzione. Questo perché le funzioni di sviluppo e test vengono in genere richiamate meno frequentemente.

Riduzione degli avvii a freddo con la simultaneità fornita

Se hai bisogno di orari di inizio delle funzioni prevedibili per il tuo carico di lavoro, la soluzione consigliata è la concorrenza fornita per garantire la latenza più bassa possibile. Questa funzionalità preinizializza gli ambienti di esecuzione, riducendo gli avvii a freddo.

Ad esempio, una funzione con una concomitanza assegnata di 6 dispone di 6 ambienti di esecuzione preriscaldati.



Ottimizzazione dell'inizializzazione statica

L'inizializzazione statica avviene prima che il codice dell'handler inizi a essere eseguito in una funzione. Questo è il codice di inizializzazione fornito dall'utente, esterno al gestore principale. Questo codice viene spesso utilizzato per importare librerie e dipendenze, impostare configurazioni e inizializzare connessioni ad altri servizi.

Il seguente esempio in Python mostra l'importazione e la configurazione dei moduli e la creazione del client Amazon S3 durante la fase di inizializzazione, prima che la funzione `lambda_handler` venga eseguita durante la fase di invocazione.

```
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...
```

Il principale fattore di latenza prima dell'esecuzione della funzione proviene dal codice di inizializzazione. Questo codice viene eseguito quando viene creato per la prima volta un nuovo

ambiente di esecuzione. Il codice di inizializzazione non viene eseguito nuovamente se una invocazione utilizza un ambiente di esecuzione caldo. I fattori che influiscono sulla latenza del codice di inizializzazione includono:

- La dimensione del pacchetto di funzioni, in termini di librerie e dipendenze importate e livelli Lambda.
- La quantità di codice e la procedura di inizializzazione.
- Le prestazioni delle librerie e di altri servizi nella configurazione di connessioni e altre risorse.

Esistono diversi passaggi che gli sviluppatori possono adottare per ottimizzare la latenza di inizializzazione statica. Se una funzione ha molti oggetti e connessioni, potresti essere in grado di riprogettare una singola funzione in più funzioni specializzate. Questi sono singolarmente più piccoli e ciascuno ha meno codice di inizializzazione.

È importante che le funzioni importino solo le librerie e le dipendenze di cui hanno bisogno. Ad esempio, se utilizzi solo Amazon DynamoDB AWS nell'SDK, puoi richiedere un servizio individuale anziché l'intero SDK. Confronta i tre esempi seguenti:

```
// 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)
```

L'inizializzazione statica è spesso anche il posto migliore per aprire connessioni al database per consentire a una funzione di riutilizzare le connessioni su più chiamate allo stesso ambiente di esecuzione. Tuttavia, è possibile che nella funzione sia presente un numero elevato di oggetti utilizzati solo in determinati percorsi di esecuzione. In questo caso, puoi caricare lentamente le variabili nell'ambito globale per ridurre la durata dell'inizializzazione statica.

Evita le variabili globali per informazioni specifiche del contesto. Se la funzione ha una variabile globale che viene utilizzata solo per la durata di una singola invocazione e viene reimpostata per l'invocazione successiva, utilizza un ambito di variabile locale per l'handler. In questo modo non solo si evitano perdite di variabili globali tra le invocazioni, ma si migliorano anche le prestazioni dell'inizializzazione statica.

Creazione di architetture basate sugli eventi con Lambda

Un evento è tutto ciò che attiva l'esecuzione di una funzione Lambda. Gli eventi possono attivare una funzione Lambda in due modi: tramite invocazione diretta (push) e mappatura delle sorgenti degli eventi (pull).

Molti AWS servizi possono richiamare direttamente le funzioni Lambda. Questi servizi inviano eventi alla tua funzione Lambda. Gli eventi che attivano le funzioni possono essere praticamente qualsiasi cosa, da una richiesta HTTP tramite API Gateway, una pianificazione gestita da una EventBridge regola, un AWS IoT evento o un evento Amazon S3. Con lo strumento di mappatura dell'origine degli eventi, Lambda recupera (pull) attivamente gli eventi da una coda o da un flusso. Puoi configurare Lambda per verificare la presenza di eventi da un servizio supportato e Lambda gestisce il polling e l'invocazione della tua funzione.

Quando passati alla funzione, gli eventi sono strutturati nel formato JSON. La struttura JSON varia a seconda del servizio che la genera e del tipo di evento. Mentre le chiamate alle funzioni Lambda standard possono durare fino a 15 minuti (o fino a un anno con funzioni durevoli), Lambda è più adatta per chiamate brevi che durano un secondo o meno. Ciò è particolarmente vero per le architetture basate sugli eventi, in cui ogni funzione Lambda viene trattata come un microservizio responsabile dell'esecuzione di una serie ristretta di istruzioni specifiche.

> **Note**
>
> Le architetture basate sugli eventi comunicano tra diversi sistemi utilizzando le reti, che introducono una latenza variabile. Per carichi di lavoro che richiedono una latenza molto bassa, come i sistemi di trading in tempo reale, questo design potrebbe non essere la scelta migliore. Tuttavia, per carichi di lavoro altamente scalabili e disponibili o con schemi di traffico imprevedibili, le architetture basate sugli eventi possono fornire un modo efficace per soddisfare queste esigenze.

Argomenti

- Vantaggi delle architetture basate sugli eventi
- Compromessi delle architetture basate sugli eventi
- Anti-modelli nelle applicazioni basate su Lambda e sugli eventi

Vantaggi delle architetture basate sugli eventi

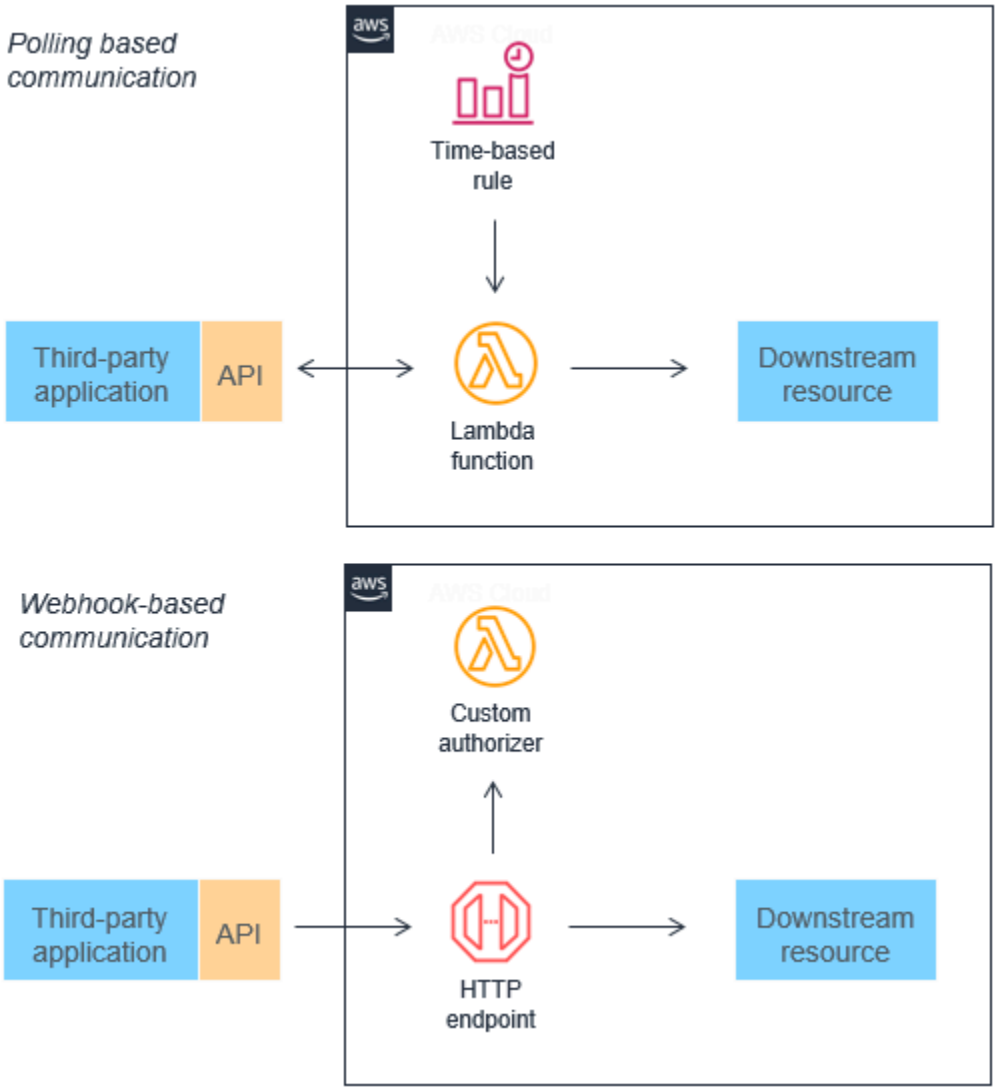
Lambda supporta due metodi di invocazione nelle architetture basate sugli eventi:

1. Invocazione diretta (metodo push): i AWS servizi attivano direttamente le funzioni Lambda. Esempio:
   - Amazon S3 attiva una funzione quando viene caricato un file
   - API Gateway attiva una funzione quando riceve una richiesta HTTP
2. Strumento di mappatura dell'origine degli eventi (metodo pull): Lambda recupera gli eventi e richiama le funzioni. Esempio:
   - Lambda recupera i messaggi da una coda Amazon SQS e invoca una funzione
   - Lambda legge i record da un flusso DynamoDB e invoca una funzione

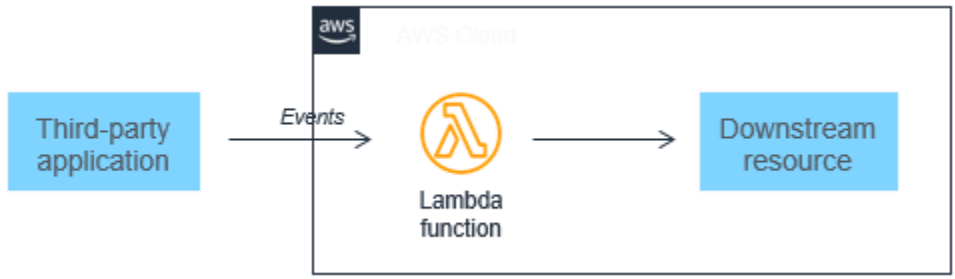
Entrambi i metodi contribuiscono ai vantaggi delle architetture basate sugli eventi, come descritto di seguito.

Sostituzione di polling e webhook con eventi

Molte architetture tradizionali utilizzano meccanismi di polling e webhook per comunicare lo stato tra diversi componenti. Il polling può essere molto inefficiente per il recupero degli aggiornamenti poiché esiste un ritardo tra la disponibilità di nuovi dati e la sincronizzazione con i servizi a valle. I webhook non sono sempre supportati da altri microservizi con cui si desidera integrarsi. Potrebbero inoltre richiedere configurazioni di autorizzazione e autenticazione personalizzate. In entrambi i casi, questi metodi di integrazione sono difficili da scalare su richiesta senza ulteriore lavoro da parte dei team di sviluppo.



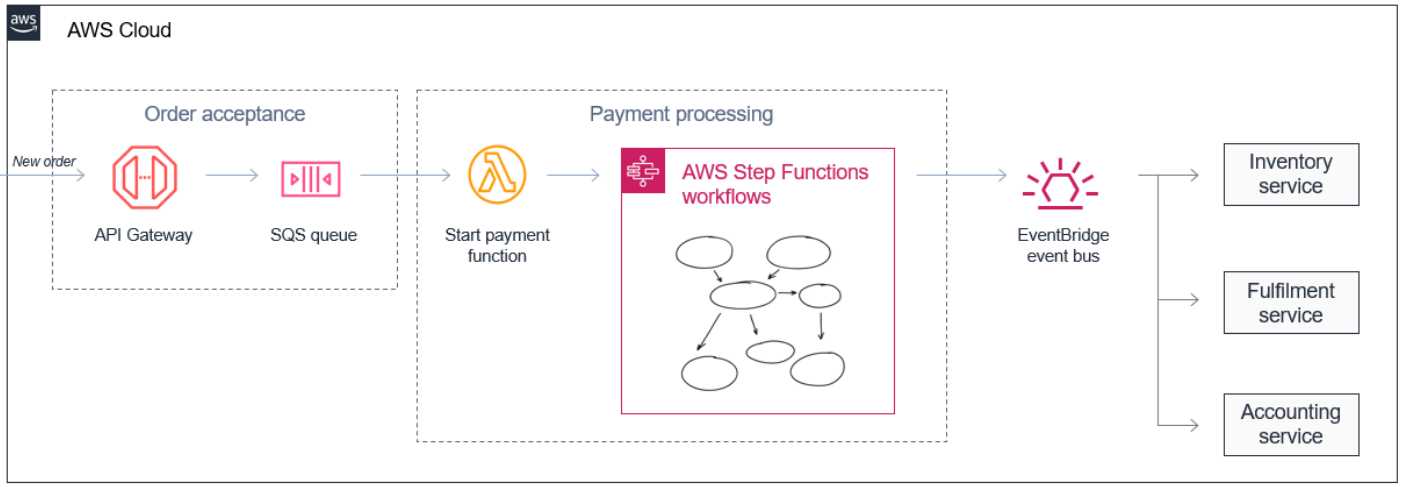
Entrambi questi meccanismi possono essere sostituiti da eventi che possono essere filtrati, indirizzati e trasferiti a valle verso l'utilizzo di microservizi. Questo approccio può comportare un minore consumo di larghezza di banda, un minore utilizzo della CPU e, potenzialmente, una riduzione dei costi. Queste architetture possono anche ridurre la complessità, poiché ogni unità funzionale è più piccola e spesso contiene meno codice.



Le architetture basate sugli eventi possono anche semplificare la progettazione dei near-real-time sistemi, aiutando le organizzazioni ad abbandonare l'elaborazione basata su batch. Gli eventi vengono generati nel momento in cui lo stato dell'applicazione cambia, quindi il codice personalizzato di un microservizio deve essere progettato per gestire l'elaborazione di un singolo evento. Poiché la scalabilità è gestita dal servizio Lambda, questa architettura può gestire aumenti significativi del traffico senza modificare il codice personalizzato. Man mano che gli eventi aumentano, aumenta anche il livello di elaborazione che elabora gli eventi.

Riduzione della complessità

I microservizi consentono a sviluppatori e architetti di semplificare flussi di lavoro complessi. Ad esempio, un monolite di e-commerce può essere suddiviso in processi di accettazione degli ordini e pagamento con servizi di inventario, evasione e contabilità separati. Ciò che potrebbe essere complesso da gestire e orchestrare in un monolite diventa una serie di servizi disaccoppiati che comunicano in modo asincrono con gli eventi.



Questo approccio consente inoltre di assemblare servizi che elaborano i dati a velocità diverse. In questo caso, un microservizio di accettazione degli ordini può archiviare volumi elevati di ordini in entrata inserendo nel buffer i messaggi in una coda Amazon SQS.

Un servizio di elaborazione dei pagamenti, che in genere è più lento a causa della complessità della gestione dei pagamenti, può ricevere un flusso costante di messaggi dalla coda di Amazon SQS. Può orchestrare complesse logiche di gestione dei tentativi e degli errori utilizzando AWS Step Functions e coordinare flussi di lavoro di pagamento attivi per centinaia di migliaia di ordini.

Migliorare la scalabilità e l'estensibilità

I microservizi generano eventi che in genere vengono pubblicati su servizi di messaggistica come Amazon SNS e Amazon SQS. Questi si comportano come un buffer elastico tra i microservizi e aiutano a gestire la scalabilità quando il traffico aumenta. Servizi come Amazon EventBridge possono quindi filtrare e indirizzare i messaggi in base al contenuto dell'evento, come definito nelle regole. Di conseguenza, le applicazioni basate sugli eventi possono essere più scalabili e offrire una maggiore ridondanza rispetto alle applicazioni monolitiche.

Questo sistema è inoltre altamente estensibile e consente ad altri team di estendere le funzionalità e aggiungere funzionalità senza influire sui microservizi di elaborazione degli ordini e di elaborazione dei pagamenti. Pubblicando eventi utilizzando EventBridge, questa applicazione si integra con i sistemi esistenti, come il microservizio di inventario, ma consente anche l'integrazione di qualsiasi applicazione futura come consumatore di eventi. I produttori di eventi non conoscono gli utenti di eventi, il che può contribuire a semplificare la logica dei microservizi.

Compromessi delle architetture basate sugli eventi

Latenza variabile

A differenza delle applicazioni monolitiche, che possono elaborare tutto all'interno dello stesso spazio di memoria su un singolo dispositivo, le applicazioni basate sugli eventi comunicano attraverso le reti. Questo design introduce una latenza variabile. Sebbene sia possibile progettare le applicazioni per ridurre al minimo la latenza, le applicazioni monolitiche possono quasi sempre essere ottimizzate per una latenza inferiore a scapito della scalabilità e della disponibilità.

I carichi di lavoro che richiedono prestazioni costanti a bassa latenza, come le applicazioni di trading ad alta frequenza nelle banche o l'automazione robotica inferiore al millisecondo nei magazzini, non sono buoni candidati per l'architettura basata sugli eventi.

Consistenza finale

Un evento rappresenta un cambiamento di stato e, poiché molti eventi fluiscono attraverso diversi servizi di un'architettura in un dato momento, tali carichi di lavoro alla fine sono spesso coerenti. Ciò rende più complessa l'elaborazione delle transazioni, la gestione dei duplicati o la determinazione dell'esatto stato generale di un sistema.

Molti carichi di lavoro contengono una combinazione di requisiti che alla fine sono coerenti (ad esempio, gli ordini totali nell'ora corrente) o fortemente coerenti (ad esempio, l'inventario corrente).

Per i carichi di lavoro che richiedono una forte coerenza dei dati, esistono modelli di architettura a supporto di questa funzionalità. Esempio:

- DynamoDB è in grado di fornire una elevata consistenza di lettura, a volte con una latenza più elevata, consumando un throughput maggiore rispetto alla modalità predefinita. DynamoDB può anche supportare le transazioni per aiutare a mantenere la coerenza dei dati.
- Puoi utilizzare Amazon RDS per funzionalità che richiedono proprietà ACID, sebbene qualsiasi database relazionale sia generalmente meno scalabile di database NoSQL come DynamoDB. Server proxy per Amazon RDS può aiutare a gestire il pooling e la scalabilità delle connessioni per utenti effimeri come le funzioni Lambda.

Le architetture basate sugli eventi sono generalmente progettate sulla base di singoli eventi anziché su grandi quantità di dati. In genere, i flussi di lavoro sono progettati per gestire le fasi di un singolo evento o flusso di esecuzione anziché operare su più eventi contemporaneamente. In modalità serverless, l'elaborazione degli eventi in tempo reale è preferita all'elaborazione in batch: i batch devono essere sostituiti con molti aggiornamenti incrementali più piccoli. Se da un lato ciò può rendere i carichi di lavoro più disponibili e scalabili, dall'altro rende più difficile per gli eventi conoscere altri eventi.

Restituzione di valori ai chiamanti

In molti casi, le applicazioni basate su eventi sono asincrone. Ciò significa che i servizi di chiamata non attendono le richieste da altri servizi prima di continuare con altre attività. Questa è una caratteristica fondamentale delle architetture basate sugli eventi che consente scalabilità e flessibilità. Significa anche che il passaggio dei valori restituiti o del risultato di un flusso di lavoro è più complesso rispetto ai flussi di esecuzione sincroni.

La maggior parte delle invocazioni Lambda nei sistemi di produzione sono asincrone e rispondono agli eventi di servizi come Amazon S3 o Amazon SQS. In questi casi, il successo o il fallimento dell'elaborazione di un evento è spesso più importante della restituzione di un valore. In Lambda vengono fornite funzionalità come dead letter queues (DLQs) per garantire che sia possibile identificare e riprovare gli eventi non riusciti, senza dover avvisare il chiamante.

Esecuzione del debug tra servizi e funzioni

Il debug di sistemi basati sugli eventi è inoltre diverso rispetto a un'applicazione monolitica. Poiché diversi sistemi e servizi trasmettono eventi, non è possibile registrare e riprodurre lo stato esatto di più servizi quando si verifica un errore. Poiché ogni invocazione di servizio e funzione ha file di

log separati, può essere più complicato determinare cosa è successo a un evento specifico che ha causato un errore.

Esistono tre requisiti importanti per creare un approccio di debug efficace nei sistemi basati sugli eventi. Innanzitutto, è fondamentale disporre di un sistema di registrazione robusto, che viene fornito attraverso diversi AWS servizi e integrato nelle funzioni Lambda da Amazon. CloudWatch In secondo luogo, in questi sistemi è importante garantire che ogni evento abbia un identificatore di transazione registrato in ogni fase della transazione, per facilitare la ricerca nei log.

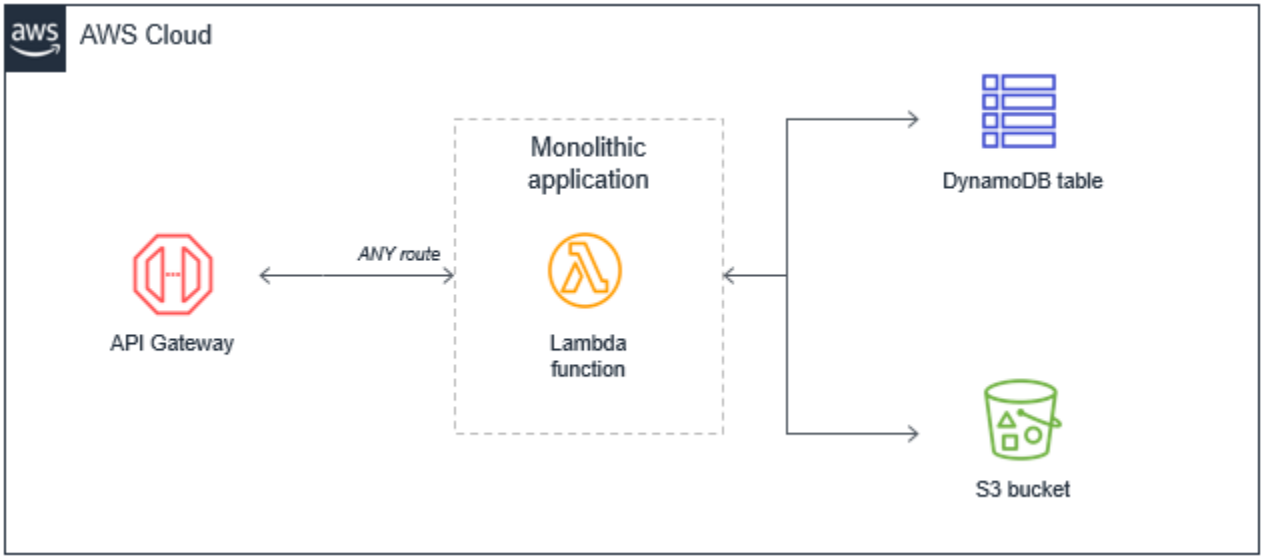
Infine, si consiglia vivamente di automatizzare l'analisi e l'analisi dei log utilizzando un servizio di debug e monitoraggio come AWS X-Ray. Ciò può consumare i log di più invocazioni Lambda e servizi, rendendo molto più semplice individuare la causa principale dei problemi. Consulta la procedura dettagliata sulla risoluzione dei problemi per una descrizione approfondita dell'utilizzo di X-Ray per la risoluzione dei problemi.

Anti-modelli nelle applicazioni basate su Lambda e sugli eventi

Quando crei architetture basate sugli eventi con Lambda, evita i seguenti anti-pattern comuni. Questi modelli funzionano ma possono aumentare i costi e la complessità.

Il monolite Lambda

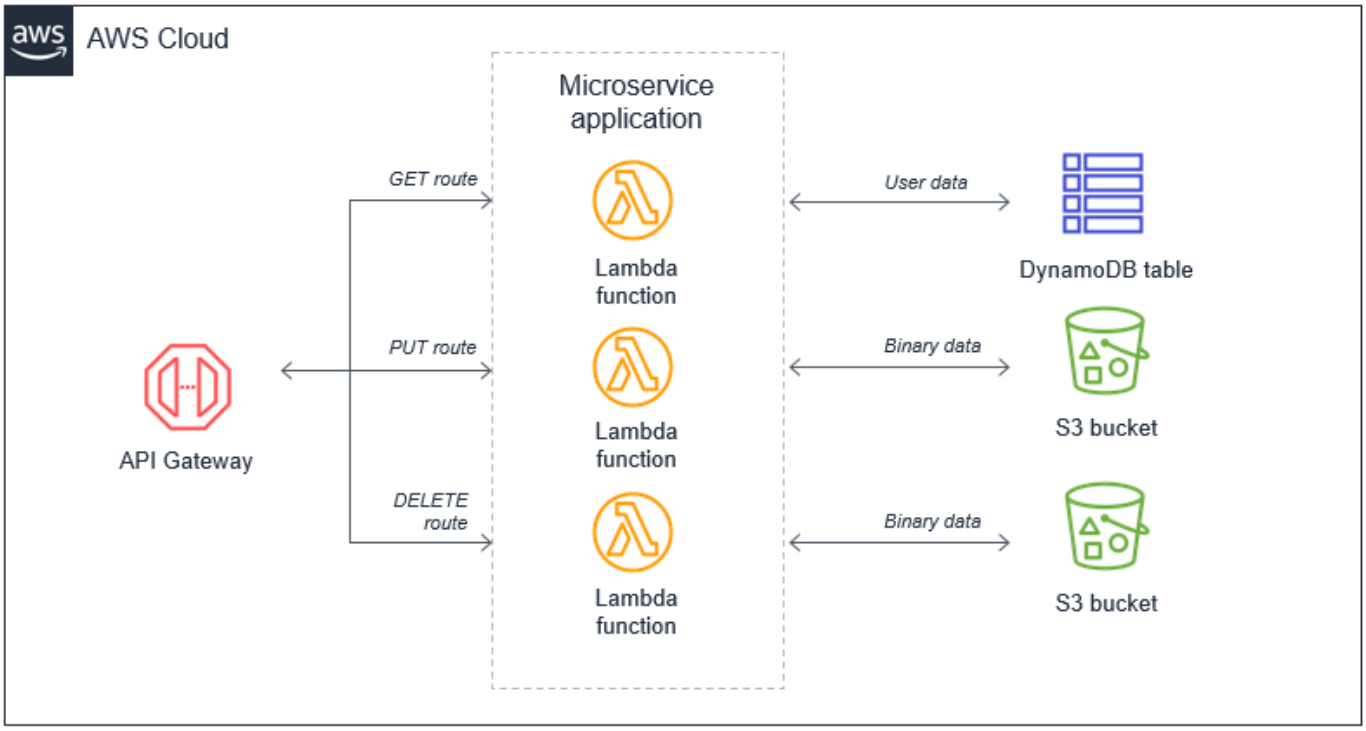
In molte applicazioni migrate da server tradizionali, come le EC2 istanze Amazon o le applicazioni Elastic Beanstalk, gli sviluppatori «sollevano e spostano» il codice esistente. Spesso, ciò si traduce in una singola funzione Lambda che contiene tutta la logica dell'applicazione attivata per tutti gli eventi. Per un'applicazione Web di base, una funzione Lambda monolitica gestirebbe tutti i percorsi API Gateway e si integrerebbe con tutte le risorse downstream necessarie.



Questo approccio presenta diversi inconvenienti:

- Dimensione del pacchetto: la funzione Lambda potrebbe essere molto più grande perché contiene tutto il codice possibile per tutti i percorsi, il che rende più lenta l'esecuzione del servizio Lambda.
- Difficile applicare il privilegio minimo: il ruolo di esecuzione della funzione deve consentire le autorizzazioni a tutte le risorse necessarie per tutti i percorsi, rendendo le autorizzazioni molto ampie. Si tratta di un problema di sicurezza. Molti percorsi nel monolite funzionale non necessitano di tutte le autorizzazioni concesse.
- Più difficile da aggiornare: in un sistema di produzione, qualsiasi aggiornamento alla singola funzione è più rischioso e potrebbe causare l'interruzione dell'intera applicazione. L'aggiornamento di un singolo percorso nella funzione Lambda è un aggiornamento dell'intera funzione.
- Più difficile da gestire: è più difficile avere più sviluppatori che lavorano sul servizio poiché si tratta di un repository di codice monolitico. Inoltre, ciò aumenta il carico cognitivo per gli sviluppatori e rende più difficile creare una copertura di test adeguata per il codice.
- Codice più difficile da riutilizzare: può essere più difficile separare le librerie riutilizzabili dai monoliti, il che rende più difficile il riutilizzo del codice. Man mano che sviluppi e supporti più progetti, ciò può rendere più difficile il supporto del codice e aumentare la velocità del team.
- Più difficile da testare: all'aumentare delle righe di codice, diventa più difficile eseguire test di unità di tutte le possibili combinazioni di input e punti di ingresso nella codebase. In genere è più semplice implementare il test di unità per servizi più piccoli con meno codice.

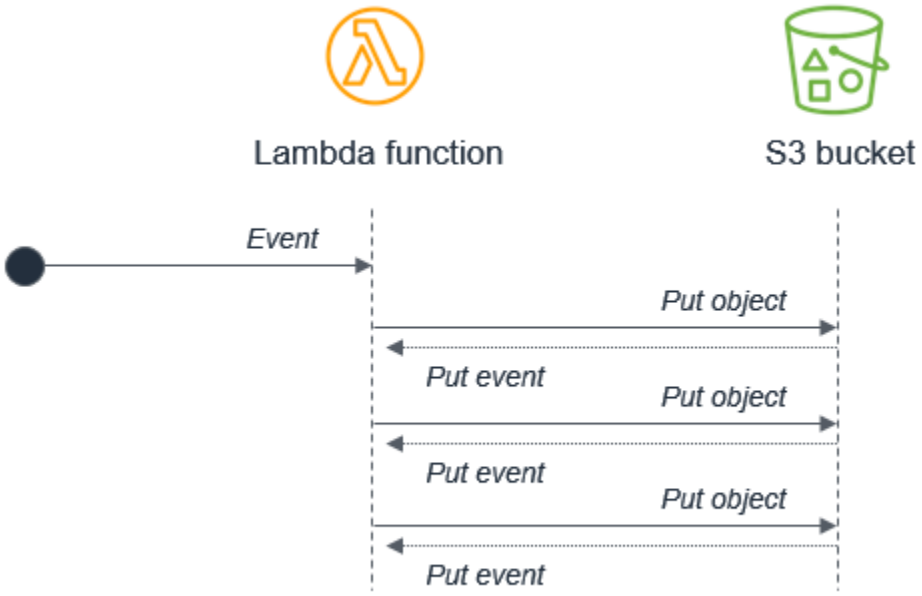
L'alternativa preferita consiste nel suddividere la funzione Lambda monolitica in singoli microservizi, mappando una singola funzione Lambda a un'unica attività ben definita. In questa semplice applicazione Web con pochi endpoint API, l'architettura risultante basata su microservizi può essere basata sui percorsi API Gateway.



Schemi ricorsivi che causano funzioni Lambda fuori controllo

AWS i servizi generano eventi che richiamano le funzioni Lambda e le funzioni Lambda possono inviare messaggi ai servizi. AWS In genere, il servizio o la risorsa che richiama una funzione Lambda deve essere diverso dal servizio o dalla risorsa a cui la funzione invia l'output. La mancata gestione di questa situazione può comportare cicli infiniti.

Ad esempio, una funzione Lambda scrive un oggetto su un oggetto Amazon S3, che a sua volta richiama la stessa funzione Lambda tramite un evento put. L'invocazione fa sì che un secondo oggetto venga scritto nel bucket, che richiama la stessa funzione Lambda:



Sebbene il potenziale dei cicli infiniti esista nella maggior parte dei linguaggi di programmazione, questo anti-modello ha il potenziale di consumare più risorse nelle applicazioni serverless. Sia Lambda che Amazon S3 scalano automaticamente in base al traffico, quindi il loop può far sì che Lambda si scali per consumare tutta la concorrenza disponibile e Amazon S3 continuerà a scrivere oggetti e generare più eventi per Lambda.

Questo esempio utilizza S3 ma il rischio di cicli ricorsivi esiste anche in Amazon SNS, Amazon SQS, DynamoDB e altri servizi. È possibile utilizzare il rilevamento ricorsivo del loop per trovare ed evitare questo anti-modello.

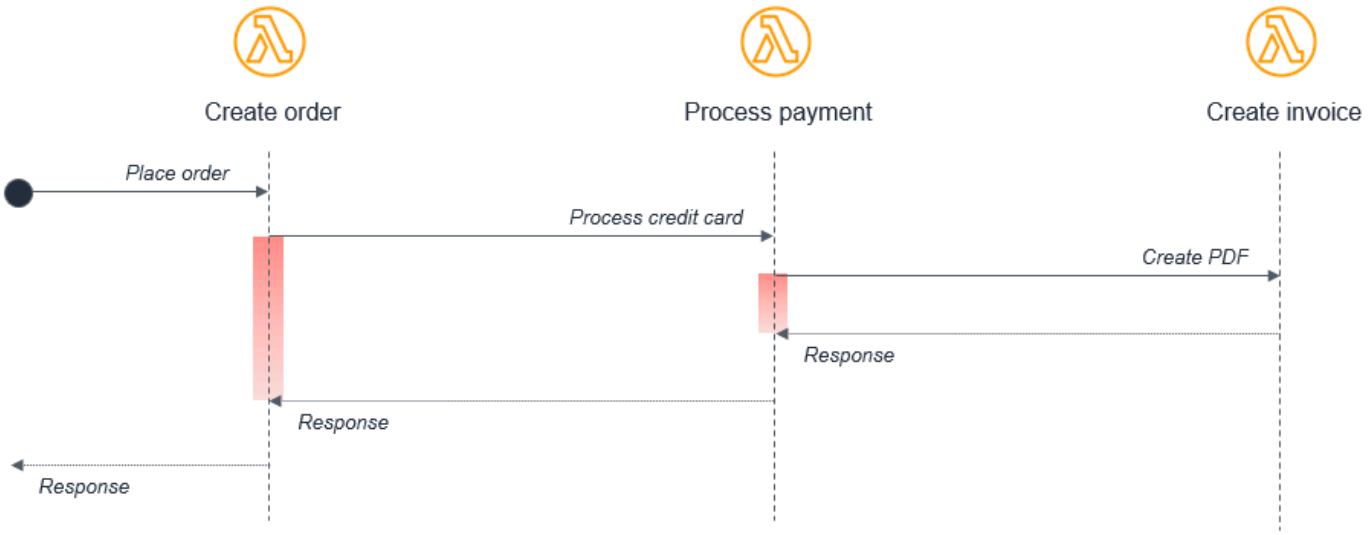
Funzioni Lambda che richiamano funzioni Lambda

Le funzioni consentono l'incapsulamento e il riutilizzo del codice. La maggior parte dei linguaggi di programmazione supporta il concetto di codice che richiama in modo sincrono le funzioni all'interno di una codebase. In questo caso, il chiamante attende che la funzione restituisca una risposta.

> **Note**
>
> Sebbene le funzioni Lambda che richiamano direttamente altre funzioni Lambda siano generalmente un anti-pattern a causa di problemi di costi e complessità, ciò non si applica alle funzioni durevoli, che sono specificamente progettate per orchestrare flussi di lavoro in più fasi richiamando altre funzioni.

Quando ciò accade su un server tradizionale o su un'istanza virtuale, lo scheduler del sistema operativo passa ad altre attività disponibili. Il fatto che la CPU funzioni allo 0% o al 100% non influisce sul costo complessivo dell'applicazione, poiché si paga il costo fisso di proprietà e gestione di un server.

Questo modello spesso non si adatta bene allo sviluppo serverless. Ad esempio, considera una semplice applicazione di e-commerce composta da tre funzioni Lambda che elaborano un ordine:



In questo caso, la funzione Crea ordine richiama la funzione Elabora pagamento, che a sua volta richiama la funzione Crea fattura. Sebbene questo flusso sincrono possa funzionare all'interno di una singola applicazione su un server, introduce diversi problemi evitabili in un'architettura serverless distribuita:

- Costo: con Lambda, paghi per la durata di una invocazione. In questo esempio, mentre le funzioni Crea fattura sono in esecuzione, anche altre due funzioni sono in esecuzione in stato di attesa, mostrate in rosso nel diagramma.
- Gestione degli errori: nelle chiamate nidificate, la gestione degli errori può diventare molto più complessa. Ad esempio, un errore nella creazione della fattura potrebbe richiedere la funzione Elabora pagamento per stornare l'addebito oppure potrebbe invece riprovare il processo di creazione della fattura.
- Accoppiamento stretto: l'elaborazione di un pagamento richiede in genere più tempo rispetto alla creazione di una fattura. In questo modello, la disponibilità dell'intero flusso di lavoro è limitata dalla funzione più lenta.
- Scalabilità: la simultaneità di tutte e tre le funzioni deve essere uguale. In un sistema affollato, ciò utilizza più simultaneità di quanto sarebbe altrimenti necessario.

Nelle applicazioni serverless, esistono due approcci comuni per evitare questo modello. Innanzitutto, usa una coda Amazon SQS tra le funzioni Lambda. Se un processo a valle è più lento di un processo a monte, la coda mantiene i messaggi in modo duraturo e disaccoppia le due funzioni. In questo esempio, la funzione Create order pubblicherebbe un messaggio in una coda Amazon SQS e la funzione di pagamento Process consumerebbe i messaggi dalla coda.

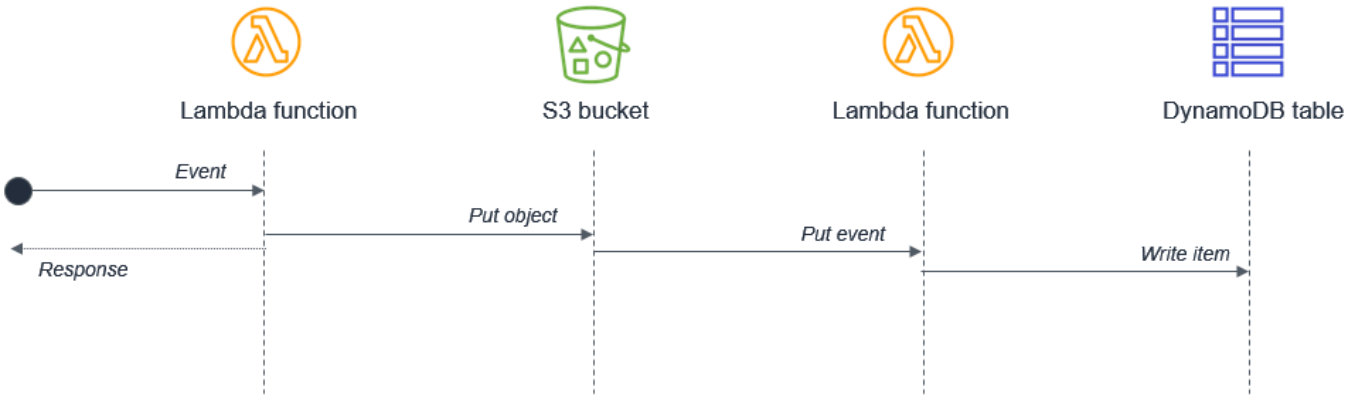
Il secondo approccio consiste nell'utilizzare AWS Step Functions. Per processi complessi con diversi tipi di logica di errore e nuovo tentativo, Step Functions può aiutare a ridurre la quantità di codice personalizzato necessario per orchestrare il flusso di lavoro. Di conseguenza, Step Functions orchestra il lavoro e gestisce in modo efficace errori e nuovi tentativi, mentre le funzioni Lambda contengono solo la logica aziendale.

Attesa sincrona all'interno di una singola funzione Lambda

Assicurati che le attività potenzialmente concorrenti non siano pianificate in modo sincrono all'interno di una singola funzione Lambda. Ad esempio, una funzione Lambda potrebbe scrivere su un bucket S3 e quindi scrivere su una tabella DynamoDB:



In questa progettazione, i tempi di attesa sono composti perché le attività sono sequenziali. Nei casi in cui la seconda attività dipende dal completamento della prima, puoi ridurre il tempo di attesa totale e il costo di esecuzione utilizzando due funzioni Lambda distinte:



In questo design, la prima funzione Lambda risponde immediatamente dopo aver inserito l'oggetto nel bucket Amazon S3. Il servizio S3 richiama la seconda funzione Lambda, che quindi scrive i dati nella tabella DynamoDB. Questo approccio riduce al minimo il tempo di attesa totale nelle esecuzioni delle funzioni Lambda.

Progettazione di applicazioni Lambda

Un'applicazione basata sugli eventi ben progettata utilizza una combinazione di AWS servizi e codice personalizzato per elaborare e gestire richieste e dati. Questo capitolo si concentra su argomenti specifici di Lambda nella progettazione di applicazioni. Ci sono numerose considerazioni importanti per gli architetti serverless quando progettano applicazioni per sistemi di produzione affollati.

Molte delle best practice che si applicano allo sviluppo di software e ai sistemi distribuiti si applicano anche allo sviluppo di applicazioni serverless. L'obiettivo generale è sviluppare carichi di lavoro che siano:

- Affidabili: offrendo agli utenti finali un elevato livello di disponibilità, i servizi serverless AWS sono affidabili perché progettati anche per evitare guasti.
- Durevoli: offrendo opzioni di archiviazione che soddisfano le esigenze di durabilità del carico di lavoro.
- Sicuri: seguendo le best practice e utilizzando gli strumenti forniti per proteggere l'accesso ai carichi di lavoro e limitare il raggio di esplosione.
- Performanti: utilizzando le risorse di elaborazione in modo efficiente e soddisfando le esigenze di prestazioni degli utenti finali.
- Efficienti in termini di costi: progettando architetture che evitino costi inutili, scalabili senza spese eccessive e che possano essere disattivate senza costi generali significativi.

I seguenti principi di progettazione possono aiutarti a creare carichi di lavoro che soddisfino questi obiettivi. Non tutti i principi possono essere applicati a tutte le architetture, ma dovrebbero guidarti nelle decisioni generali sull'architettura.

Argomenti

- Utilizzare i servizi anziché il codice personalizzato
- Informazioni sui livelli di astrazione Lambda
- Implementazione stateless nelle funzioni
- Riduzione dell'associazione
- Creazione di dati on demand anziché in batch
- Prendi in considerazione l'orchestrazione AWS Step Functions
- Implementa l'idempotenza
- Utilizza AWS più account per la gestione delle quote

Utilizzare i servizi anziché il codice personalizzato

Le applicazioni serverless di solito comprendono diversi AWS servizi, integrati con codice personalizzato eseguito nelle funzioni Lambda. Sebbene Lambda possa essere integrato con la maggior parte dei AWS servizi, i servizi più comunemente utilizzati nelle applicazioni serverless sono:

Categoria	AWS servizio
Calcolo	AWS Lambda
Archiviazione di dati	Simple Storage Service (Amazon S3) Amazon DynamoDB Amazon RDS
"Hello, World!"	Gateway Amazon API
Integrazione delle applicazioni	Amazon EventBridge Amazon SNS Amazon SQS
Orchestrazione	AWS Step Functions
Streaming di dati e analisi	Amazon Data Firehose

> **Note**
>
> Molti servizi serverless forniscono replica e supporto per più regioni, tra cui DynamoDB e Amazon S3. Le funzioni Lambda possono essere implementate in più regioni come parte di una pipeline di implementazione e API Gateway può essere configurato per supportare questa configurazione. Guarda questo esempio di architettura che mostra come raggiungere questo obiettivo.

Esistono molti modelli comuni e consolidati nelle architetture distribuite che è possibile creare autonomamente o implementare utilizzando i servizi. AWS Per la maggior parte dei clienti, investire

tempo per sviluppare questi modelli partendo da zero ha poco valore commerciale. Quando l'applicazione necessita di uno di questi modelli, utilizzate il servizio corrispondente: AWS

Pattern	AWS servizio
Queue	Amazon SQS
Router di eventi	Amazon EventBridge
Pubblicazione/sottoscrizione (fan-out)	Amazon SNS
Orchestrazione	AWS Step Functions
"Hello, World!"	Gateway Amazon API
Flussi di eventi	Amazon Kinesis

Questi servizi sono progettati per integrarsi con Lambda e puoi utilizzare l'infrastructure as code (IaC) per creare ed eliminare risorse nei servizi. Puoi utilizzare uno qualsiasi di questi servizi tramite l'SDK AWS senza dover installare applicazioni o configurare server. Diventare esperti nell'uso di questi servizi tramite codice nelle funzioni Lambda è un passo importante per la produzione di applicazioni serverless ben progettate.

Informazioni sui livelli di astrazione Lambda

Il servizio Lambda limita l'accesso ai sistemi operativi, agli hypervisor e all'hardware sottostanti che eseguono le funzioni Lambda. Il servizio migliora e modifica continuamente l'infrastruttura per aggiungere funzionalità, ridurre i costi e rendere il servizio più performante. Il codice non deve presupporre alcuna conoscenza dell'architettura di Lambda né presupporre alcuna affinità hardware.

Allo stesso modo, le integrazioni di Lambda con altri servizi sono gestite da AWS, con solo un numero limitato di opzioni di configurazione a tua disposizione. Ad esempio, quando API Gateway e Lambda interagiscono, non sussiste il concetto di bilanciamento del carico poiché è interamente gestito dai servizi. Inoltre, non hai alcun controllo diretto sulle zone di disponibilità utilizzate dai servizi per invocare le funzioni in qualsiasi momento, o su come Lambda determina quando aumentare o ridurre verticalmente il numero di ambienti di esecuzone.

Questa astrazione ti aiuta a concentrarti sugli aspetti di integrazione dell'applicazione, sul flusso di dati e sulla logica aziendale in cui il carico di lavoro fornisce valore agli utenti finali. Consentire ai

servizi di gestire i meccanismi sottostanti consente di sviluppare applicazioni più rapidamente con meno codice personalizzato da gestire.

Implementazione stateless nelle funzioni

Per le funzioni Lambda standard, è necessario presupporre che l'ambiente esista solo per una singola chiamata. La funzione dovrebbe inizializzare qualsiasi stato richiesto al primo avvio. Ad esempio, la funzione potrebbe richiedere il recupero di dati da una tabella DynamoDB. Deve eseguire eventuali modifiche permanenti ai dati in un archivio durevole come Amazon S3, DynamoDB o Amazon SQS prima dell'uscita. Non dovrebbe basarsi su strutture di dati o file temporanei esistenti o su qualsiasi stato interno che verrebbe gestito da più invocazioni.

Quando si utilizza Durable Functions, lo stato viene automaticamente preservato tra le chiamate, eliminando la necessità di mantenere manualmente lo stato sullo storage esterno. Tuttavia, è comunque necessario seguire i principi stateless per tutti i dati non gestiti in modo esplicito tramite. DurableContext

Per inizializzare le connessioni e le librerie del database o lo stato di caricamento, è possibile sfruttare l'inizializzazione statica. Poiché gli ambienti di esecuzione vengono riutilizzati ove possibile per migliorare le prestazioni, è possibile ammortizzare il tempo impiegato per inizializzare queste risorse su più invocazioni. Tuttavia, non è necessario archiviare alcuna variabile o dato utilizzato nella funzione all'interno di questo ambito globale.

Riduzione dell'associazione

La maggior parte delle architetture dovrebbe preferire molte funzioni più brevi rispetto a un numero inferiore di funzioni più grandi. Lo scopo di ogni funzione dovrebbe essere quello di gestire l'evento trasmesso alla funzione, senza alcuna conoscenza o aspettativa del flusso di lavoro complessivo o del volume delle transazioni. Ciò rende la funzione indipendente dall'origine eventi con un abbinamento minimo ad altri servizi.

Tutte le costanti di ambito globale che cambiano di rado devono essere implementate come variabili di ambiente per consentire aggiornamenti senza implementazioni. Eventuali informazioni segrete o sensibili devono essere archiviate in AWS Systems Manager Parameter Store o AWS Secrets Manager e caricate dalla funzione. Poiché queste risorse sono specifiche dell'account, puoi creare pipeline di compilazione su più account. Le pipeline caricano i segreti appropriati per ambiente, senza esporli agli sviluppatori o richiedere modifiche al codice.

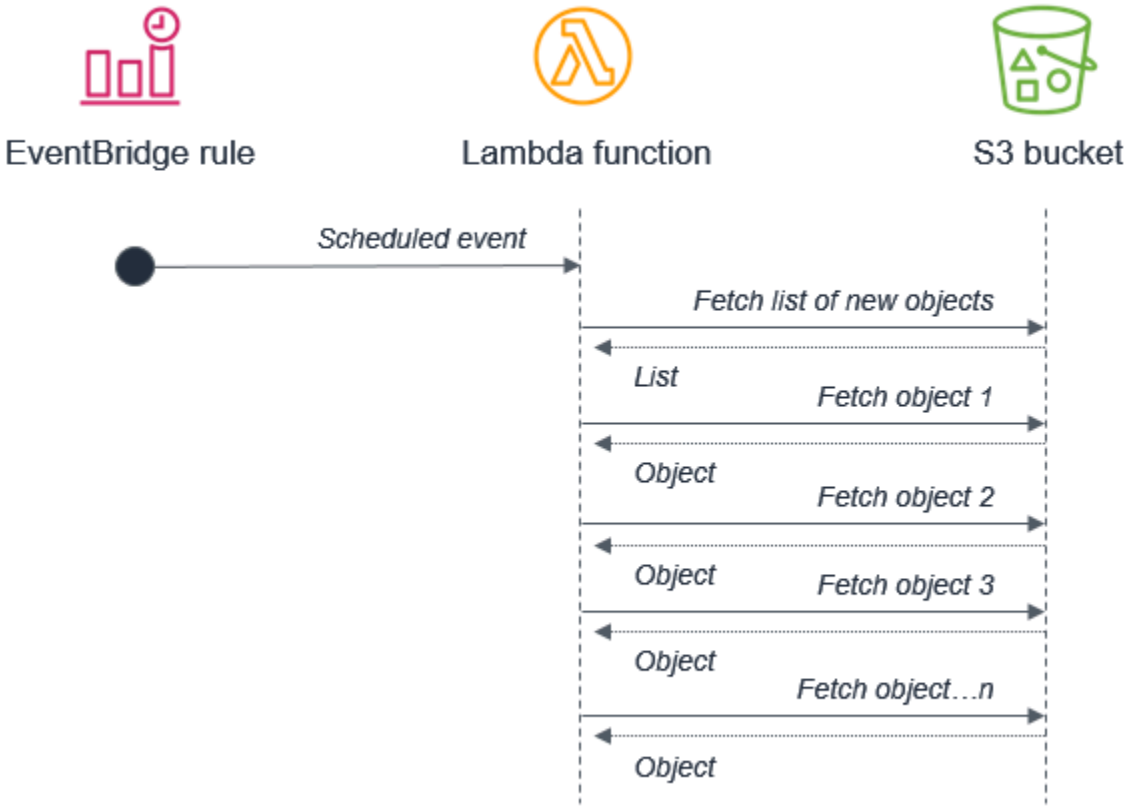
Creazione di dati on demand anziché in batch

Molti sistemi tradizionali sono progettati per funzionare periodicamente ed elaborare batch di transazioni accumulati nel tempo. Ad esempio, un'applicazione bancaria può essere eseguita ogni ora per elaborare le transazioni dei bancomat in log centrali. Nelle applicazioni basate su Lambda, l'elaborazione personalizzata deve essere attivata da ogni evento, consentendo al servizio di aumentare la simultaneità secondo necessità, per fornire un'elaborazione delle transazioni quasi in tempo reale.

Sebbene le funzioni Lambda standard siano limitate a 15 minuti di esecuzione, Durable Functions può funzionare fino a un anno, il che le rende adatte a esigenze di elaborazione più lunghe. Tuttavia, dovresti comunque preferire l'elaborazione basata sugli eventi rispetto all'elaborazione in batch, quando possibile.

Sebbene sia possibile eseguire attività cron in applicazioni serverless utilizzando espressioni pianificate per le regole in Amazon EventBridge, queste dovrebbero essere utilizzate con parsimonia o come ultima risorsa. In qualsiasi attività pianificata che elabora un batch, esiste la possibilità che il volume delle transazioni cresca oltre quanto può essere elaborato entro il limite di durata Lambda di 15 minuti. Se le limitazioni dei sistemi esterni ti obbligano a utilizzare uno scheduler, in genere dovresti programmare per il periodo di tempo ricorrente più breve e ragionevole.

Ad esempio, non è consigliabile utilizzare un processo batch che attiva una funzione Lambda per recuperare un elenco di nuovi oggetti Amazon S3. Questo perché il servizio può ricevere più nuovi oggetti tra un batch e l'altro di quanti ne possano essere elaborati in una funzione Lambda di 15 minuti.



Invece, Amazon S3 deve invocare la funzione Lambda ogni volta che un nuovo oggetto viene inserito nel bucket. Questo approccio è significativamente più scalabile e funziona quasi in tempo reale.



Prendi in considerazione l'orchestrazione AWS Step Functions

I flussi di lavoro che coinvolgono la logica di ramificazione, diversi tipi di modelli di errore e la logica dei tentativi in genere utilizzano un orchestratore per tenere traccia dello stato dell'esecuzione complessiva. Evita di utilizzare le funzioni Lambda per questo scopo, poiché si ottengono routing per la gestione del codice ad accoppiamento stretto e complessi.

Con AWS Step Functions, si utilizzano macchine a stati per gestire l'orchestrazione. Questo estrae la gestione degli errori, il routing e la logica di ramificazione dal codice, sostituendola con macchine a stati dichiarate utilizzando JSON. Oltre a rendere i flussi di lavoro più robusti e osservabili, puoi

anche aggiungere il controllo delle versioni ai flussi di lavoro e rendere la macchina a stati una risorsa codificata da aggiungere a un archivio di codice.

È normale che i flussi di lavoro più semplici nelle funzioni Lambda diventino più complessi nel tempo. Quando si utilizza un'applicazione serverless di produzione, è importante identificare quando ciò accade, in modo da poter migrare questa logica su una macchina a stati.

Implementa l'idempotenza

AWS i servizi serverless, tra cui Lambda, sono tolleranti ai guasti e progettati per gestire i guasti. Ad esempio, se un servizio richiama una funzione Lambda e si verifica un'interruzione del servizio, Lambda invoca la funzione in una zona di disponibilità diversa. Se la funzione emette un errore, Lambda ritenta l'invocazione.

Poiché lo stesso evento può essere ricevuto più di una volta, le funzioni devono essere progettate in modo da essere idempotenti. Ciò significa che ricevere lo stesso evento più volte non modifica il risultato dopo la prima ricezione dell'evento.

Puoi implementare l'idempotenza nelle funzioni Lambda utilizzando una tabella DynamoDB per tenere traccia degli identificatori elaborati di recente per determinare se la transazione è già stata gestita in precedenza. La tabella DynamoDB di solito implementa un valore Time To Live (TTL) per gli elementi con scadenza per limitare lo spazio di archiviazione utilizzato.

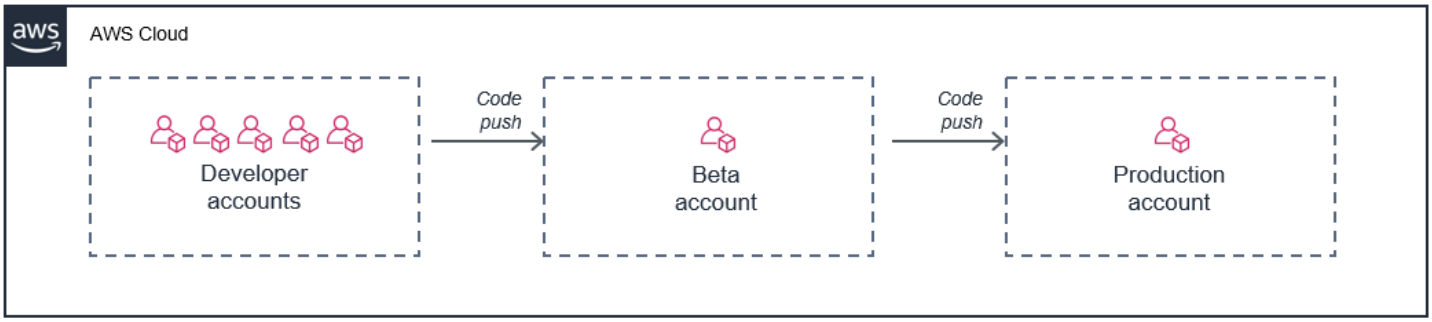
Utilizza AWS più account per la gestione delle quote

Molte quote di servizio AWS sono impostate a livello di account. Ciò significa che man mano che aggiungi più carichi di lavoro, puoi esaurire rapidamente i tuoi limiti.

Un modo efficace per risolvere questo problema consiste nell'utilizzare più AWS account, dedicando ogni carico di lavoro al proprio account. Ciò impedisce che le quote vengano condivise con altri carichi di lavoro o risorse non di produzione.

In aggiunta, utilizzando Organizzazioni AWS, puoi gestire centralmente la fatturazione, la conformità e la sicurezza di questi account. È possibile collegare policy a gruppi di account per evitare script personalizzati e processi manuali.

Un approccio comune consiste nel fornire a ogni sviluppatore un AWS account e quindi utilizzare account separati per la fase di distribuzione beta e la produzione:



In questo modello, ogni sviluppatore ha i propri limiti per l'account, quindi il loro utilizzo non ha alcun impatto sull'ambiente di produzione. Questo approccio consente inoltre agli sviluppatori di testare le funzioni Lambda localmente sulle proprie macchine di sviluppo con risorse cloud attive nei propri account individuali.

Crea la tua prima funzione Lambda

Per iniziare a utilizzare Lambda, usa la console Lambda per creare una funzione. In pochi minuti puoi creare e implementare una funzione, e testarla nella console.

Mano a mano che esegui il tutorial, apprenderai alcuni concetti fondamentali di Lambda, ad esempio come passare argomenti a una funzione tramite l'oggetto evento di Lambda. Imparerai anche come restituire gli output di log dalla tua funzione e come visualizzare i log di invocazione della tua funzione in Amazon Logs. CloudWatch

A scopo di semplificazione, crea la funzione utilizzando il runtime Python o Node.js. Con questi linguaggi interpretati, puoi modificare il codice della funzione direttamente nell'editor del codice integrato della console. Con linguaggi compilati come Java e C#, devi creare un pacchetto di distribuzione sulla tua macchina di build locale e caricarlo su Lambda. Per informazioni sull'implementazione di funzioni in Lambda tramite altri runtime, consulta i link nella sezione the section called “Fasi successive”.

> **Tip**
>
> Per scoprire come creare soluzioni serverless, consulta la Guida allo sviluppo serverless.

Prerequisiti

Registrati per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la https://portal.aws.amazon.com/billing/registrazione.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata o un messaggio di testo e ti verrà chiesto di inserire un codice di verifica attraverso la tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L’utente root dispone dell’accesso a tutte le risorse e tutti i Servizi AWS nell’account. Come best practice di

sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

AWS ti invia un'email di conferma dopo il completamento della procedura di registrazione. In qualsiasi momento, puoi visualizzare l'attività corrente del tuo account e gestirlo accedendo a https://aws.amazon.com/e scegliendo Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi Console di gestione AWScome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .
2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita il Centro identità IAM.

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .
2. In IAM Identity Center, assegna l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con l'impostazione predefinita IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accesso come utente amministratore

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso a ulteriori utenti

1. In IAM Identity Center, crea un set di autorizzazioni conforme alla best practice dell'applicazione di autorizzazioni con il privilegio minimo.

   Segui le istruzioni riportate nella pagina Creazione di un set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .
2. Assegna al gruppo prima gli utenti e poi l'accesso con autenticazione unica (Single Sign-On).

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente di AWS IAM Identity Center .

Creare una funzione Lambda con la console

In questo esempio, la funzione acquisisce un oggetto JSON contenente due valori interi etichettati `"length"` e `"width"`. La funzione moltiplica tali valori per calcolare un'area e la restituisce come stringa JSON.

La funzione stampa anche l'area calcolata, insieme al nome del relativo gruppo di CloudWatch log. Più avanti nel tutorial, imparerai a usare CloudWatch Logs per visualizzare i record di invocazione delle tue funzioni.

Per creare una funzione Lambda Hello world con la console

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli Crea funzione.
3. Scegli Crea da zero.
4. Nel riquadro Informazioni base, per Nome funzione inserisci **`myLambdaFunction`**.
5. Per Runtime, scegli Node.js 24 o Python 3.14.
6. Lascia l'architettura impostata su x86_64 e scegli Crea funzione.

Oltre a una funzione semplice che restituisce il messaggio `Hello from Lambda!`, Lambda crea anche un ruolo di esecuzione per la tua funzione. Un ruolo di esecuzione è un ruolo AWS Identity and Access Management (IAM) che concede a una funzione Lambda l'autorizzazione all' Servizi AWS accesso e alle risorse. Per la tua funzione, il ruolo creato da Lambda concede le autorizzazioni di base per la scrittura nei registri. CloudWatch

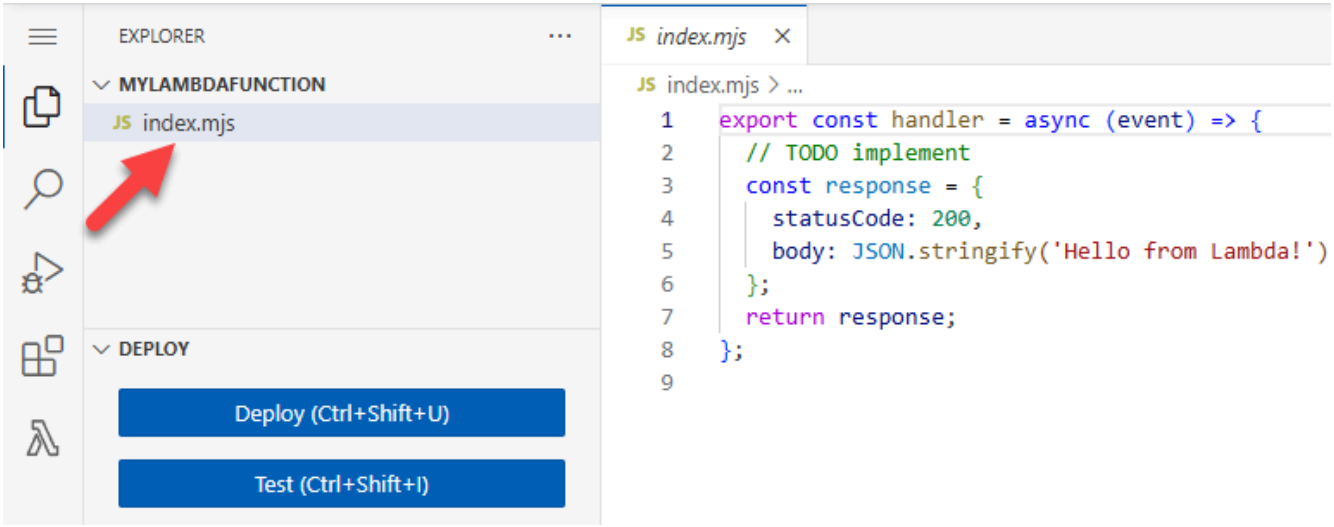
Utilizza l'editor di codice integrato della console per sostituire il codice Hello world creato da Lambda con il codice della tua funzione.

Node.js

Per modificare il codice nella console

1. Scegli la scheda Codice.

   Nell'editor di codice integrato della console, dovresti vedere il codice della funzione creato da Lambda. Se non vedi la scheda index.js nell'editor di codice, seleziona index.js in Esplora file, come illustrato nel diagramma seguente.



2. Incolla il codice seguente nella scheda index.js, sostituendo il codice creato da Lambda.

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

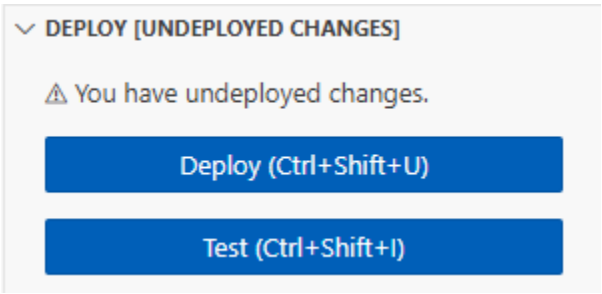
  const length = event.length;
  const width = event.width;
  let area = calculateArea(length, width);
  console.log(`The area is ${area}`);

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

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

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

3. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



Comprendere il codice della funzione

Prima di passare alla fase successiva, esaminiamo il codice della funzione e apprendiamo alcuni concetti chiave di Lambda.

- Il gestore Lambda:

  La tua funzione Lambda contiene una funzione Node.js denominata `handler`. Una funzione Lambda in Node.js può contenere più di una funzione Node.js, ma la funzione del gestore è sempre il punto di ingresso al codice. Quando viene richiamata la funzione, Lambda esegue questo metodo.

  Una volta creata la funzione Hello world utilizzando la console, Lambda imposta automaticamente su `handler` il nome del metodo del gestore per la funzione. Non modificare il nome di questa funzione Node.js. Se modifichi il nome, Lambda non sarà in grado di eseguire il codice quando richiami la funzione.

  Per ulteriori informazioni sul gestore Lambda in Node.js, consulta the section called “Gestore”.

- L'oggetto evento Lambda:

La funzione `handler` acquisisce due argomenti, `event` e `context`. Un evento in Lambda è un documento in formato JSON contenente i dati che la funzione deve elaborare.

Se la funzione viene richiamata da un'altra Servizio AWS, l'oggetto evento contiene informazioni sull'evento che ha causato l'invocazione. Ad esempio, se la funzione viene richiamata quando un oggetto viene caricato in un bucket Amazon Simple Storage Service (Amazon S3), l'evento conterrà il nome del bucket e la chiave dell'oggetto.

In questo esempio verrà creato un evento nella console inserendo un documento in formato JSON con due coppie chiave-valore.

- L'oggetto contestuale Lambda:

  Il secondo argomento acquisito dalla funzione è `context`. Lambda passa l'oggetto contestuale alla tua funzione automaticamente. L'oggetto contestuale contiene informazioni sull'invocazione della funzione e sull'ambiente di esecuzione.

  Puoi utilizzare l'oggetto contestuale per generare informazioni sull'invocazione della funzione a scopo di monitoraggio. In questo esempio, la funzione utilizza il `logGroupName` parametro per restituire il nome del relativo CloudWatch gruppo di log.

  Per ulteriori informazioni sull'oggetto contestuale Lambda in Node.js, consulta the section called “Contesto”.

- Accesso a Lambda:

  Con Node.js, puoi usare metodi della console come `console.log` e `console.error` per inviare informazioni al log della funzione. Il codice di esempio utilizza `console.log` istruzioni per restituire l'area calcolata e il nome del gruppo CloudWatch Logs della funzione. Puoi utilizzare anche qualunque libreria di log che scrive in `stdout` o `stderr`.

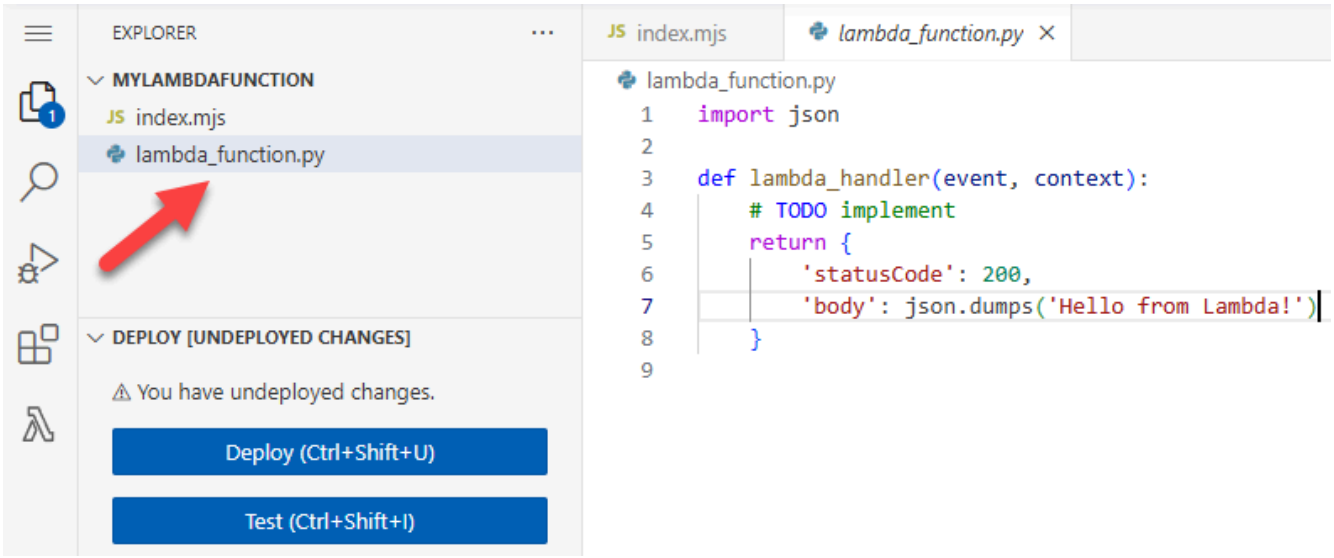
  Per ulteriori informazioni, consulta the section called “Registrazione dei log”. Per informazioni sui log in altri runtime, consulta le pagine “Creazione con” per i runtime a cui sei interessato.

Python

Per modificare il codice nella console

1. Scegli la scheda Codice.

Nell'editor di codice integrato della console, dovresti vedere il codice della funzione creato da Lambda. Se non vedi la scheda lambda_function.py nell'editor di codice, seleziona lambda_function.py in Esplora file, come illustrato nel diagramma seguente.



2. Incolla il codice seguente nella scheda lambda_function.py, sostituendo il codice creato da 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']

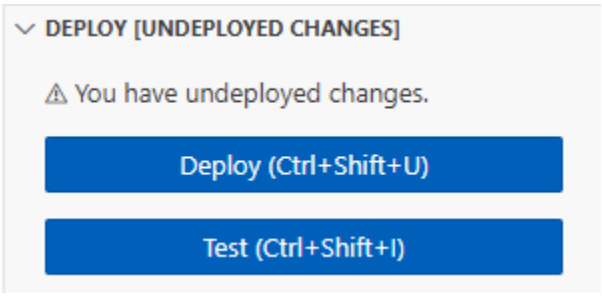
    area = calculate_area(length, width)
    print(f"The area is {area}")

    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. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



Comprendere il codice della funzione

Prima di passare alla fase successiva, esaminiamo il codice della funzione e apprendiamo alcuni concetti chiave di Lambda.

- Il gestore Lambda:

  La funzione Lambda contiene una funzione Python denominata `lambda_handler`. Una funzione Lambda in Python può contenere più di una funzione Python, ma la funzione del gestore è sempre il punto di ingresso al codice. Quando viene richiamata la funzione, Lambda esegue questo metodo.

  Una volta creata la funzione Hello world utilizzando la console, Lambda imposta automaticamente su `lambda_handler` il nome del metodo del gestore per la funzione. Non modificare il nome di questa funzione Python. Se modifichi il nome, Lambda non sarà in grado di eseguire il codice quando richiami la funzione.

  Per ulteriori informazioni sul gestore Lambda in Python, consulta the section called “Gestore”.
- L'oggetto evento Lambda:

  La funzione `lambda_handler` acquisisce due argomenti, `event` e `context`. Un evento in Lambda è un documento in formato JSON contenente i dati che la funzione deve elaborare.

  Se la funzione viene richiamata da un'altra Servizio AWS, l'oggetto evento contiene informazioni sull'evento che ha causato l'invocazione. Ad esempio, se la funzione viene richiamata quando un oggetto viene caricato in un bucket Amazon Simple Storage Service (Amazon S3), l'evento conterrà il nome del bucket e la chiave dell'oggetto.

In questo esempio verrà creato un evento nella console inserendo un documento in formato JSON con due coppie chiave-valore.

- L'oggetto contestuale Lambda:

  Il secondo argomento acquisito dalla funzione è `context`. Lambda passa l'oggetto contestuale alla tua funzione automaticamente. L'oggetto contestuale contiene informazioni sull'invocazione della funzione e sull'ambiente di esecuzione.

  Puoi utilizzare l'oggetto contestuale per generare informazioni sull'invocazione della funzione a scopo di monitoraggio. In questo esempio, la funzione utilizza il `log_group_name` parametro per restituire il nome del relativo CloudWatch gruppo di log.

  Per ulteriori informazioni sull'oggetto contestuale Lambda in Python, consulta the section called “Contesto”.

- Accesso a Lambda:

  Con Python, puoi usare un'istruzione `print` o una libreria di log Python per inviare informazioni al log della funzione. Per illustrare la differenza tra i dati acquisiti, il codice di esempio utilizza entrambi i metodi. In un'applicazione in produzione, è preferibile utilizzare una libreria di log.

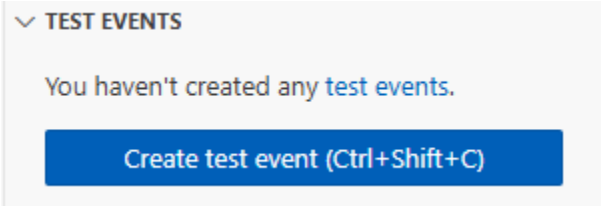
  Per ulteriori informazioni, consulta the section called “Registrazione dei log”. Per informazioni sui log in altri runtime, consulta le pagine “Creazione con” per i runtime a cui sei interessato.

Richiamare la funzione Lambda tramite l'editor di codice della console

Per richiamare la funzione utilizzando l'editor di codice della console Lambda, devi prima creare un evento di test da inviare alla tua funzione. L'evento è un documento in formato JSON contenente due coppie chiave-valore con le chiavi `"length"` e `"width"`.

Per creare un evento di test

1. Nella sezione TEST EVENTS dell'editor di codice della console, scegli Crea evento di test.



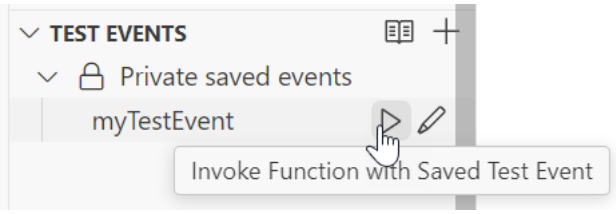
2. Per Event name (Nome evento), immettere **`myTestEvent`**.
3. Nella sezione JSON dell'evento, sostituisci il codice JSON predefinito con il seguente:

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

4. Scegli Save (Salva).

Per testare la funzione e visualizzare i record di invocazione nella console

Nella sezione TEST EVENTS dell'editor di codice della console, scegli l'icona di esecuzione accanto all'evento di test:



Al termine dell'esecuzione della funzione, verranno visualizzati i log della risposta e della funzione nella scheda OUTPUT. Sono visualizzati risultati simili ai seguenti:

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}"

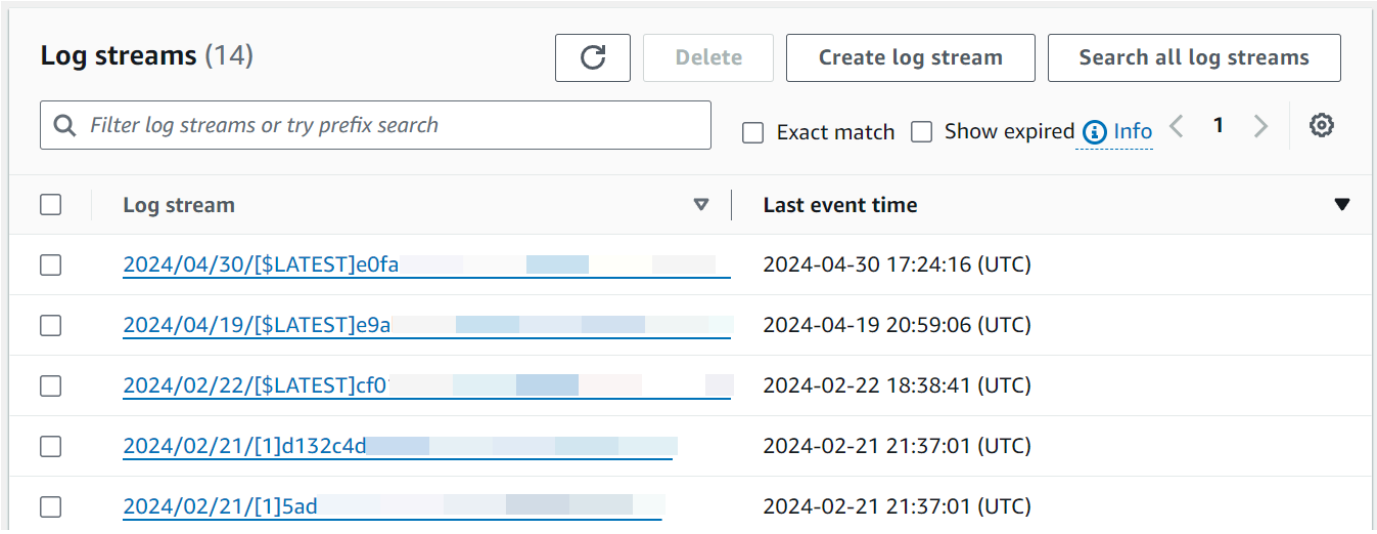
Function Logs
START RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b Version: $LATEST
The area is 42
[INFO] 2024-08-31T23:43:26.428Z 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b CloudWatch logs
 group: /aws/lambda/myLambdaFunction
END RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b
REPORT RequestId: 2d0b1579-46fb-4bf7-a6e1-8e08840eae5b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

Quando richiami la tua funzione al di fuori della console Lambda, devi CloudWatch usare Logs per visualizzare i risultati di esecuzione della funzione.

Per visualizzare i record di invocazione della funzione in Logs CloudWatch

1. Apri la pagina Log groups della console. CloudWatch
2. Scegli il nome del gruppo di log per la funzione (`/aws/lambda/myLambdaFunction`). Questo è il nome del gruppo di log che la funzione ha stampato sulla console.
3. Scorri verso il basso e scegli il flusso di log per le invocazioni delle funzioni che desideri esaminare.



Verrà visualizzato un output simile al seguente:

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
```

Eliminazione

Quando hai terminato il lavoro con la funzione di esempio, eliminala. Puoi anche eliminare il gruppo di log che memorizza i log della funzione e il ruolo di esecuzione creato dalla console.

Per eliminare la funzione Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **`confirm`** nel campo di immissione del testo, quindi scegli Elimina.

Come eliminare il gruppo di log

1. Apri la pagina Log groups page (Pagina dei gruppi di log) della console CloudWatch.
2. Scegli il gruppo di log della funzione (`/aws/lambda/myLambdaFunction`).
3. Scegli Actions (Azioni), Delete log group(s) (Elimina gruppo/i di log).
4. Nella finestra di dialogo Delete log group(s) (Elimina gruppo/i di log) scegli Delete (Elimina).

Come eliminare il ruolo di esecuzione

1. Apri la pagina Ruoli della console AWS Identity and Access Management (IAM).
2. Seleziona il ruolo di esecuzione della funzione (ad esempio, `myLambdaFunction-role-`*`31exxmpl`*).
3. Scegli Elimina.
4. Nella finestra di dialogo Elimina ruolo, immetti il nome del ruolo, quindi scegli Elimina.

Risorse aggiuntive e fasi successive

Una volta creata e testata una semplice funzione Lambda tramite la console, completa le seguenti fasi successive:

- Scopri come aggiungere dipendenze alla tua funzione e implementala utilizzando un pacchetto di implementazione .zip. Scegli la lingua preferita dai seguenti link.

  Node.js

  the section called “Implementazione di archivi di file .zip”

  Typescript

  the section called “Implementare archivi di file .zip”

  Python

  the section called “Implementazione di archivi di file .zip”

  Ruby

  the section called “Implementazione di archivi di file .zip”

  Java

  the section called “Implementazione di archivi di file .zip”

  Go

  the section called “Implementazione di archivi di file .zip”

  C#

  the section called “Pacchetto di implementazione”

- Per informazioni su come richiamare una funzione Lambda utilizzandone Servizio AWS un'altra, vedere. Tutorial: uso di un trigger Amazon S3 per richiamare una funzione Lambda
- Scegli uno dei seguenti tutorial per esempi più complessi di utilizzo di Lambda con altri Servizi AWS.
  - Tutorial: uso di Lambda con Amazon API Gateway: crea una REST API di Gateway Amazon API che richiama una funzione Lambda.
  - Utilizzo di una funzione Lambda per accedere a un database Amazon RDS: utilizza una funzione Lambda per scrivere dati in un database Amazon Relational Database Service (Amazon RDS) tramite un proxy RDS.
  - Utilizzo di un trigger Amazon S3 per creare anteprime: utilizza una funzione Lambda per creare un'anteprima ogni volta che un file immagine viene caricato in un bucket Amazon S3.

Guida introduttiva ad applicazioni e modelli di esempio

Le seguenti risorse possono essere utilizzate per creare e distribuire rapidamente app serverless che implementano alcuni casi d'uso comuni di Lambda. Per ciascuna delle app di esempio, forniamo istruzioni per creare e configurare le risorse manualmente utilizzando o AWS Serverless Application Model per distribuire le risorse utilizzando IaC. Console di gestione AWS Segui le istruzioni della console per saperne di più sulla configurazione delle singole AWS risorse per ogni app o usale per distribuire rapidamente le risorse come faresti in un ambiente di produzione. AWS SAM

Elaborazione dei file

- Applicazione di crittografia PDF: crea un'applicazione serverless che crittografa i file PDF quando vengono caricati in un bucket Amazon Simple Storage Service e li salva in un altro bucket, utile per proteggere i documenti sensibili al momento del caricamento.

- Applicazione di analisi delle immagini: crea un'applicazione serverless che estrae il testo dalle immagini utilizzando Amazon Rekognition, utile per l'elaborazione di documenti, la moderazione dei contenuti e l'analisi automatizzata delle immagini.

Integrazione del database

- Queue-to-Database Applicazione: crea un'applicazione serverless che scrive messaggi in coda su un database Amazon RDS, utile per elaborare le registrazioni degli utenti e gestire gli invii degli ordini.

- Database Event Handler: crea un'applicazione serverless che risponda alle modifiche delle tabelle di Amazon DynamoDB, utile per la registrazione degli audit, la replica dei dati e i flussi di lavoro automatizzati.

Attività pianificate

- Applicazione di manutenzione del database: crea un'applicazione serverless che elimina automaticamente le voci più vecchie di 12 mesi da una tabella Amazon DynamoDB utilizzando una

pianificazione cron, utile per la manutenzione automatizzata del database e la gestione del ciclo di vita dei dati.

- Crea una regola EventBridge pianificata per le funzioni Lambda: utilizza le espressioni pianificate per le regole per attivare una funzione Lambda in base EventBridge a una pianificazione temporizzata. Questo formato utilizza la sintassi cron e può essere impostato con una granularità di un minuto.

Flussi di lavoro a lunga durata

- Applicazione di elaborazione degli ordini: crea un'applicazione serverless utilizzando Durable Functions che gestisce l'evasione degli ordini complessi, tra cui l'elaborazione dei pagamenti, i controlli dell'inventario e il coordinamento delle spedizioni. Questo esempio dimostra come creare flussi di lavoro che possono essere eseguiti per periodi prolungati mantenendo lo stato.

Risorse aggiuntive

Utilizza le seguenti risorse per esplorare ulteriormente Lambda e lo sviluppo di applicazioni serverless:

- Serverless Land: una libreria di ready-to-use modelli per la creazione di app serverless. Aiuta gli sviluppatori a creare applicazioni più velocemente utilizzando AWS servizi come Lambda, API Gateway e. EventBridge Il sito offre soluzioni predefinite e best practice, che semplificano lo sviluppo di sistemi serverless.
- Applicazioni di esempio Lambda: applicazioni disponibili nell' GitHub archivio di questa guida. Questi esempi dimostrano l'uso di vari linguaggi e AWS servizi. Ogni applicazione di esempio include script per semplificare l'implementazione, la pulizia e il supporto delle risorse.
- Esempi di codice per l'utilizzo di Lambda AWS SDKs: esempi che mostrano come usare Lambda con i kit di sviluppo AWS software (). SDKs Questi esempi includono nozioni di base, azioni, scenari e contributi della community. AWS Gli esempi riguardano operazioni essenziali, singole funzioni di servizio e attività specifiche che utilizzano più funzioni o AWS servizi.

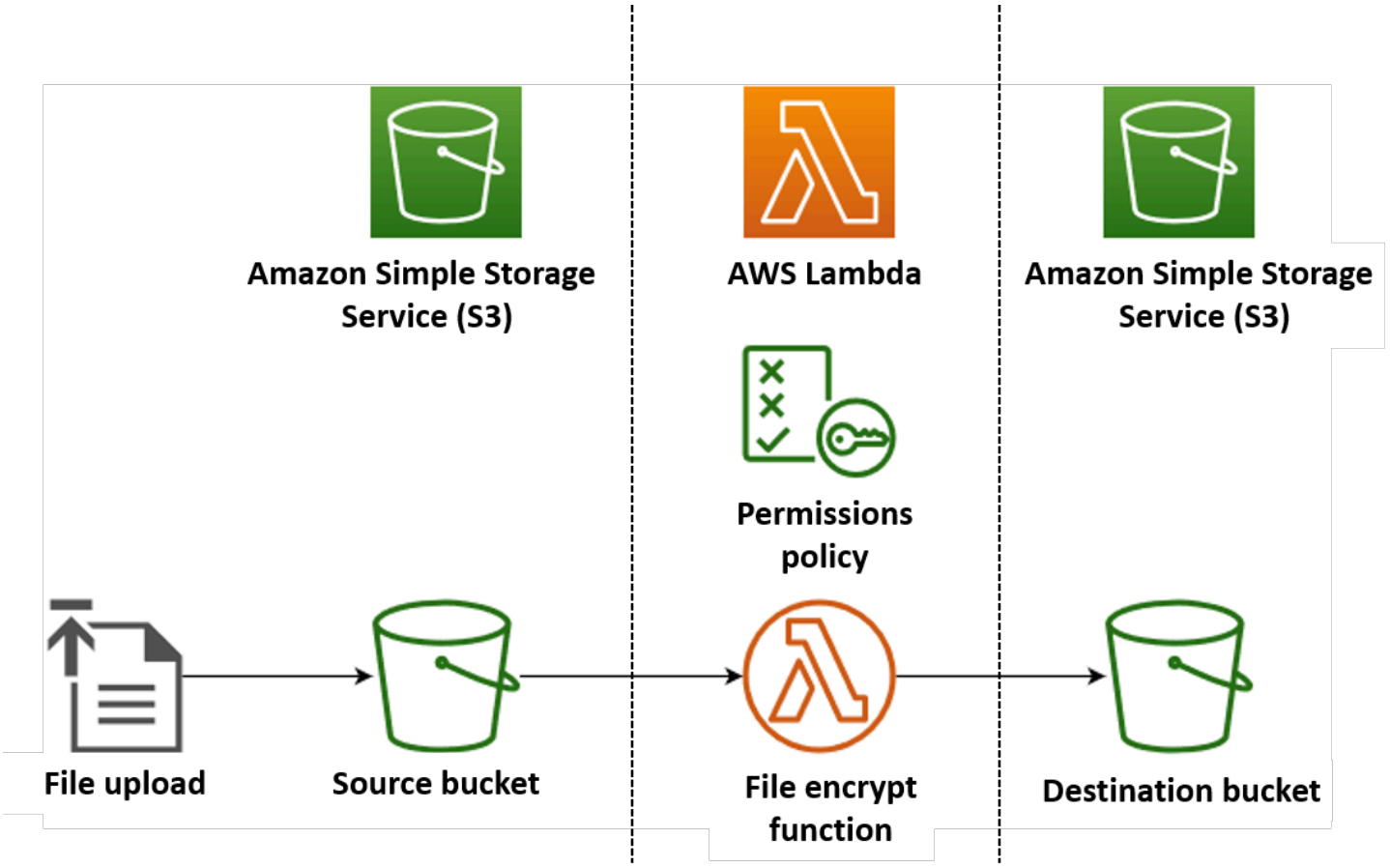
Creare un'app di elaborazione file serverless

Uno dei casi d'uso più comuni di Lambda è l'esecuzione di attività di elaborazione dei file. Ad esempio, è possibile utilizzare una funzione Lambda per creare automaticamente file PDF da file o immagini HTML o per creare miniature quando un utente carica un'immagine.

In questo esempio, viene creata un'app che crittografa automaticamente i file PDF quando vengono caricati in un bucket Amazon Simple Storage Service (Amazon S3). Per creare questa app, crea le risorse seguenti:

- Un bucket S3 in cui gli utenti possono caricare i file PDF
- Una funzione Lambda in Python che legge il file caricato e ne crea una versione crittografata e protetta da password
- Un secondo bucket S3 per Lambda in cui salvare il file crittografato

È inoltre possibile creare una policy AWS Identity and Access Management (IAM) per consentire alla funzione Lambda di eseguire operazioni di lettura e scrittura sui bucket S3.



> **Tip**
>
> Se non conosci Lambda, ti consigliamo di iniziare con il tutorial Crea la tua prima funzione prima di creare questa app di esempio.

Puoi distribuire l'app manualmente creando e configurando risorse con Console di gestione AWS o il AWS Command Line Interface ().AWS CLI Puoi anche distribuire l'app utilizzando (). AWS Serverless Application Model AWS SAM AWS SAM è uno strumento Infrastructure as Code (IaC). Con IaC, non vengono create risorse manualmente, ma le si definisce in codice e poi le si distribuisce automaticamente.

Se desideri saperne di più sull'utilizzo di Lambda con IaC prima di implementare questa app di esempio, consulta the section called “Infrastructure as code (IaC)”.

Creare i file del codice sorgente della funzione Lambda

Crea i seguenti file nella directory del tuo progetto:

- `lambda_function.py`: il codice della funzione Python per la funzione Lambda che esegue la crittografia del database
- `requirements.txt`: un file manifesto che definisce le dipendenze richieste dal codice della funzione Python

Espandi le seguenti sezioni per visualizzare il codice e per saperne di più sul ruolo di ogni file. Per creare i file sul computer locale, copia e incolla il codice seguente oppure scarica i file dal aws-lambda-developer-guide GitHub repository.

Codice della funzione Python

Copia e incolla il codice seguente in un nuovo file `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**
>
> In questo codice di esempio, una password per il file crittografato (`my-secret-password`) è inserita nel codice della funzione. In un'applicazione di produzione, non includere informazioni

sensibili come le password nel codice funzione. Invece, crea un Gestione dei segreti AWS segreto e poi usa l'estensione Lambda AWS Parameters and Secrets per recuperare le credenziali nella funzione Lambda.

Il codice della funzione python contiene tre funzioni: la funzione handler che Lambda esegue quando la funzione viene richiamata e due funzioni separate denominate `add_encrypted_suffix` e `encrypt_pdf` che l'handler chiama per eseguire la crittografia del PDF.

Quando la funzione viene richiamata da Amazon S3, Lambda passa un argomento di evento in formato JSON alla funzione che contiene dettagli sull'evento che ha causato l'invocazione. In questo caso, le informazioni includono il nome del bucket S3 e le chiavi oggetto per i file caricati. Per maggiori informazioni sul formato dell'oggetto evento per Amazon S3, consulta the section called "S3".

La funzione utilizza quindi il AWS SDK per Python (Boto3) per scaricare i file PDF specificati nell'oggetto evento nella relativa directory di archiviazione temporanea locale, prima di crittografarli utilizzando la libreria. `pypdf`

Infine, la funzione utilizza l'SDK Boto3 per archiviare il file crittografato nel bucket di destinazione S3.

File manifesto **`requirements.txt`**

Copia e incolla il codice seguente in un nuovo file `requirements.txt`.

```
boto3
pypdf
```

Per questo esempio, il codice della funzione ha solo due dipendenze che non fanno parte della libreria Python standard: l'SDK per Python (Boto3) e il pacchetto `pypdf` utilizzato dalla funzione per eseguire la crittografia PDF.

> **Note**
>
> Una versione dell'SDK for Python (Boto3) è inclusa come parte del runtime Lambda, quindi il codice può essere eseguito senza aggiungere Boto3 al pacchetto di implementazione della funzione. Tuttavia, per mantenere il pieno controllo delle tue dipendenze ed evitare possibili problemi di disallineamento della versione, la best practice per Python è includere tutte le

dipendenze della funzione nel pacchetto di implementazione della tua funzione. Per ulteriori informazioni, consulta the section called “Dipendenze di runtime in Python”.

Distribuisci l'app

È possibile creare e distribuire le risorse per questa app di esempio manualmente o utilizzando. AWS SAM In un ambiente di produzione, si consiglia di utilizzare uno strumento IaC come quello AWS SAM per distribuire in modo rapido e ripetibile intere applicazioni serverless senza utilizzare processi manuali.

Implementare le risorse manualmente

Per distribuire l'app manualmente:

- Creare bucket Amazon S3 di origine e di destinazione
- Creare una funzione Lambda che crittografa un file PDF e salva la versione crittografata in un bucket S3
- Configurare un trigger Lambda che richiama la tua funzione quando gli oggetti vengono caricati nel bucket di origine

Prima di iniziare, assicurati che Python sia installato sulla tua macchina di compilazione.

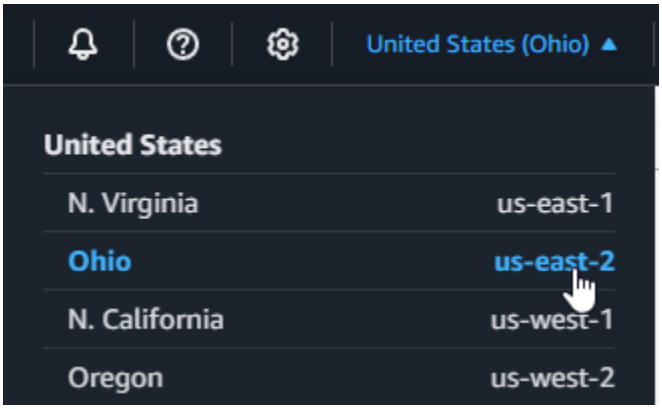
Creare due bucket S3

Per prima cosa, crea due bucket S3. Il primo bucket è il bucket di origine in cui caricherai i tuoi file PDF. Il secondo bucket è utilizzato da Lambda per salvare il file crittografato quando richiami la tua funzione.

Console

Per creare i bucket S3 (console)

1. Apri la pagina General Purpose bucket della console Amazon S3.
2. Seleziona quello più Regione AWS vicino alla tua posizione geografica. Puoi modificare la regione utilizzando l'elenco a discesa nella parte superiore dello schermo.



3. Scegliere Create bucket (Crea bucket).
4. In General configuration (Configurazione generale), eseguire le operazioni seguenti:
   a. Per Tipo di secchio, assicurati che sia selezionata l'opzione Uso generale.
   b. Per Bucket name, inserisci un nome univoco globale che soddisfi le regole di denominazione dei bucket di Amazon S3. I nomi dei bucket possono contenere solo lettere minuscole, numeri, punti (.) e trattini (-).
5. Lascia tutte le altre opzioni impostate sui valori predefiniti e scegli Crea bucket.
6. Ripeti i passaggi da 1 a 4 per creare il bucket di destinazione. Per Nome del bucket, inserisci **`amzn-s3-demo-bucket-encrypted`**, dove **`amzn-s3-demo-bucket`** è il nome del bucket di origine che hai appena creato.

AWS CLI

Prima di iniziare, assicurati che AWS CLI sia installato sul tuo computer di compilazione.

Creazione dei bucket Amazon S3 (AWS CLI)

1. Esegui il comando della CLI sotto riportato per creare il bucket di origine. Il nome che scegli per il tuo bucket deve essere univoco a livello globale e seguire le regole di denominazione dei bucket di Amazon S3. I nomi possono contenere solo lettere minuscole, numeri, punti (.) e trattini (-). Per `region` e `LocationConstraint`, scegli la Regione AWS più vicina alla tua posizione geografica.

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

Più avanti nel tutorial, devi creare la tua funzione Lambda nello Regione AWS stesso bucket di origine, quindi prendi nota della regione che hai scelto.

2. Esegui il comando sotto riportato per creare il bucket di destinazione. Per il nome del bucket, devi usare **`amzn-s3-demo-bucket-encrypted`**, dove **`amzn-s3-demo-bucket`** è il nome del bucket di origine che hai creato nel passaggio 1. Per `region` e`LocationConstraint`, scegli lo stesso Regione AWS che hai usato per creare il bucket sorgente.

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

Creazione di un ruolo di esecuzione

Un ruolo di esecuzione è un ruolo IAM che concede a una funzione Lambda l'autorizzazione all' Servizi AWS accesso e alle risorse. Per concedere alla funzione l'accesso in lettura e scrittura ad Amazon S3, è necessario collegare la policy gestita da AWS `AmazonS3FullAccess`.

Console

Per creare un ruolo di esecuzione e allegare la policy **`AmazonS3FullAccess`** gestita (console)

1. Aprire la pagina Roles (Ruoli) nella console IAM.
2. Scegli Crea ruolo.
3. Per il tipo di entità affidabile, seleziona AWS servizio e per Use case, seleziona Lambda.
4. Scegli Next (Successivo).
5. Aggiungi la politica `AmazonS3FullAccess` gestita effettuando le seguenti operazioni:
   a. Nelle politiche di autorizzazione, **`AmazonS3FullAccess`** accedi alla barra di ricerca.
   b. Seleziona la casella di controllo accanto alla politica.
   c. Scegli Next (Successivo).
6. In Dettagli del ruolo, per il nome del ruolo inserisci**`LambdaS3Role`**.
7. Selezionare Crea ruolo.

AWS CLI

Per creare un ruolo di esecuzione e collegare la policy gestita da **`AmazonS3FullAccess`** (AWS CLI)

1. Salva il seguente JSON in un file denominato `trust-policy.json`. Questa politica di fiducia consente a Lambda di utilizzare le autorizzazioni del ruolo concedendo al servizio principale l'`lambda.amazonaws.com`autorizzazione a chiamare l'azione AWS Security Token Service ()AWS STS. `AssumeRole`

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

2. Nella directory in cui hai salvato il documento della policy di attendibilità JSON, esegui il comando della CLI sotto riportato per creare il ruolo di esecuzione.

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

3. Per collegare la policy gestita da `AmazonS3FullAccess`, esegui il comando della CLI seguente.

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

Creazione del pacchetto di implementazione della funzione

Per creare la funzione, occorre creare un pacchetto di implementazione contenente la funzione e le rispettive dipendenze. Per questa applicazione, il codice della funzione utilizza una libreria separata per la crittografia dei PDF.

Per creare il pacchetto di implementazione

1. Passa alla directory del progetto contenente i `requirements.txt` file `lambda_function.py` e che hai creato o scaricato in GitHub precedenza e crea una nuova directory denominata. `package`
2. Installa le dipendenze specificate nel file `requirements.txt` nella tua directory `package` eseguendo il comando seguente.

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

3. Crea un file .zip contenente il codice dell'applicazione e le relative dipendenze. Su Linux o MacOS, esegui i comandi riportati di seguito dall'interfaccia della linea di comando.

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

   Su Windows, usa il tuo strumento di compressione preferito per creare il file `lambda_function.zip`. Assicurati che il tuo file `lambda_function.py` e le cartelle contenenti le tue dipendenze si trovino tutti nella directory principale del file .zip.

Puoi creare il tuo pacchetto di implementazione anche utilizzando un ambiente virtuale Python. Per informazioni, consultare Utilizzo di archivi di file .zip per le funzioni Lambda in Python.

Creazione della funzione Lambda

Ora usi il pacchetto di implementazione creato nel passaggio precedente per implementare la tua funzione Lambda.

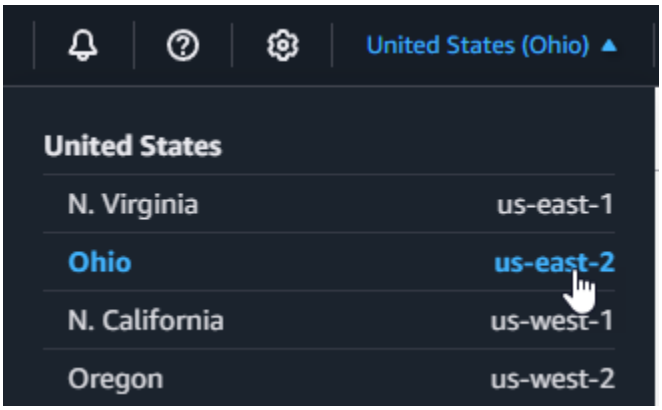
Console

Creazione della funzione (console)

Per creare la tua funzione Lambda utilizzando la console, devi prima creare una funzione di base contenente del codice "Hello world". Quindi, sostituisci questo codice con il codice della tua funzione caricando il file .zip creato nel passaggio precedente.

Per garantire che la funzione non scada durante la crittografia di file PDF di grandi dimensioni, configura le impostazioni di memoria e timeout della funzione. Puoi anche impostare il formato di log della funzione su JSON. La configurazione dei log in formato JSON è necessaria quando si utilizza lo script di test fornito in modo che possa leggere lo stato di chiamata della funzione da CloudWatch Logs per confermare l'avvenuta chiamata.

1. Aprire la pagina Funzioni della console Lambda.
2. Assicurati di lavorare nello stesso bucket in cui hai creato il bucket S3. Regione AWS Puoi modificare la regione utilizzando l'elenco a discesa nella parte superiore dello schermo.



3. Selezionare Create function (Crea funzione).
4. Scegli Author from scratch (Crea da zero).
5. In Basic information (Informazioni di base) eseguire queste operazioni:
   a. Nel campo Function name (Nome funzione), immettere **`EncryptPDF`**.
   b. In Runtime, scegli Python 3.12.
   c. In Architecture (Architettura), scegli x86_64.
6. Allega il ruolo di esecuzione che hai creato nel passaggio precedente effettuando le seguenti operazioni:

a. Espandi la sezione Change default execution role (Cambia ruolo di esecuzione predefinito).

b. Seleziona Utilizza un ruolo esistente.

c. In Ruolo esistente, seleziona il tuo ruolo (`LambdaS3Role`).

7. Scegli Crea funzione.

Caricamento del codice della funzione (console)

1. Nel riquadro Origine del codice, scegli Carica da.
2. Scegli File .zip.
3. Scegli Carica.
4. Nel selettore di file, seleziona il tuo file .zip e scegli Apri.
5. Scegli Save (Salva).

Per configurare la memoria e il timeout della funzione (console)

1. Seleziona la scheda Configurazione per la tua funzione.
2. Nel riquadro Configurazione generale, scegli Modifica.
3. Imposta Memoria su 256 MB e Timeout su 15 secondi.
4. Scegli Save (Salva).

Per configurare il formato di log (console)

1. Seleziona la scheda Configurazione per la tua funzione.
2. Seleziona Strumenti di monitoraggio e operazioni.
3. Nel riquadro Configurazione della registrazione, scegli Modifica.
4. Per Configurazione della registrazione, seleziona JSON.
5. Scegli Save (Salva).

AWS CLI

Creazione della funzione (AWS CLI)

- Esegui il comando seguente dalla directory contenente il file `lambda_function.zip`. Per il parametro `region`, sostituisci `us-east-2` con la regione in cui hai creato i 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
```

Configurare un trigger Amazon S3 per richiamare la funzione

Affinché la funzione Lambda venga eseguita quando carichi un file nel bucket di origine, devi configurare un trigger per la funzione. È possibile configurare il trigger Amazon S3 utilizzando la console Lambda o la AWS CLI.

> **Important**
>
> Questa procedura configura il bucket S3 per richiamare la funzione ogni volta che un oggetto viene creato nel bucket. Assicurati di configurare questa opzione solo sul bucket di origine. Se la tua funzione Lambda crea oggetti nello stesso bucket che la richiama, la tua funzione può essere richiamata continuamente in un ciclo ricorsivo (loop). Ciò può comportare l'addebito di addebiti imprevisti a tuo Account AWS favore.

Console

Configurazione del trigger Amazon S3 (console)

1. Apri la pagina Funzioni della console Lambda e scegli la tua funzione (`EncryptPDF`).
2. Selezionare Add trigger (Aggiungi trigger).
3. Seleziona S3.
4. In Bucket, seleziona il tuo bucket di origine.
5. In Tipi di eventi, seleziona Tutti gli eventi di creazione di oggetti.

6. In Invocazione ricorsiva, seleziona la casella di controllo per confermare che non è consigliabile utilizzare lo stesso bucket S3 per input e output. Per maggiori informazioni sui modelli di invocazione ricorsivi in Lambda, consulta Schemi ricorsivi che causano loop indeterminati delle funzioni Lambda in Serverless Land.
7. Scegliere Aggiungi.

   Quando crei un trigger utilizzando la console Lambda, Lambda crea automaticamente una policy basata sulle risorse per concedere al servizio selezionato l'autorizzazione a richiamare la funzione.

AWS CLI

Configurazione del trigger Amazon S3 (AWS CLI)

1. Aggiungi una policy basata sulle risorse alla tua funzione che consenta al bucket di sorgenti Amazon S3 di richiamare la tua funzione quando aggiungi un file. Una dichiarazione politica basata sulle risorse fornisce altre Servizi AWS autorizzazioni per richiamare la tua funzione. Per autorizzare Amazon S3 a richiamare la tua funzione, esegui il comando della CLI comando sotto riportato. Assicurati di sostituire il `source-account` parametro con il tuo Account AWS ID e di utilizzare il tuo nome del bucket di origine.

```
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
```

   La policy che definisci con questo comando consente ad Amazon S3 di richiamare la tua funzione solo quando viene eseguita un'operazione sul tuo bucket di origine.

> **Note**
>
> Sebbene i nomi dei bucket S3 siano univoci a livello globale, quando utilizzi policy basate sulle risorse è consigliabile specificare che il bucket deve appartenere al tuo account. Questo perché se elimini un bucket, è possibile che un altro lo Account AWS crei con lo stesso Amazon Resource Name (ARN).

2. Salva il seguente JSON in un file denominato `notification.json`. Quando viene applicato al tuo bucket di origine, questo JSON configura il bucket in modo che invii una notifica alla funzione Lambda ogni volta che viene aggiunto un nuovo oggetto. Sostituisci il Account AWS numero e Regione AWS nella funzione Lambda ARN con il tuo numero di account e la tua regione.

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

3. Esegui il comando della CLI comando sotto riportato per applicare le impostazioni di notifica nel file JSON che hai creato al tuo bucket di origine. Sostituisci `amzn-s3-demo-bucket` con il nome del tuo bucket di origine.

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

   Per ulteriori informazioni sul `put-bucket-notification-configuration` comando e sull'`notification-configuration`opzione, consulta put-bucket-notification-configurationla AWS CLI Command Reference.

Distribuisci le risorse utilizzando AWS SAM

Prima di iniziare, assicurati che Docker e la versione più recente di AWS SAMCLI siano installati sulla tua macchina di compilazione.

1. Nella directory del progetto, copia e incolla il seguente codice in un file denominato`template.yaml`. Sostituisci i nomi dei bucket segnaposto:

   - Per il bucket di origine, sostituiscilo `amzn-s3-demo-bucket` con qualsiasi nome conforme alle regole di denominazione dei bucket S3.

- Per il bucket di destinazione, sostituisci `amzn-s3-demo-bucket-encrypted` con `<source-bucket-name>-encrypted`, dove `<source-bucket>` è il nome che hai scelto per il bucket di origine.

```
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
```

Il AWS SAM modello definisce le risorse che crei per la tua app. In questo esempio, il modello definisce una funzione Lambda utilizzando il tipo `AWS::Serverless::Function` e due

bucket S3 che utilizzano il tipo `AWS::S3::Bucket`. I nomi dei bucket specificati nel modello sono segnaposti. Prima di distribuire l'app utilizzando AWS SAM, devi modificare il modello per rinominare i bucket con nomi univoci a livello globale che soddisfino le regole di denominazione dei bucket S3. Questo passaggio viene spiegato ulteriormente in the section called "Distribuisci le risorse utilizzando AWS SAM".

La definizione della risorsa della funzione Lambda configura un trigger per la funzione utilizzando la proprietà dell'evento `S3Event`. Questo trigger fa sì che la funzione venga richiamata ogni volta che viene creato un oggetto nel bucket di origine.

La definizione della funzione specifica anche una policy AWS Identity and Access Management (IAM) da allegare al ruolo di esecuzione della funzione. La policy gestita da AWS `AmazonS3FullAccess` fornisce alla tua funzione le autorizzazioni necessarie per leggere e scrivere oggetti su Amazon S3.

2. Eseguire il comando seguente nella directory in cui sono stati salvati i file `template.yaml`, `lambda_function.py` e `requirements.txt`.

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

Questo comando raccoglie gli artefatti di compilazione per l'applicazione e li colloca nel formato e nella posizione corretti per l'implementazione. La specifica dell'opzione `--use-container` consente di creare la funzione all'interno di un container Docker simile a Lambda. Lo usiamo qui quindi non è necessario che Python 3.12 sia installato sul computer locale per far funzionare la build.

Durante il processo di compilazione, AWS SAM cerca il codice della funzione Lambda nella posizione specificata con la `CodeUri` proprietà nel modello. In questo caso, abbiamo specificato la directory corrente come posizione (`./`).

Se è presente un `requirements.txt` file, lo AWS SAM usa per raccogliere le dipendenze specificate. Per impostazione predefinita, AWS SAM crea un pacchetto di distribuzione.zip con il codice della funzione e le dipendenze. Puoi anche scegliere di distribuire la tua funzione come immagine contenitore utilizzando la proprietà. PackageType

3. Per distribuire l'applicazione e creare le risorse Lambda e Amazon S3 specificate nel modello, AWS SAM esegui il comando seguente.

```
sam deploy --guided
```

L'uso del `--guided` flag significa che ti AWS SAM verranno mostrate le istruzioni per guidarti attraverso il processo di distribuzione. Per questa implementazione, accetta le opzioni predefinite premendo Invio.

Durante il processo di distribuzione, AWS SAM crea le seguenti risorse nel tuo: Account AWS

- Una CloudFormation pila denominata `sam-app`
- Una funzione Lambda con il nome `EncryptPDF`
- Due bucket S3 con i nomi che hai scelto quando hai modificato il file modello `template.yaml` AWS SAM
- Un ruolo di esecuzione IAM per la tua funzione con il formato del nome `sam-app-EncryptPDFFunctionRole-2qGaapHFW0Q8`

Al AWS SAM termine della creazione delle risorse, dovresti visualizzare il seguente messaggio:

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

Esecuzione del test dell'app

Per testare l'app, carica un file PDF nel bucket di origine e conferma che Lambda crei una versione crittografata del file nel bucket di destinazione. In questo esempio, puoi testarlo manualmente utilizzando la console o il AWS CLI, oppure utilizzando lo script di test fornito.

Per le applicazioni di produzione, puoi utilizzare metodi e tecniche di test tradizionali, come il test di unità, per confermare il corretto funzionamento del codice di funzione Lambda. La best practice consiste anche nell'eseguire test come quelli dello script di test fornito, che eseguono test di integrazione con risorse reali basate sul cloud. I test di integrazione nel cloud confermano che l'infrastruttura è stata implementata correttamente e che gli eventi fluiscono tra i diversi servizi come previsto. Per ulteriori informazioni, consulta Test di funzioni serverless.

Test manuale dell'app

Puoi testare la tua funzione manualmente aggiungendo un file PDF al bucket di origine Amazon S3. Quando aggiungi un file al bucket di origine, la tua funzione Lambda dovrebbe essere richiamata automaticamente e dovrebbe memorizzare una versione crittografata del file nel bucket di destinazione.

Console

Per testare l'app caricando un file (console)

1. Per caricare un file PDF nel bucket S3, procedi come segue:
   a. Apri la pagina Bucket della console Amazon S3 e scegli il bucket di origine.
   b. Scegli Carica.
   c. Scegli Aggiungi file e utilizza il selettore di file per scegliere il file PDF da caricare.
   d. Seleziona Apri, quindi Carica.
2. Verifica che Lambda abbia salvato una versione crittografata del tuo file PDF nel bucket di destinazione effettuando le seguenti operazioni:
   a. Torna alla pagina Bucket della console Amazon S3 e scegli il bucket di destinazione.
   b. Nel riquadro Oggetti, ora dovresti vedere un file con il formato del nome `filename_encrypted.pdf` (dove `filename.pdf` era il nome del file che hai caricato nel bucket di origine). Per scaricare il PDF crittografato, seleziona il file, quindi scegli Scarica.
   c. Conferma di poter aprire il file scaricato con la password con cui la funzione Lambda lo ha protetto (`my-secret-password`).

AWS CLI

Per testare l'app caricando un file (AWS CLI)

1. Dalla directory contenente il file PDFche desideri caricare, esegui il comando della CLI sotto riportato. Sostituisci il parametro `--bucket` con il nome del bucket di origine. Per i parametri `--key` e `--body`, usa il nome del file di test.

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

2. Verifica che la funzione abbia creato una versione crittografata del file e l'abbia salvata nel bucket S3 di destinazione. Esegui il comando della CLI sotto riportato sostituendo `amzn-s3-demo-bucket-encrypted` con il nome del tuo bucket di destinazione.

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

Se la tua funzione viene eseguita correttamente, vedrai un output simile al seguente. Il bucket di destinazione deve contenere un file con il formato del nome *<your_test_file>*_encrypted.pdf, dove <your_test_file> dov'è il nome del file che hai caricato.

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

3. Per scaricare il file salvato da Lambda nel bucket di destinazione, esegui il comando della CLI sotto riportato. Sostituire il parametro `--bucket` con il nome del tuo bucket di destinazione. Per il parametro `--key`, usa il nome del file *<your_test_file>*_encrypted.pdf, dove <your_test_file> è il nome del file di test che hai caricato.

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

Questo comando scarica il file nella directory corrente e lo salva come `my_encrypted_file.pdf`.

4. Conferma di poter aprire il file scaricato con la password con cui la funzione Lambda lo ha protetto (`my-secret-password`).

Test dell'app con lo script automatico

Crea i seguenti file nella directory del tuo progetto:

- `test_pdf_encrypt.py`: uno script di test che puoi utilizzare per testare automaticamente l'applicazione
- `pytest.ini`: un file di configurazione per lo script di test

Espandi le seguenti sezioni per visualizzare il codice e per saperne di più sul ruolo di ogni file.

Script di test automatizzato

Copia e incolla il codice seguente in un nuovo file `test_pdf_encrypt.py`. Assicurati di sostituire i nomi dei bucket segnaposto:

- Nella funzione `test_source_bucket_available`, sostituisci `amzn-s3-demo-bucket` con il nome del tuo bucket di origine.
- Nella funzione `test_encrypted_file_in_bucket`, sostituisci `amzn-s3-demo-bucket-encrypted` con *`source-bucket`*`-encrypted`, dove `source-bucket>` è il nome del tuo bucket di origine.
- Nella `cleanup` funzione, sostituisci `amzn-s3-demo-bucket` con il nome del bucket di origine e sostituisci `amzn-s3-demo-bucket-encrypted` con il nome del bucket di destinazione.

```
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
```

Lo script di test automatizzato esegue tre funzioni di test per confermare il corretto funzionamento dell'app:

- Il test `test_source_bucket_available` conferma che il bucket di origine è stato creato correttamente caricando un file PDF di prova nel bucket.

- Il test `test_lambda_invoked` interroga il flusso di log di CloudWatch Logs più recente della funzione per confermare che quando hai caricato il file di test, la funzione Lambda è stata eseguita e ha segnalato il successo.
- Il test `test_encrypted_file_in_bucket` conferma che il bucket di destinazione contiene il file crittografato `test_encrypted.pdf`.

Dopo l'esecuzione di tutti questi test, lo script esegue un ulteriore passaggio di pulizia per eliminare i file `test.pdf` e `test_encrypted.pdf` dai bucket di origine e di destinazione.

Come nel AWS SAM modello, i nomi dei bucket specificati in questo file sono segnaposto. Prima di eseguire il test, devi modificare questo file con i nomi reali dei bucket dell'app. Questo passaggio viene spiegato ulteriormente in the section called “Test dell'app con lo script automatico”.

File di configurazione dello script di test

Copia e incolla il codice seguente in un nuovo file `pytest.ini`.

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

È necessario per specificare l'ordine in cui vengono eseguiti i test nello script `test_pdf_encrypt.py`.

Per eseguire i test, procedere come segue:

1. Assicurati che il `pytest` modulo sia installato nel tuo ambiente locale. È possibile installare `pytest` eseguendo il comando seguente:

   ```
   pip install pytest
   ```

2. Salvate un file PDF denominato `test.pdf` nella directory contenente i `pytest.ini` file `test_pdf_encrypt.py` and.
3. Apri un programma di terminale o di shell ed esegui il comando sotto riportato dalla directory contenente i file di test.

   ```
   pytest -s -v
   ```

   Una volta completato il test, l'output dovrebbe essere simile al seguente:

```
============================================================= 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
 ==============================================================
```

Fasi successive

Ora che hai creato questa app di esempio, puoi utilizzare il codice fornito come base per creare altri tipi di applicazioni per l'elaborazione di file. Modifica il codice nel file `lambda_function.py` per implementare la logica di elaborazione dei file per il tuo caso d'uso.

Numerosi casi d'uso tipici dell'elaborazione di file riguardano l'elaborazione delle immagini. Quando si usa Python, le librerie di elaborazione delle immagini più popolari come pillow contengono in genere componenti C o C++. Per garantire che il pacchetto di implementazione della funzione sia compatibile con l'ambiente di esecuzione Lambda, è importante utilizzare il binario di distribuzione del codice sorgente corretto.

Quando si distribuiscono le risorse con AWS SAM, è necessario adottare alcune misure aggiuntive per includere la corretta distribuzione dei sorgenti nel pacchetto di distribuzione. Poiché AWS SAM non installerà dipendenze per una piattaforma diversa dalla macchina di compilazione, specificare la corretta distribuzione del codice sorgente (`.whl`file) nel `requirements.txt` file non funzionerà

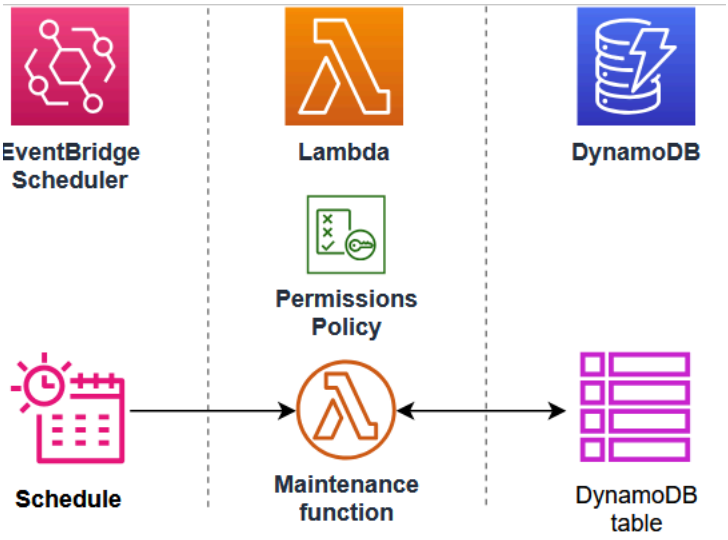
se la macchina di compilazione utilizza un sistema operativo o un'architettura diversi dall'ambiente di esecuzione Lambda. Invece, effettua una delle seguenti operazioni:

- Usa l'opzione `--use-container` durante l'esecuzione di `sam build`. Quando specifichi questa opzione, AWS SAM scarica un'immagine di base del contenitore compatibile con l'ambiente di esecuzione Lambda e crea il pacchetto di distribuzione della funzione in un contenitore Docker utilizzando quell'immagine. Per ulteriori informazioni, consulta Creazione di una funzione Lambda all'interno di un container fornito.
- Crea tu stesso il pacchetto di distribuzione.zip della tua funzione utilizzando il file binario di distribuzione sorgente corretto e salva il file.zip nella directory specificata come nel modello. `CodeUri` AWS SAM Per ulteriori informazioni sulla creazione di pacchetti di implementazione .zip per Python utilizzando distribuzioni binarie, consulta the section called “Creazione di un pacchetto di implementazione .zip con dipendenze” e the section called “Creazione di un pacchetto di implementazione .zip con librerie native”.

Creare un'app per eseguire la manutenzione pianificata del database

Puoi utilizzarlo AWS Lambda per sostituire processi pianificati come backup automatici di sistema, conversioni di file e attività di manutenzione. In questo esempio, viene creata un'applicazione serverless che esegue una manutenzione programmata regolare su una tabella DynamoDB eliminando le voci precedenti. L'app utilizza EventBridge Scheduler per richiamare una funzione Lambda su una pianificazione cron. Quando viene richiamata, la funzione interroga la tabella per individuare gli elementi più vecchi di un anno e li elimina. La funzione registra ogni elemento eliminato in Logs. CloudWatch

Per implementare questo esempio, crea prima una tabella DynamoDB e completala con alcuni dati di test per la funzione da interrogare. Quindi, crea una funzione Python Lambda con un trigger EventBridge Scheduler e un ruolo di esecuzione IAM che dia alla funzione il permesso di leggere ed eliminare gli elementi dalla tua tabella.



> **Tip**
>
> Se non hai familiarità con Lambda, completa il tutorial Crea la tua prima funzione prima di creare questa app di esempio.

Puoi distribuire la tua app manualmente creando e configurando risorse con. Console di gestione AWS Puoi anche distribuire l'app utilizzando (). AWS Serverless Application Model AWS SAM AWS SAM è uno strumento Infrastructure as Code (IaC). Con IaC, non vengono create risorse manualmente, ma le si definisce in codice e poi le si distribuisce automaticamente.

Se desideri saperne di più sull'utilizzo di Lambda con IaC prima di implementare questa app di esempio, consulta the section called “Infrastructure as code (IaC)”.

Prerequisiti

Prima di creare l'app di esempio, assicurati di aver installato gli strumenti e i programmi da riga di comando necessari.

- Python

  Per popolare la tabella DynamoDB creata per testare l'app, questo esempio utilizza uno script Python e un file CSV per scrivere i dati nella tabella. Assicurati che Python versione 3.8 o successiva sia installato sulla tua macchina.

- AWS SAM CLI

Se desideri creare la tabella DynamoDB e distribuire l'app di esempio AWS SAM utilizzando, devi installare la CLI. AWS SAM Segui le istruzioni di installazione nella Guida per l'utente di AWS SAM .

- AWS CLI

  Per utilizzare lo script Python fornito per popolare la tabella di test, è necessario aver installato e configurato il AWS CLI. Questo perché lo script utilizza il AWS SDK per Python (Boto3), che richiede l'accesso alle tue credenziali AWS Identity and Access Management (IAM). È inoltre necessario che AWS CLI sia installato per distribuire le risorse utilizzando. AWS SAM Installa la CLI seguendo le istruzioni di installazione riportate nella Guida per l'utente di AWS Command Line Interface .

- Docker

  Per distribuire l'app utilizzando AWS SAM, Docker deve essere installato anche sulla macchina di compilazione. Segui le istruzioni in Installa Docker Engine sul sito Web della documentazione di Docker.

Download dei file dell'app di esempio

Per creare il database di esempio e l'app di manutenzione programmata, è necessario creare i seguenti file nella directory del progetto:

File di database di esempio

- `template.yaml`- un AWS SAM modello che puoi usare per creare la tabella DynamoDB
- `sample_data.csv`: un file CSV contenente dati di esempio da caricare nella tabella
- `load_sample_data.py`: uno script Python che scrive i dati del file CSV nella tabella

File dell'app di manutenzione programmata

- `lambda_function.py`: il codice della funzione Python per la funzione Lambda che esegue la manutenzione del database
- `requirements.txt`: un file manifesto che definisce le dipendenze richieste dal codice della funzione Python
- `template.yaml`- un AWS SAM modello che puoi usare per distribuire l'app

File di test

- `test_app.py`: uno script Python che scansiona la tabella e conferma il corretto funzionamento della funzione emettendo tutti i record più vecchi di un anno

Espandi le seguenti sezioni per visualizzare il codice e per saperne di più sul ruolo di ciascun file nella creazione e nel test dell'app. Per creare i file sul tuo computer locale, copia e incolla il codice seguente.

AWS SAM modello (tabella DynamoDB di esempio)

Copia e incolla il codice seguente in un nuovo file `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 i modelli utilizzano una convenzione di denominazione standard di. `template.yaml` In questo esempio, hai due file modello: uno per creare il database di esempio e l'altro per creare l'app stessa. Salvali in sottodirectory separate nella cartella di progetto.

Questo AWS SAM modello definisce la risorsa della tabella DynamoDB che crei per testare la tua app. La tabella utilizza una chiave primaria di `Order_number` con una chiave di ordinamento di `Date`. Affinché la funzione Lambda trovi gli elementi direttamente in base alla data, definiamo anche un indice secondario globale denominato `Date-index`.

Per ulteriori informazioni sulla creazione e la configurazione di una tabella DynamoDB tramite la risorsa `AWS::DynamoDB::Table`, consulta AWS::DynamoDB::Table nella Guida per l'utente di AWS CloudFormation .

File di dati del database di esempio

Copia e incolla il codice seguente in un nuovo file `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
```

Questo file contiene alcuni esempi di dati di test con cui compilare la tabella DynamoDB in formato CSV (valori separati da virgola) standard.

Script Python per caricare dati di esempio

Copia e incolla il codice seguente in un nuovo file `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.")
```

Questo script Python utilizza innanzitutto il AWS SDK per Python (Boto3) per creare una connessione alla tabella DynamoDB. Quindi esegue un'iterazione su ogni riga del file CSV di dati di esempio, crea un elemento da quella riga e scrive l'elemento nella tabella DynamoDB utilizzando l'SDK boto3.

Codice della funzione Python

Copia e incolla il codice seguente in un nuovo file `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'
    }
```

Il codice della funzione Python contiene la funzione handler (`lambda_handler`) che Lambda esegue quando viene invocata la funzione.

Quando la funzione viene richiamata da EventBridge Scheduler, utilizza il AWS SDK per Python (Boto3) per creare una connessione alla tabella DynamoDB su cui deve essere eseguita l'attività di manutenzione pianificata. Quindi utilizza la libreria `datetime` Python per calcolare la data di un anno fa, prima di scansionare la tabella alla ricerca di elementi più vecchi di questo ed eliminarli.

Tieni presente che le risposte delle operazioni di query e scansione di DynamoDB sono limitate a una dimensione massima di 1 MB. Se la risposta è superiore a 1 MB, DynamoDB impagina i dati e restituisce un elemento `LastEvaluatedKey` nella risposta. Per garantire che la nostra funzione elabori tutti i record della tabella, controlliamo la presenza di questa chiave e continuiamo a eseguire scansioni della tabella dall'ultima posizione valutata fino alla scansione dell'intera tabella.

File manifesto **`requirements.txt`**

Copia e incolla il codice seguente in un nuovo file `requirements.txt`.

```
boto3
```

Per questo esempio, il codice della funzione ha solo una dipendenza che non fa parte della libreria Python standard: l'SDK for Python (Boto3) che la funzione utilizza per scansionare ed eliminare elementi dalla tabella DynamoDB.

> **Note**
>
> Una versione dell'SDK for Python (Boto3) è inclusa come parte del runtime Lambda, quindi il codice può essere eseguito senza aggiungere Boto3 al pacchetto di implementazione della funzione. Tuttavia, per mantenere il pieno controllo delle tue dipendenze ed evitare possibili problemi di disallineamento della versione, la best practice per Python è includere tutte le

dipendenze della funzione nel pacchetto di implementazione della tua funzione. Per ulteriori informazioni, consulta the section called “Dipendenze di runtime in Python”.

AWS SAM modello (app per manutenzione programmata)

Copia e incolla il codice seguente in un nuovo file `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 i modelli utilizzano una convenzione di denominazione standard di. `template.yaml` In questo esempio, hai due file modello: uno per creare il database di esempio e l'altro per creare l'app stessa. Salvali in sottodirectory separate nella cartella di progetto.

Questo AWS SAM modello definisce le risorse per la tua app. Definiamo la funzione Lambda utilizzando la risorsa `AWS::Serverless::Function`. La EventBridge pianificazione dello Scheduler e il trigger per richiamare la funzione Lambda vengono creati utilizzando la `Events` proprietà di questa risorsa utilizzando un tipo di. `ScheduleV2` Per ulteriori informazioni sulla definizione delle EventBridge pianificazioni di Scheduler nei AWS SAM modelli, consulta ScheduleV2 nella Guida per gli sviluppatori.AWS Serverless Application Model

Oltre alla funzione Lambda e alla EventBridge pianificazione Scheduler, definiamo anche un gruppo di CloudWatch log a cui la funzione può inviare i record degli elementi eliminati.

Script di test

Copia e incolla il codice seguente in un nuovo file `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)}")
```

Questo script di test utilizza il AWS SDK per Python (Boto3) per creare una connessione alla tabella DynamoDB ed eseguire la scansione degli elementi più vecchi di un anno. Per confermare se la funzione Lambda è stata eseguita correttamente, alla fine del test, la funzione stampa il numero di record più vecchi di un anno ancora nella tabella. Se la funzione Lambda ha avuto successo, il numero di vecchi record nella tabella dovrebbe essere zero.

Creazione e compilazione della tabella DynamoDB di esempio

Per testare la tua app di manutenzione programmata, devi prima creare una tabella DynamoDB e popolarla con alcuni dati di esempio. È possibile creare la tabella manualmente utilizzando o la Console di gestione AWS o AWS SAM. Si consiglia di AWS SAM utilizzarlo per creare e configurare rapidamente la tabella utilizzando alcuni AWS CLI comandi.

Console

Creare la tabella DynamoDB

1. Aprire la pagina Tables (Tabelle) della console DynamoDB.
2. Scegliere Create table (Crea tabella).
3. Crea la tabella completando le operazioni seguenti:
   a. In Dettagli tabella, per Nome tabella, immetti **`MyOrderTable`**.
   b. Per Partition key (Chiave di partizione), immettere **`Order_number`** e mantenere il tipo di dati impostato come String (Stringa).
   c. Per Chiave di ordinamento, immetti **`Date`** e lascia il tipo come String.
   d. Lascia Impostazioni della tabella impostato su Impostazioni predefinite e scegli Crea tabella.
4. Una volta completata la creazione della tabella e lo stato è impostato su Attivo, crea un indice secondario globale (GSI) procedendo come segue. L'app utilizzerà questo GSI per cercare gli elementi direttamente in base alla data e determinare cosa eliminare.
   a. Scegliete MyOrderTabledall'elenco delle tabelle.
   b. Scegli la scheda Indici.
   c. In Indici secondari globali, scegli Crea indice.

d. In Dettagli dell'indice, inserisci **`Date`** per Chiave di partizione e lascia il campo Tipo di dati impostato su Stringa.

e. Per Index name (Nome indice), inserisci **`Date-index`**.

f. Lascia tutti gli altri parametri impostati sui valori predefiniti, scorri fino alla parte inferiore della pagina e scegli Crea indice.

AWS SAM

Creare la tabella DynamoDB

1. Passa alla cartella in cui è stato salvato il file `template.yaml` per la tabella DynamoDB. Nota che questo esempio utilizza due file `template.yaml`. Assicurati che siano salvati in sottocartelle separate e di trovarti nella cartella corretta contenente il modello per creare la tua tabella DynamoDB.
2. Eseguire il seguente comando seguente.

```
sam build
```

Questo comando raccoglie gli artefatti di compilazione per l'applicazione e li colloca nel formato e nella posizione corretti per l'implementazione.

3. Per creare la risorsa DynamoDB specificata nel file `template.yaml`, eseguire il comando seguente.

```
sam deploy --guided
```

L'uso del `--guided` flag significa che ti AWS SAM verranno mostrate le istruzioni per guidarti nel processo di distribuzione. Per questa implementazione, inserisci un `Stack name` pari a **`cron-app-test-db`** e accetta i valori predefiniti per tutte le altre opzioni utilizzando Invio.

Al AWS SAM termine della creazione della risorsa DynamoDB, dovrebbe apparire il seguente messaggio.

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

4. È inoltre possibile confermare che la tabella DynamoDB è stata creata aprendo la pagina Tabelle della console DynamoDB. Dovresti vedere una tabella denominata `MyOrderTable`.

Dopo aver creato la tabella, aggiungi alcuni dati di esempio per testare l'app. Il file CSV `sample_data.csv` scaricato in precedenza contiene una serie di voci di esempio tra cui numeri d'ordine, date e informazioni su clienti e ordini. Usa lo script python `load_sample_data.py` fornito per aggiungere questi dati alla tua tabella.

Per aggiungere i dati di esempio alla tabella

1. Passa alla directory contenente i file `sample_data.csv` e `load_sample_data.py`. Se questi file si trovano in directory separate, spostali in modo che vengano salvati nella stessa posizione.
2. Crea un ambiente virtuale Python in cui eseguire lo script mediante il comando seguente. Ti consigliamo di utilizzare un ambiente virtuale perché nel passaggio successivo dovrai installare AWS SDK per Python (Boto3).

   ```
   python -m venv venv
   ```
3. Nell'ambiente virtuale, esegui il comando seguente.

   ```
   source venv/bin/activate
   ```
4. Installa l'SDK per Python (Boto3) nell'ambiente virtuale mediante il comando seguente. Lo script utilizza questa libreria per connettersi alla tabella DynamoDB e aggiungere gli elementi.

   ```
   pip install boto3
   ```
5. Esegui lo script per popolare la tabella mediante il comando seguente.

   ```
   python load_sample_data.py
   ```

   Se lo script viene eseguito correttamente, dovrebbe stampare ogni elemento sulla console man mano che lo carica e riporta `Data loading completed`.
6. Per disattivare l'ambiente virtuale, esegui il comando seguente.

   ```
   deactivate
   ```
7. È possibile verificare che i dati siano stati caricati nella tabella DynamoDB eseguendo le operazioni seguenti:

   a. Apri la pagina Esplora elementi della console DynamoDB e seleziona la tua tabella (`MyOrderTable`).

b. Nel riquadro Elementi restituiti, dovresti vedere i 25 elementi del file CSV che lo script ha aggiunto alla tabella.

Creazione dell'app per la manutenzione programmata

È possibile creare e distribuire le risorse per questa app di esempio passo dopo passo utilizzando Console di gestione AWS o utilizzando. AWS SAM In un ambiente di produzione, si consiglia di utilizzare uno strumento Infrustracture-as-Code (IaC), AWS SAM ad esempio per distribuire ripetutamente applicazioni serverless senza utilizzare processi manuali.

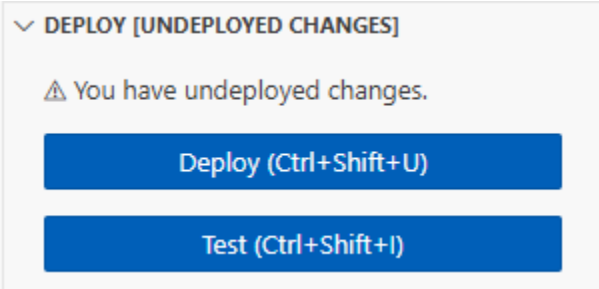
Per questo esempio, segui le istruzioni della console per scoprire come configurare ogni AWS risorsa separatamente oppure segui AWS SAM le istruzioni per distribuire rapidamente l'app utilizzando i comandi. AWS CLI

Console

Per creare la funzione utilizzando il Console di gestione AWS

Innanzitutto, crea una funzione contenente il codice di avvio di base. Quindi, sostituisci questo codice con il codice della tua funzione copiando e incollando il codice direttamente nell'editor di codice Lambda o caricandolo come pacchetto `.zip`. Per questa operazione, si consiglia di copiare e incollare il codice.

1. Aprire la pagina Funzioni della console Lambda.
2. Selezionare Create function (Crea funzione).
3. Scegli Author from scratch (Crea da zero).
4. In Basic information (Informazioni di base) eseguire queste operazioni:
   a. Nel campo Function name (Nome funzione), immettere **`ScheduledDBMaintenance`**.
   b. Per Runtime scegli l'ultima versione di Python.
   c. In Architecture (Architettura), scegli x86_64.
5. Scegli Crea funzione.
6. Dopo aver creato la funzione, è possibile configurarla con il codice di funzione fornito.
   a. Nel riquadro Codice sorgente, sostituisci il codice Hello world creato da Lambda con il codice della funzione Python dal file `lambda_function.py` salvato in precedenza.
   b. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



Per configurare la memoria e il timeout della funzione (console)

1. Seleziona la scheda Configurazione per la tua funzione.
2. Nel riquadro Configurazione generale, scegli Modifica.
3. Imposta Memoria su 256 MB e Timeout su 15 secondi. Se stai elaborando una tabella di grandi dimensioni con molti record, ad esempio nel caso di un ambiente di produzione, potresti prendere in considerazione l'impostazione di Timeout su un numero maggiore. In questo modo la funzione ha più tempo per scansionare e pulire il database.
4. Scegli Save (Salva).

Per configurare il formato di log (console)

Puoi configurare le funzioni Lambda per generare log in formato testo non strutturato o JSON. È consigliabile utilizzare il formato JSON per i log per agevolare la ricerca e il filtraggio dei dati di log. Per ulteriori informazioni sulle opzioni di configurazione del log Lambda, consulta the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”.

1. Seleziona la scheda Configurazione per la tua funzione.
2. Seleziona Strumenti di monitoraggio e operazioni.
3. Nel riquadro Configurazione della registrazione, scegli Modifica.
4. Per Configurazione della registrazione, seleziona JSON.
5. Scegli Save (Salva).

Impostazione delle autorizzazioni IAM

Per concedere alla funzione le autorizzazioni necessarie per leggere ed eliminare gli elementi di DynamoDB, è necessario aggiungere una policy al ruolo di esecuzione della funzione che definisca le autorizzazioni necessarie.

1. Apri la scheda Configurazione, quindi scegli Autorizzazioni dalla barra di navigazione a sinistra.
2. Scegli il nome del ruolo Ruolo di esecuzione.
3. Nella console IAM, scegli Aggiungi autorizzazioni, quindi Crea policy in linea.
4. Utilizza l'editor JSON e inserisci la seguente policy:

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

5. Assegna un nome alla policy **`DynamoDBCleanupPolicy`**, quindi creala.

Per configurare EventBridge Scheduler come trigger (console)

1. Apri la EventBridge console.
2. Nel riquadro di navigazione a sinistra, scegli Pianificatori nella sezione Pianificatore.
3. Scegli Crea pianificazione.
4. Configura le impostazioni del VPC effettuando le seguenti operazioni:

   a. Per Nome pianificazione, inserisci un nome per la pianificazione (ad esempio, **`DynamoDBCleanupSchedule`**).

   b. In Schema di pianificazione, scegli Pianificazione ricorrente.

   c. Per Tipo di pianificazione, lascia l'impostazione predefinita come Pianificazione basata su CRON, quindi inserisci i seguenti dettagli di pianificazione:

      - minuti**`0`**

- Ore: **3**
- Giorno del mese: **1**
- Mese: *****
- Giorno della settimana: **?**
- Anno: *****

Una volta valutata, questa espressione cron viene eseguita il primo giorno di ogni mese alle 03:00.

d. Per Finestra temporale flessibile, seleziona Off.

5. Scegli Next (Successivo).
6. Configura il trigger per la tua funzione Lambda eseguendo le operazioni seguenti:

   a. Nel riquadro Dettagli della destinazione, lascia il campo API di destinazione impostato su Destinazioni basate su modelli, quindi seleziona Invocazione di AWS Lambda .

   b. In Richiama, seleziona la tua funzione Lambda (`ScheduledDBMaintenance`) dall'elenco a discesa.

   c. Lascia vuoto il campo Payload e scegli Avanti.

   d. Scorri verso il basso fino a Autorizzazioni e seleziona Crea un nuovo ruolo per questa pianificazione. Quando crei una nuova EventBridge pianificazione di Scheduler utilizzando la console, EventBridge Scheduler crea una nuova politica con le autorizzazioni necessarie alla pianificazione per richiamare la tua funzione. Per ulteriori informazioni sulla gestione delle autorizzazioni di pianificazione, consulta Pianificazioni basate su CRON nella Guida per l'utente di Scheduler. EventBridge

   e. Scegli Next (Successivo).

7. Controlla le impostazioni e scegli Crea pianificazione per completare la creazione della pianificazione e del trigger Lambda.

AWS SAM

Per distribuire l'app utilizzando AWS SAM

1. Passa alla cartella in cui è stato salvato il file `template.yaml` per l'app. Nota che questo esempio utilizza due file `template.yaml`. Assicurati che siano salvati in sottocartelle separate e di trovarti nella cartella corretta contenente il modello per creare l'app.

2. Copia i file `lambda_function.py` e `requirements.txt` che hai scaricato in precedenza nella stessa cartella. La posizione del codice specificata nel AWS SAM modello è `./`, ovvero la posizione corrente. AWS SAM cercherà in questa cartella il codice della funzione Lambda quando tenti di distribuire l'app.
3. Eseguire il seguente comando seguente.

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

   Questo comando raccoglie gli artefatti di compilazione per l'applicazione e li colloca nel formato e nella posizione corretti per l'implementazione. La specifica dell'opzione `--use-container` consente di creare la funzione all'interno di un container Docker simile a Lambda. Lo usiamo qui quindi non è necessario che Python 3.12 sia installato sul computer locale per far funzionare la build.
4. Per creare le risorse Lambda e EventBridge Scheduler specificate nel `template.yaml` file, esegui il comando seguente.

   ```
   sam deploy --guided
   ```

   L'uso del `--guided` flag significa che ti AWS SAM verranno mostrate delle istruzioni per guidarti nel processo di distribuzione. Per questa implementazione, inserisci un `Stack name` pari a **`cron-maintenance-app`** e accetta i valori predefiniti per tutte le altre opzioni utilizzando Invio.

   Al AWS SAM termine della creazione delle risorse Lambda e EventBridge Scheduler, dovrebbe apparire il seguente messaggio.

   ```
   Successfully created/updated stack - cron-maintenance-app in us-west-2
   ```
5. Puoi inoltre confermare che la funzione Lambda è stata creata aprendo la pagina Funzioni della console Lambda. Dovresti vedere una funzione denominata. `ScheduledDBMaintenance`

Test dell'app

Per verificare che la pianificazione attivi correttamente la funzione e che la funzione pulisca correttamente i record dal database, è possibile modificare temporaneamente la pianificazione in modo che venga eseguita una volta in un momento specifico. È quindi possibile eseguire `sam`

`deploy` nuovamente l'operazione per reimpostare il programma di ricorrenza in modo che venga eseguito una volta al mese.

Per eseguire l'applicazione utilizzando il Console di gestione AWS

1. Torna alla pagina della console EventBridge Scheduler.
2. Scegli la tua pianificazione, quindi scegli Modifica.
3. Nella sezione Schema di pianificazione, in Ricorrenza, scegli Pianificazione unica.
4. Imposta il tempo di chiamata a pochi minuti da oggi, rivedi le impostazioni, quindi scegli Salva.

Dopo l'esecuzione della pianificazione e il richiamo della destinazione, esegui `test_app.py` lo script per verificare che la funzione abbia rimosso correttamente tutti i vecchi record dalla tabella DynamoDB.

Per verificare che i vecchi record vengano eliminati utilizzando uno script Python

1. Nella riga di comando, accedi alla cartella in cui hai salvato`test_app.py`.
2. Eseguire lo script.

   ```
   python test_app.py
   ```

   Se il test viene superato, viene visualizzato il seguente output.

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

Fasi successive

Ora puoi modificare la EventBridge pianificazione dello Scheduler per soddisfare i requisiti specifici dell'applicazione. EventBridge Scheduler supporta le seguenti espressioni di pianificazione: cron, rate e one-time schedules.

Per ulteriori informazioni sulle espressioni di EventBridge pianificazione di Scheduler, consulta Scheduler types nella Scheduler User GuideEventBridge . Gestione degli accessi nella Guida per l'utente IAM

Creazione di un sistema di elaborazione degli ordini con Lambda Durable Functions

> **Note**
>
> ESIGENZA: Aggiungi un diagramma di architettura che mostri API Gateway, il flusso di lavoro Durable Function e i servizi di supporto (DynamoDB,) EventBridge

Prerequisiti

- AWS CLI installato e configurato
- ESIGENZA: Requisiti specifici per le funzioni durevoli

Creare i file del codice sorgente

Crea i seguenti file nella directory del tuo progetto:

- `lambda_function.py`- il codice della funzione
- `requirements.txt`- manifesto delle dipendenze

Codice della funzione

```
# NEED: Verify correct imports
import boto3
import json

def lambda_handler(event, context):
    # NEED: Verify DurableContext syntax
    durable = context.durable

    try:
        # Validate and store order
        order = await durable.step('validate', async () => {
            return validate_order(event['order'])
        })

        # Process payment
```

```
        # NEED: Verify wait syntax
        await durable.wait(/* wait configuration */)

        # Additional steps
        # NEED: Complete implementation

    except Exception as e:
        # NEED: Error handling patterns
        raise e

def validate_order(order_data):
    # NEED: Implementation
    pass
```

File di requisiti

```
# NEED: List of required packages
```

Distribuire l'app

Creare una tabella DynamoDB per gli ordini

1. Apri la console DynamoDB all'indirizzo https://console.aws.amazon.com/dynamodb/
2. Scegli Crea tabella
3. Per Nome tabella, inserisci **`Orders`**
4. Per la chiave di partizione, immettere **`orderId`**
5. Lascia le altre impostazioni come predefinite
6. Scegli Crea tabella

Creazione della funzione Lambda

1. Apri la console Lambda all'indirizzo https://console.aws.amazon.com/lambda/
2. Selezionare Create function (Crea funzione).
3. Seleziona l'autore da zero
4. Nel campo Nome funzione, inserisci **`ProcessOrder`**.
5. Per Runtime, scegli il tuo runtime preferito
6. NECESSITÀ: Aggiungi una configurazione specifica per Durable Functions

7. Selezionare Create function (Crea funzione).

Crea l'endpoint API Gateway

1. Apri la console API Gateway all'indirizzo https://console.aws.amazon.com/apigateway/
2. Scegli Crea API
3. Seleziona HTTP API
4. Scegli Build
5. Aggiungi un'integrazione con la tua funzione Lambda
6. Configura i percorsi per l'elaborazione degli ordini
7. Distribuzione dell'API

Eseguire il test dell'app

Invia un ordine di prova:

```
{
    "orderId": "12345",
    "items": [
        {
            "productId": "ABC123",
            "quantity": 1
        }
    ]
}
```

NECESSITÀ: Aggiungi istruzioni di monitoraggio specifiche per Durable Functions

Fasi successive

Aggiungi business Logic

Implementa la gestione dell'inventario:

```
async def check_inventory(order):
    # Add inventory check logic
    pass
```

Aggiungi calcoli dei prezzi:

```
async def calculate_total(order):
    # Add pricing logic
    pass
```

Migliora la gestione degli errori

Aggiungi logica di compensazione:

```
async def reverse_payment(order):
    # Add payment reversal logic
    pass
```

Gestisci gli annullamenti degli ordini:

```
async def cancel_order(order):
    # Add cancellation logic
    pass
```

Integrazione di sistemi esterni

```
async def notify_shipping_provider(order):
    # Add shipping integration
    pass

async def send_customer_notification(order):
    # Add notification logic
    pass
```

Migliora il monitoraggio

- Crea CloudWatch dashboard
- Imposta le metriche per i tempi di elaborazione degli ordini
- Configura gli avvisi per gli ordini in ritardo

Strumenti di sviluppo per Lambda

Hai accesso a una varietà di strumenti che aumentano la produttività e la facilità d'uso durante l'intero ciclo di vita dello sviluppo. Questa sezione fornisce informazioni sugli strumenti che aiutano molti clienti Lambda a progettare, sviluppare e gestire le proprie applicazioni. Dallo sviluppo locale nell'IDE alla distribuzione e gestione di applicazioni serverless complesse, questi strumenti consentono di semplificare il flusso di lavoro, migliorare la qualità del codice e accelerare lo sviluppo di solide soluzioni basate su Lambda.

- Sviluppo locale: scrivi e testa le funzioni Lambda più velocemente nel tuo ambiente di sviluppo preferito. Il AWS Toolkit for VS Code consente lo sviluppo, il debug e il test di funzioni locali con funzionalità di distribuzione diretta su Lambda.
- Infrastructure as Code (IaC): distribuisci e gestisci applicazioni serverless in modo coerente, dai test locali alla produzione. AWS SAMAWS CDK, e CloudFormation consente di definire e gestire l'infrastruttura serverless tramite codice per implementazioni coerenti e controllate dalla versione.
- GitHub Actions: automatizza le distribuzioni Lambda direttamente dal tuo repository di codice. GitHub Actions ti consente di configurare flussi di lavoro che distribuiscono automaticamente le funzioni Lambda ogni volta che esegui modifiche al codice o alla configurazione, semplificando la pipeline CI/CD.
- Powertools for AWS Lambda: crea applicazioni serverless pronte per la produzione con meno codice personalizzato. Powertools for AWS Lambda (noto anche come Powertools AWS for) è un toolkit per sviluppatori open source che semplifica l'implementazione di best practice serverless come osservabilità, recupero dei parametri e idempotenza su Python, TypeScript, Java e.NET.
- Flussi di lavoro ed eventi: coordina le funzioni Lambda con AWS servizi, API e sistemi esterni. AWS Step Functionse Amazon EventBridge forniscono funzionalità di orchestrazione e gestione degli eventi per coordinare flussi di lavoro applicativi complessi basati su Lambda.

Strumenti di sviluppo locali

Gli ambienti di sviluppo locali consentono di lavorare offline e sfruttare le funzionalità IDE avanzate mentre iterate rapidamente sulle funzioni Lambda. Questi strumenti consentono di eseguire il debug di funzioni complesse e di svilupparle in ambienti con connettività limitata. Supportano anche la collaborazione in team e l'integrazione con i sistemi di controllo delle versioni.

Per ulteriori informazioni sulle funzioni Lambda, consulta la Guida per gli sviluppatori di . Questa pagina descrive come spostare lo sviluppo di funzioni Lambda dalla AWS console a Visual Studio Code, che fornisce un ambiente di sviluppo avanzato con funzionalità come il debug e il completamento del codice. Per effettuare la transizione, è necessario configurare le credenziali AWS Toolkit for Visual Studio Code and, dopodiché è possibile utilizzare le funzionalità avanzate di VS Code mantenendo la possibilità di eseguire la distribuzione direttamente su. AWS

Lo sviluppo locale per Lambda offre diverse funzionalità chiave:

- Usa l'integrazione di Visual Studio Code con la console Lambda
- Configurazione degli ambienti di sviluppo Lambda locali
- Esegui il debug e il test delle funzioni localmente
- Applica le migliori pratiche per la gestione delle funzioni locali

Per ulteriori informazioni, consulta Sviluppo locale di funzioni Lambda con VS Code.

Infrastructure as code (IaC)

Con gli strumenti Infrastructure as Code (IaC), puoi definire e gestire la tua architettura serverless utilizzando il codice. Questo approccio aiuta a mantenere la coerenza tra gli ambienti, consente di controllare le versioni dell'infrastruttura e facilita le pratiche DevOps. IaC è particolarmente utile per automatizzare le implementazioni, garantire ambienti coerenti e gestire implementazioni multiregionali.

Gli strumenti e i concetti chiave di IaC per Lambda includono framework per la creazione di modelli, la gestione dell'implementazione e le migliori pratiche per l'infrastruttura serverless:

- Principi IaC di base per lo sviluppo di Lambda
- CloudFormatione AWS SAM funzionalità AWS CDK
- Criteri di selezione e confronto degli utensili
- Best practice per l'implementazione di Lambda di IaC

Che tu stia lavorando in modo indipendente su un piccolo progetto o come parte di un grande team che gestisce applicazioni serverless su scala aziendale, questi strumenti di sviluppo e distribuzione possono aiutarti a scrivere, implementare e gestire le tue funzioni Lambda in modo più efficace.

Per ulteriori informazioni, consulta Utilizzo di Lambda con l'infrastructure as code (IaC).

Strumenti GitHub Actions

GitHub Actions offre funzionalità di distribuzione automatizzata per le funzioni Lambda direttamente dal tuo repository di codice. Creando file di workflow nel tuo repository, puoi implementare automaticamente le funzioni Lambda ogni volta che vengono apportate modifiche di codice o di configurazione, semplificando la pipeline per l'integrazione continua e la distribuzione continua (CI/CD). L'azione Deploy Lambda Function offre un'interfaccia YAML dichiarativa che semplifica il processo di distribuzione, gestisce le credenziali AWS tramite OpenID Connect (OIDC) e supporta vari scenari di distribuzione tra cui aggiornamenti del codice, modifiche alla configurazione e convalide a secco. Questa integrazione consente ai team di mantenere un processo di distribuzione coerente e automatizzato sfruttando al contempo i flussi di lavoro GitHub esistenti.

Per ulteriori informazioni, consulta Utilizzo di GitHub Actions per implementare le funzioni Lambda.

Utensili elettrici per AWS Lambda

Powertools for AWS è un toolkit per sviluppatori open source che consente di implementare le migliori pratiche serverless con un codice personalizzato minimo. Disponibile per Python, TypeScript/Node.js, Java e.NET, fornisce funzioni di utilità, decoratori e middleware che semplificano le attività di sviluppo Lambda più comuni. Il toolkit include funzionalità di osservabilità integrate come la registrazione strutturata, il tracciamento e la raccolta di metriche, come utilità per il recupero dei parametri, la gestione dei segreti e i modelli di idempotenza. Questi strumenti si allineano a best practice ben strutturate e aiutano gli sviluppatori a creare applicazioni serverless pronte per la produzione in modo più AWS efficiente. Riducendo il codice standard e standardizzando i modelli comuni, Powertools for AWS consente ai team di concentrarsi sulla logica di business mantenendo al contempo l'implementazione coerente delle migliori pratiche serverless nelle loro applicazioni.

Per ulteriori informazioni, consulta Utensili elettrici per AWS Lambda.

Strumenti di gestione del flusso di lavoro e degli eventi

Le applicazioni Lambda possono essere utilizzate per l'orchestrazione di flussi di lavoro complessi e la gestione di vari eventi. AWSfornisce strumenti specializzati per aiutarti a gestire questi aspetti dello sviluppo serverless. Scopri AWS Step Functions l'orchestrazione del flusso di lavoro e Amazon

EventBridge per la gestione degli eventi e come integrarli con le tue funzioni Lambda. Questi strumenti possono migliorare in modo significativo la scalabilità e l'affidabilità delle applicazioni serverless fornendo una solida gestione dello stato e architetture basate sugli eventi. Sfruttando questi servizi, è possibile creare soluzioni basate su Lambda più sofisticate e resilienti in grado di gestire processi aziendali complessi e reagire a un'ampia gamma di eventi di sistema e applicazioni.

Per ulteriori informazioni, consulta Gestione dei flussi di lavoro e degli eventi Lambda.

Sviluppo locale di funzioni Lambda con VS Code

Puoi spostare le tue funzioni Lambda dalla console Lambda a Visual Studio Code, che fornisce un ambiente di sviluppo completo e ti consente di utilizzare altre opzioni di sviluppo locali come AWS SAM e AWS CDK.

Vantaggi principali dello sviluppo locale

Sebbene la console Lambda offra un modo rapido per modificare e testare le funzioni, lo sviluppo locale offre funzionalità più avanzate:

- Funzionalità IDE avanzate: strumenti di debug, completamento del codice e refattorizzazione
- Sviluppo offline: elabora e testa le modifiche localmente prima dell'implementazione nel cloud
- Integrazione dell'infrastruttura come codice: utilizzo senza interruzioni con AWS SAM, AWS CDK e Infrastructure Composer
- Gestione delle dipendenze: controllo completo sulle dipendenze delle funzioni

Prerequisiti

Prima di sviluppare funzioni Lambda localmente in VS Code, è necessario:

- VS Code: per le istruzioni di installazione, consulta Scarica VS Code.
- AWS Toolkit for Visual Studio Code: per le istruzioni sull'installazione, consulta Configurazione di AWS Toolkit for Visual Studio Code. Per una panoramica, consulta AWS Toolkit for Visual Studio Code.
- Credenziali di AWS: per informazioni sulla configurazione delle credenziali, vedi Configurazione delle credenziali di AWS.
- AWS SAM CLI: per istruzioni sull'installazione, consulta Installazione di AWS SAM CLI.

- Docker installato (facoltativo, ma richiesto per i test locali): per le istruzioni di installazione, vedi Ottieni Docker.

> **Note**
>
> Se disponi già di un account AWS e di un profilo configurati localmente, assicurati che la policy gestita di AdministratorAccess sia stata aggiunta al tuo profilo AWS configurato.

Autenticazione e controllo degli accessi

Per sviluppare le funzioni Lambda localmente, devi avere credenziali AWS per accedere e gestire in modo sicuro le risorse AWS per tuo conto, proprio come farebbero nel cloud. Il Toolkit AWS per VS Code supporta i seguenti metodi di autenticazione:

Il Toolkit AWS per VS Code supporta i seguenti metodi di autenticazione:

- Credenziali a lungo termine per utenti IAM
- Credenziali temporanee provenienti dai ruoli presunti
- Federazione delle identità
- Credenziali dell'utente root dell'account AWS (non consigliato)

Questa sezione funge da guida per ottenere e configurare queste credenziali utilizzando le credenziali a lungo termine per utenti IAM.

Credenziali IAM AWS

Se hai già un utente IAM con chiavi di accesso, tieni a portata di mano l'ID della chiave di accesso e la chiave di accesso segreta. Se non disponi di tali chiavi, creale eseguendo i passaggi descritti di seguito:

> **Note**
>
> È necessario utilizzare sia l'ID chiave di accesso sia la chiave di accesso segreta insieme per autenticare le richieste dell'utente.

Per creare una chiave di accesso per un utente IAM:

1. Apri la console IAM all'indirizzo https://console.aws.amazon.com/iam/.
2. Nel pannello di navigazione, seleziona Utenti.
3. Selezionare Create user (Crea utente).
4. In Nome utente, inserisci un nome e scegli Successivo.
5. In Imposta autorizzazioni, seleziona Allega policy direttamente.
6. Seleziona AdministratorAccess e scegli Successivo.
7. Selezionare Create user (Crea utente).
8. Nel banner che indica la riuscita dell'operazione, scegli Visualizza utente.
9. Selezionare Create access key (Crea chiave di accesso).
10. Per Caso d'uso, seleziona Codice locale.
11. Seleziona la casella di controllo di conferma e scegli Successivo.
12. (Opzionale) Inserisci un valore di tag descrittivo.
13. Selezionare Create access key (Crea chiave di accesso).
14. Copia immediatamente la chiave di accesso e la chiave di accesso segreta. Dopo aver lasciato questa pagina, non potrai accedere nuovamente alla chiave di accesso segreta.

> **Important**
>
> Non condividere mai la chiave segreta e non eseguirne il commit nel controllo del codice sorgente. Archivia queste chiavi in modo sicuro ed eliminale quando non sono più necessarie.

> **Note**
>
> Per ulteriori informazioni, consulta Creazione di un utente IAM nel tuo account AWS e Gestione delle chiavi di accesso per utenti IAM nella Guida per l'utente di IAM.

Configura le credenziali AWS utilizzando il Toolkit AWS

La tabella seguente riassume il processo di configurazione delle credenziali che completerai nella procedura seguente.

Cosa fare	Perché?
Apri il pannello di accesso	Avvio dell'autenticazione
Usa la tavolozza dei comandi per cercare AWS Aggiungi una nuova connessione	Accedi all'interfaccia di accesso
Scegli credenziali IAM	Usa le chiavi di accesso per l'accesso programmatico
Inserisci il nome del profilo, la chiave di accesso, la chiave segreta	Fornisci le credenziali per la connessione
Vedi l'aggiornamento dell'Esploratore AWS	Conferma l'avvenuta connessione

Completa i passaggi seguenti per autenticarti nel tuo account AWS:

1. Apri il pannello di accesso in VS Code:
   a. Per avviare il processo di autenticazione, seleziona l'icona AWS nel riquadro di navigazione a sinistra oppure apri la tavolozza dei comandi (Cmd+Maiusc+P su Mac o Ctrl+Maiusc+P su Windows/Linux) e cerca e seleziona AWS Aggiungi una nuova connessione.
2. Nel pannello di accesso, scegli Credenziali IAM e seleziona Continua.

> **Note**
>
> Per procedere, dovrai consentire alle estensioni IDE AWS per VS Code di accedere ai tuoi dati.

3. Inserisci il nome del tuo profilo, l'ID della chiave di accesso e la chiave di accesso segreta, quindi seleziona Continua.
4. Verifica la connessione controllando Esploratore AWS in VS Code per i tuoi servizi e risorse AWS.

Per informazioni sulla configurazione dell'autenticazione con credenziali a lungo termine, consulta Utilizzo di credenziali a lungo termine per autenticare SDK e strumenti AWS.

Per informazioni sulla configurazione dell'autenticazione, consulta Credenziali IAM AWS nella Guida per l'utente di AWS Toolkit for Visual Studio Code.

Passaggio da console a sviluppo locale

> **Note**
>
> Se hai apportato modifiche alla console, assicurati che tutte le modifiche siano state implementate prima di passare allo sviluppo locale.

Per spostare una funzione Lambda dalla console Lambda a VS Code, completa i seguenti passaggi:

1. Aprire la console Lambda.
2. Scegli il nome della funzione .
3. Seleziona la scheda Fonte codice.
4. Scegli Apri in Visual Studio Code.

   > **Note**
   >
   > Il pulsante Apri in Visual Studio Code è disponibile solo nel Toolkit AWS versione 3.69.0 e successive. Se è installata una versione precedente del Toolkit AWS, è possibile che venga visualizzato un messaggio `Cannot open the handler` in VS Code. Per risolvere questo problema, aggiorna il tuo Toolkit AWS alla versione più recente.

5. Quando richiesto, consenti al browser di aprire VS Code.

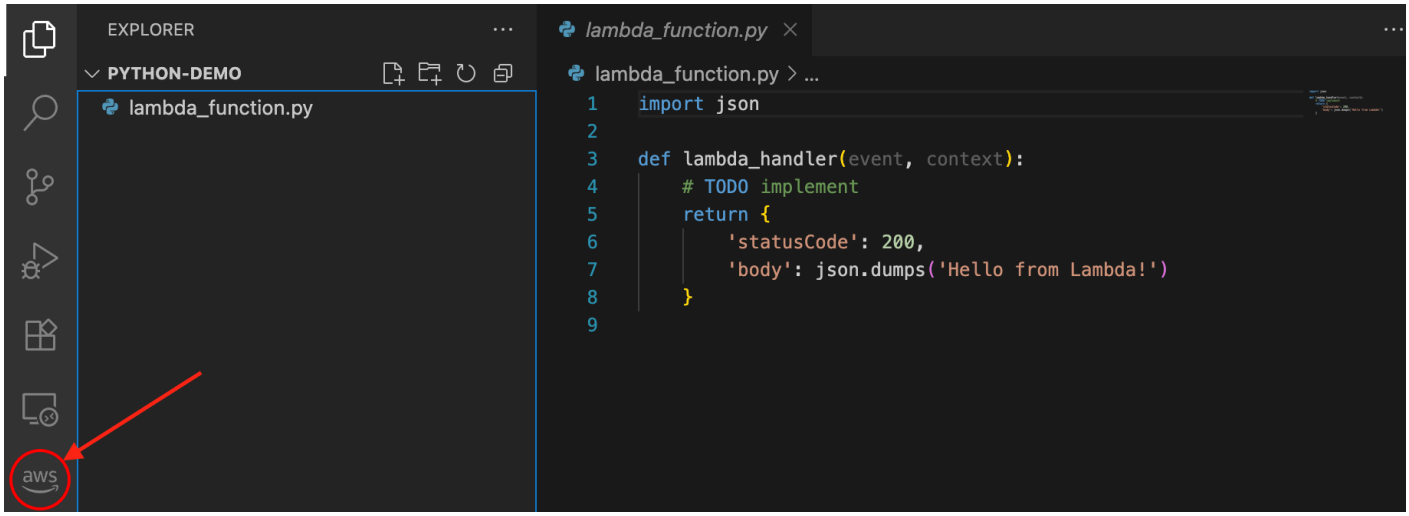
Quando apri la funzione in VS Code, Lambda crea un progetto locale con il codice della funzione in una posizione temporanea progettata per test e implementazione rapidi. Ciò include il codice della funzione, le dipendenze e una struttura di progetto di base che puoi utilizzare per lo sviluppo locale.

Per i dettagli sull'utilizzo di AWS in VS Code, consulta la Guida per l'utente di AWS Toolkit for Visual Studio Code.

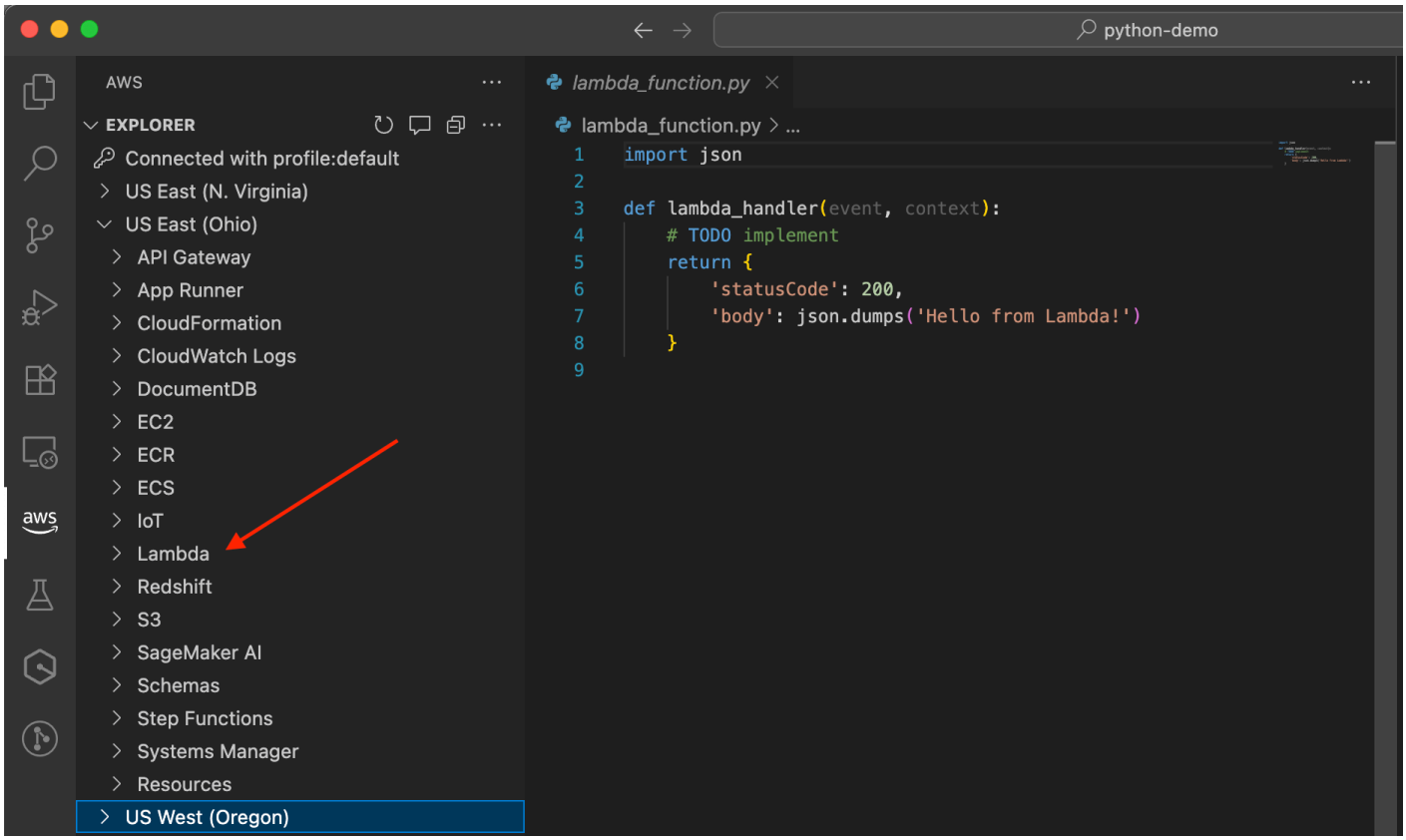
Utilizzo delle funzioni a livello locale

Dopo aver aperto la funzione in VS Code, segui questi passaggi per accedere e gestire le tue funzioni:

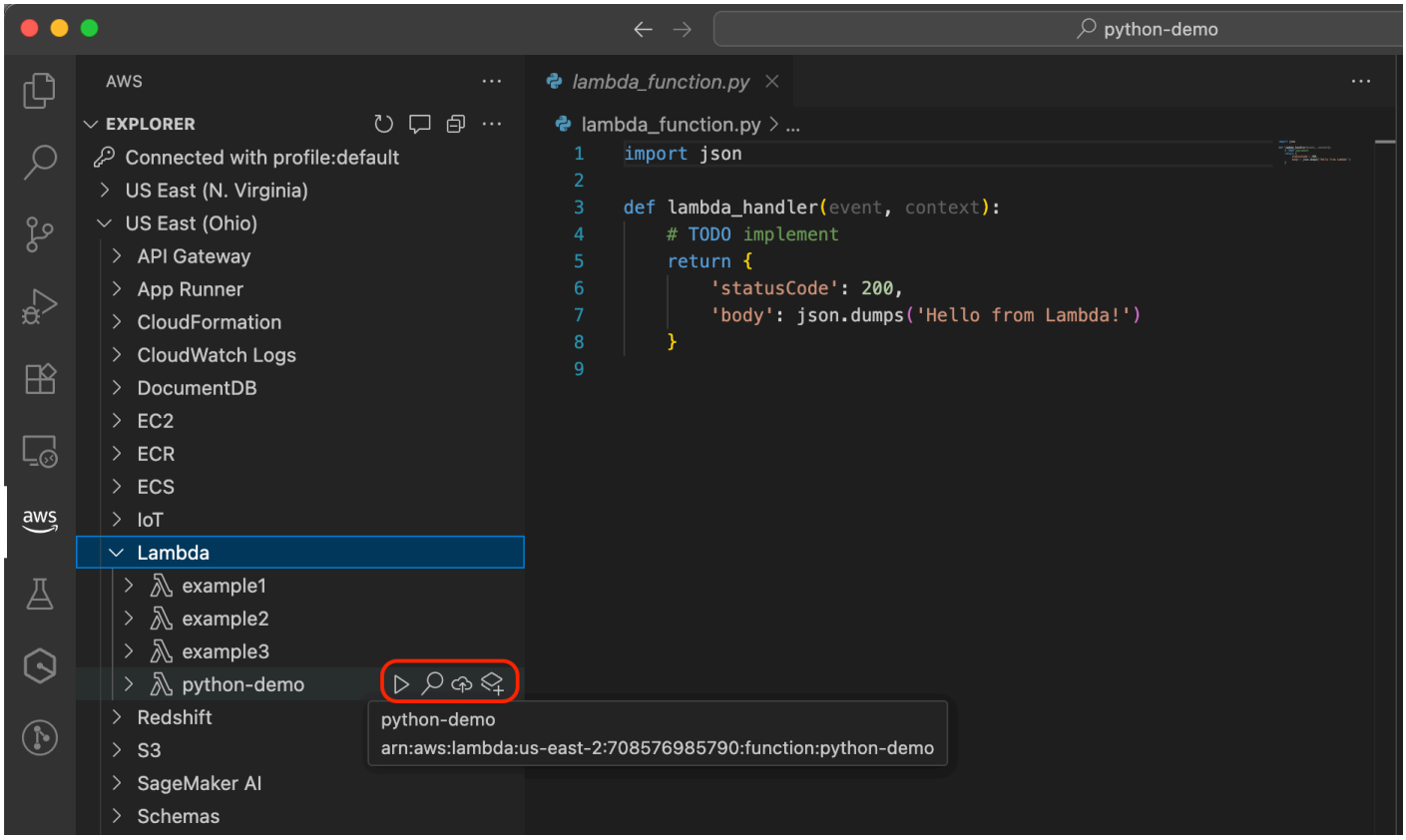
1. Seleziona l'icona AWS nella barra laterale per aprire l’Esploratore AWS:



2. Nell'Esploratore AWS, seleziona la regione con la tua funzione Lambda:



3. Nella regione selezionata, espandi la sezione Lambda per visualizzare e gestire le tue funzioni:



Con la funzione aperta in VS Code, puoi:

- Modificare il codice della funzione con supporto linguistico completo e completamento del codice.
- Utilizzare l'integrazione LocalStack in VS Code per testare le funzioni Lambda che effettuano chiamate API ad altri Servizi AWS durante l'esecuzione, come la lettura da tabelle DynamoDB o la scrittura su bucket Amazon S3. LocalStack è un emulatore di servizi cloud che fornisce un ambiente di sviluppo locale completo per testare le integrazioni di servizi. Puoi anche usare AWS SAM CLI per testare la tua funzione in un container locale. Se la tua funzione effettua chiamate API ad altri Servizi AWS, tali chiamate raggiungeranno risorse AWS reali, non quelle emulate.
- Esegui il debug della tua funzione con punti di interruzione e ispezione delle variabili. Per ulteriori informazioni, consulta Esecuzione e debug delle funzioni Lambda direttamente dal codice nella Guida per l'utente di AWS Toolkit for Visual Studio Code.
- Implementa di nuovo in AWS la funzione aggiornata utilizzando l'icona a forma di nuvola.
- Installa e gestisci le dipendenze per la tua funzione.

Per ulteriori informazioni, consulta Utilizzo delle funzioni Lambda AWS nella Guida per l'utente di AWS Toolkit for Visual Studio Code.

Converti la tua funzione in un modello AWS SAM e usa gli strumenti IaC

In VS Code, puoi convertire la funzione Lambda in un modello AWS SAM scegliendo l'icona Converti in applicazione AWS SAM accanto alla funzione Lambda. Ti verrà richiesto di selezionare la sede del progetto AWS SAM. Una volta selezionata, la funzione Lambda verrà convertita in un file `template.yaml` salvato nel nuovo progetto AWS SAM.

Con la funzione convertita in un modello AWS SAM, puoi:

- Eseguire il controllo delle versioni della tua infrastruttura
- Automatizzare le implementazioni
- Eseguire il debug delle funzioni da remoto
- Aggiungere risorse AWS alla tua applicazione
- Mantenere ambienti coerenti per tutto il ciclo di vita dello sviluppo
- Usare Infrastructure Composer per modificare visivamente il modello AWS SAM

Per ulteriori informazioni sull'utilizzo degli strumenti IaC, consulta le seguenti guide:

- La Guida per gli sviluppatori AWS Serverless Application Model
- La Guida per gli sviluppatori AWS Cloud Development Kit (AWS CDK)
- La guida per sviluppatori di Infrastructure Composer
- La Guida per l'utente di AWS CloudFormation

Questi strumenti forniscono funzionalità aggiuntive per la definizione, il test e l'implementazione delle applicazioni serverless.

Passaggi successivi

Per ulteriori informazioni sull'utilizzo delle funzioni Lambda in VS Code, consulta le seguenti risorse:

- Utilizzo delle funzioni Lambda AWS nella Guida per l'utente di Toolkit AWS per VS Code
- Utilizzo delle applicazioni serverless nella Guida per l'utente di Toolkit AWS per VS Code
- Infrastruttura come codice nella Guida per sviluppatori Lambda

Utilizzo di GitHub Actions per implementare le funzioni Lambda

Puoi usare GitHub Actions per implementare automaticamente le funzioni Lambda quando invii modifiche al codice o alla configurazione nel tuo repository. L'azione Implementa funzione Lambda fornisce un'interfaccia YAML dichiarativa e semplice che elimina la complessità delle fasi di implementazione manuale.

Flusso di lavoro di esempio

Per configurare l'implementazione automatizzata della funzione Lambda, crea un file di workflow nella directory `.github/workflows/` del repository:

Example Flusso di lavoro Github Actions per l'implementazione 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
```

Questo flusso di lavoro viene eseguito quando invii le modifiche al ramo `main`. Controlla il tuo repository, configura le credenziali AWS utilizzando OpenID Connect (OIDC) e implementa la tua funzione utilizzando il codice nella directory `./dist`.

Per altri esempi, tra cui l'aggiornamento della configurazione delle funzioni, l'implementazione tramite bucket S3 e la convalida del test di analisi, consulta il README di implementazione della funzione Lambda.

Risorse aggiuntive

- Configurazione dell'azione GitHub delle credenziali AWS
- Configurazione di OpenID Connect in AWS

Utilizzo di Lambda con l'infrastructure as code (IaC)

Le funzioni Lambda raramente vengono eseguite in modo isolato. Spesso fanno invece parte di un'applicazione serverless con altre risorse come database, code e spazio di archiviazione. Con l'infrastructure as code (IaC), è possibile automatizzare i processi di implementazione per implementare e aggiornare in modo rapido e ripetibile intere applicazioni serverless che coinvolgono molte risorse AWS separate. Questo approccio accelera il ciclo di sviluppo, semplifica la gestione della configurazione e garantisce che le risorse vengano implementate sempre allo stesso modo.

Strumenti di IaC per Lambda

CloudFormation

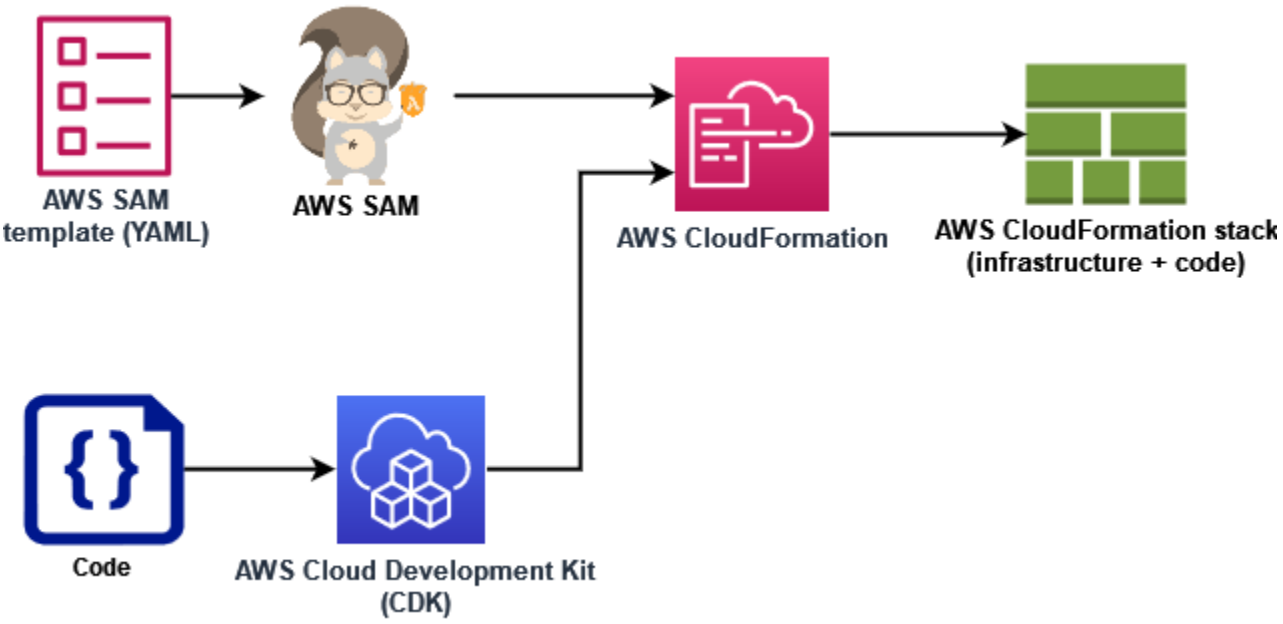
> CloudFormation è il servizio IaC fondamentale di AWS. Puoi utilizzare i modelli YAML o JSON per modellare ed eseguire il provisioning dell'intera infrastruttura AWS, comprese le funzioni Lambda. CloudFormation gestisce le complessità legate alla creazione, all'aggiornamento e all'eliminazione delle risorse AWS.

AWS Serverless Application Model (AWS SAM)

> AWS SAM è un framework open source basato su CloudFormation. Fornisce una sintassi semplificata per definire applicazioni serverless. Usa i modelli AWS SAM per effettuare rapidamente il provisioning di funzioni Lambda, API, database e origini eventi con poche righe di YAML.

AWS Cloud Development Kit (AWS CDK)

Il CDK è un approccio basato sul codice a IaC. Puoi definire la tua architettura basata su Lambda utilizzando TypeScript, JavaScript, Python, Java, C#/.NET o Go. Scegli il tuo linguaggio preferito e utilizza elementi di programmazione come parametri, condizionali, cicli, composizione ed ereditarietà per definire il risultato desiderato della tua infrastruttura. Il CDK genera quindi i modelli CloudFormation sottostanti per l'implementazione. Per un esempio di come utilizzare Lambda con CDK, consulta Implementazione delle funzioni Lambda con AWS CDK.



Inoltre, AWS fornisce un servizio chiamato AWS Infrastructure Composer per sviluppare modelli di IaC utilizzando una semplice interfaccia grafica. Con Infrastructure Composer, è possibile progettare un'architettura di applicazioni trascinando, raggruppando e connettendo i Servizi AWS in un canvas visivo. Infrastructure Composer crea quindi un modello AWS SAM oppure un modello CloudFormation a partire dal tuo progetto che puoi utilizzare per implementare l'applicazione.

Nella sezione the section called “Utilizzo di AWS SAM e Infrastructure Composer” seguente, Infrastructure Composer viene utilizzato per sviluppare un modello per un'applicazione serverless basata su una funzione Lambda esistente.

Utilizzo delle funzioni Lambda in AWS SAM e Infrastructure Composer

In questo tutorial, puoi iniziare a utilizzare l'IaC con Lambda creando un modello AWS SAM da una funzione Lambda esistente e successivamente creando un'applicazione serverless in Infrastructure Composer mediante l'aggiunta di altre risorse AWS.

Man mano che procedi in questo tutorial, imparerai alcuni concetti fondamentali, ad esempio come vengono specificate le risorse AWS in AWS SAM. Imparerai anche a utilizzare Infrastructure Composer per creare un'applicazione serverless che puoi implementare tramite AWS SAM oppure CloudFormation.

Per completare questo tutorial, eseguirai le seguenti attività:

- Creazione di una funzione Lambda di esempio
- Utilizzo della console Lambda per visualizzare il modello AWS SAM della funzione
- Esportazione della configurazione della funzione in AWS Infrastructure Composer e progettazione di una semplice applicazione serverless basata sulla configurazione della funzione
- Salvataggio di un modello AWS SAM aggiornato che può essere utilizzato come base per implementare un'applicazione serverless

Prerequisiti

In questo tutorial, utilizzi la funzionalità di sincronizzazione locale di Infrastructure Composer per salvare i file di modello e codice sul computer di compilazione locale. Per utilizzare questa funzionalità, è necessario un browser che supporti l'API File System Access, che consente alle applicazioni web di leggere, scrivere e salvare file nel file system locale. Ti consigliamo di utilizzare Google Chrome o Microsoft Edge. Per ulteriori informazioni sull'API File System Access, consulta la pagina What is the File System Access API?

Creazione di una funzione Lambda

In questo primo passaggio, creerai una funzione Lambda da utilizzare nei passaggi successivi del tutorial. Per semplicità, utilizzerai la console Lambda per creare una funzione di base "Hello world" tramite il runtime Python 3.11.

Creazione di una funzione Lambda "Hello world" tramite la console

1. Aprire la console Lambda.

2. Scegli Crea funzione.
3. Lascia selezionato Crea da zero e, in Informazioni di base, immetti **`LambdaIaCDemo`** come Nome della funzione.
4. In Runtime, scegli Python 3.11.
5. Scegli Crea funzione.

Visualizzazione del modello AWS SAM per la funzione

Prima di esportare la configurazione della funzione in Infrastructure Composer, utilizza la console Lambda per visualizzare la configurazione corrente della funzione come modello AWS SAM. Seguendo i passaggi di questa sezione, imparerai a conoscere la struttura di un modello AWS SAM e a definire risorse come le funzioni Lambda per iniziare a specificare un'applicazione serverless.

Visualizzazione del modello AWS SAM per la funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la funzione creata in precedenza (`LambdaIaCDemo`).
3. Nel riquadro Panoramica della funzione, scegli Modello.

   Al posto del diagramma che rappresenta la configurazione della funzione, vedrai un modello AWS SAM relativo alla funzione. Il modello avrà un aspetto simile al seguente.

```
# 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:*
```

Esaminiamo il modello YAML per la funzione in uso e apprendiamo alcuni concetti chiave.

Il modello inizia con la dichiarazione `Transform: AWS::Serverless-2016-10-31`. Questa dichiarazione è obbligatoria perché, dietro le quinte, i modelli AWS SAM vengono implementati tramite CloudFormation. L'utilizzo dell'istruzione `Transform` identifica il modello come file di modello AWS SAM.

Dopo la dichiarazione `Transform`, arriva la sezione `Resources`. Qui definisci le risorse AWS che desideri implementare con il modello AWS SAM. I modelli AWS SAM possono contenere una combinazione di risorse AWS SAM e risorse CloudFormation. Questo perché, durante l'implementazione, i modelli AWS SAM si espandono in modelli CloudFormation, quindi a un modello AWS SAM è possibile aggiungere qualsiasi sintassi CloudFormation valida.

Al momento, nella sezione `Resources` del modello è presente soltanto una risorsa definita, la funzione Lambda `LambdaIaCDemo`. Per aggiungere una funzione Lambda a un modello AWS SAM, utilizzi il tipo di risorsa `AWS::Serverless::Function`. La risorsa `Properties` di una funzione Lambda definisce il runtime della funzione, il gestore delle funzioni e altre opzioni di configurazione. Qui viene definito anche il percorso del codice sorgente della funzione che AWS SAM deve utilizzare per implementare la funzione. Per ulteriori informazioni sulle risorse della funzione Lambda in AWS SAM, consulta la sezione AWS::Serverless::Function nella Guida per gli sviluppatori di AWS SAM.

Oltre alle proprietà e alle configurazioni della funzione, il modello specifica anche una policy AWS Identity and Access Management (IAM) per la funzione. Questa policy autorizza la funzione a scrivere i log in Amazon CloudWatch Logs. Quando crei una funzione nella console Lambda, Lambda associa automaticamente questa policy alla tua funzione. Per ulteriori informazioni sulla definizione di una policy IAM per una funzione in un modello AWS SAM, consulta la proprietà `policies` nella pagina AWS::Serverless::Function della Guida per gli sviluppatori di AWS SAM.

Per ulteriori informazioni sulla struttura dei modelli AWS SAM, consulta la pagina AWS SAM template anatomy.

Utilizzo di AWS Infrastructure Composer per progettare un'applicazione serverless

Per iniziare a creare una semplice applicazione serverless utilizzando il modello AWS SAM della funzione come punto di partenza, esporta la configurazione della funzione in Infrastructure Composer e attiva la modalità di sincronizzazione locale di Infrastructure Composer. La sincronizzazione locale salva automaticamente il codice della funzione e il modello AWS SAM sul computer di compilazione locale e mantiene sincronizzato il modello salvato man mano che si aggiungono altre risorse AWS a Infrastructure Composer.

Per esportare una funzione Lambda in Infrastructure Composer

1. Nel riquadro Panoramica della funzione, scegli Esporta in Strumento per la creazione di applicazioni.

   Per esportare la configurazione e il codice della funzione in Infrastructure Composer, Lambda crea un bucket Amazon S3 nel tuo account in cui archiviare temporaneamente questi dati.
2. Nella finestra di dialogo, scegli Conferma e crea progetto per accettare il nome predefinito per questo bucket ed esportare la configurazione e il codice della funzione in Infrastructure Composer.

3. (Facoltativo) Per scegliere un altro nome per il bucket Amazon S3 creato da Lambda, immetti un nuovo nome e scegli Conferma e crea progetto. I nomi dei bucket Amazon S3 devono essere univoci a livello globale e seguire le regole di denominazione dei bucket.

   Selezionando Conferma e crea progetto si apre la console Infrastructure Composer. Nel canvas vedrai la funzione Lambda.
4. Dal menu a discesa Menu, scegli Attiva sincronizzazione locale.
5. Nella finestra di dialogo che si apre, scegli Seleziona cartella e seleziona una cartella sul tuo computer di compilazione locale.
6. Scegli Attiva per attivare la sincronizzazione locale.

Per esportare la funzione in Infrastructure Composer, è necessaria l'autorizzazione a utilizzare determinate operazioni API. Se non sei in grado di esportare la funzione, consulta the section called "Autorizzazioni richieste" e assicurati di disporre delle autorizzazioni necessarie.

> **Note**
>
> Il bucket che Lambda crea quando esporti una funzione in Infrastructure Composer è soggetto ai prezzi di Amazon S3 standard. Gli oggetti che Lambda inserisce nel bucket vengono eliminati automaticamente dopo 10 giorni, ma il bucket in sé non viene eliminato. Per evitare l'addebito di costi aggiuntivi sul tuo Account AWS, segui le istruzioni in Eliminazione di un bucket dopo aver esportato la funzione in Infrastructure Composer. Per ulteriori informazioni sul bucket Amazon S3 creato da Lambda, consulta la pagina the section called "Infrastructure Composer".

Per progettare un'applicazione serverless in Infrastructure Composer

Dopo aver attivato la sincronizzazione locale, le modifiche apportate in Infrastructure Composer si rifletteranno nel modello AWS SAM salvato sul computer di compilazione locale. Ora puoi trascinare e rilasciare altre risorse AWS sul canvas di Infrastructure Composer per creare l'applicazione. In questo esempio, aggiungi una coda semplice Amazon SQS come trigger per la funzione Lambda e una tabella DynamoDB in cui la funzione può scrivere i dati.

1. Per aggiungere un trigger Amazon SQS alla funzione Lambda, effettua le seguenti operazioni:

   a. Nel campo di ricerca della palette Risorse, immetti **SQS**.

b. Trascina la risorsa Coda SQS sul canvas e posizionala a sinistra della funzione Lambda.

c. Seleziona Dettagli e per ID logico immetti **`LambdaIaCQueue`**.

d. Selezionare Salva.

e. Connetti le tue risorse Amazon SQS e Lambda facendo clic sulla porta Abbonamento sulla scheda della coda SQS e trascinandola sulla porta sinistra della scheda della funzione Lambda. La comparsa di una linea tra le due risorse indica che la connessione è stabilita. Inoltre, nella parte inferiore del canvas, Infrastructure Composer visualizza un messaggio che indica che le due risorse sono state connesse correttamente.

2. Per aggiungere una tabella Amazon DynamoDB in cui la funzione Lambda può scrivere dati, effettua le seguenti operazioni:

a. Nel campo di ricerca della palette Risorse, immetti **`DynamoDB`**.

b. Trascina la risorsa Tabella DynamoDB sul canvas e posizionala a destra della funzione Lambda.

c. Seleziona Dettagli e per ID logico immetti **`LambdaIaCTable`**.

d. Selezionare Salva.

e. Connetti la tabella DynamoDB alla funzione Lambda facendo clic sulla porta destra della scheda della funzione Lambda e trascinandola sulla porta sinistra della scheda di DynamoDB.

Ora che hai aggiunto queste nuove risorse, diamo un'occhiata al modello AWS SAM aggiornato creato da Infrastructure Composer.

Visualizzazione del modello AWS SAM aggiornato

- Nel canvas di Infrastructure Composer, scegli Modello per passare dalla visualizzazione del canvas alla visualizzazione del modello.

Ora il modello AWS SAM dovrebbe contenere le seguenti risorse e proprietà aggiuntive:

- Una coda Amazon SQS con l'identificativo `LambdaIaCQueue`

```
LambdaIaCQueue:
    Type: AWS::SQS::Queue
    Properties:
```

```
      MessageRetentionPeriod: 345600
```

Quando aggiungi una coda Amazon SQS utilizzando Infrastructure Composer, questo imposta la proprietà `MessageRetentionPeriod`. Puoi anche impostare la proprietà `FifoQueue` selezionando Dettagli sulla scheda della coda SQS e selezionando o deselezionando Coda Fifo.

Per impostare altre proprietà per la coda, puoi modificare manualmente il modello per aggiungerle. Per ulteriori informazioni sulla risorsa `AWS::SQS::Queue` e sulle sue proprietà disponibili, consulta la sezione AWS::SQS::Queue nella Guida per l'utente di CloudFormation.

- Una proprietà `Events` nella definizione della funzione Lambda che specifica la coda Amazon SQS come trigger per la funzione

```
Events:
  LambdaIaCQueue:
    Type: SQS
    Properties:
      Queue: !GetAtt LambdaIaCQueue.Arn
      BatchSize: 1
```

La proprietà `Events` è composta da un tipo di evento e da un insieme di proprietà che dipendono dal tipo. Per ulteriori informazioni sui diversi Servizi AWS che puoi configurare per attivare una funzione Lambda e le proprietà che puoi impostare, consulta la sezione EventSource nella Guida per gli sviluppatori di AWS SAM.

- Una tabella DynamoDB con l'identificatore `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
```

Quando aggiungi una tabella DynamoDB utilizzando Infrastructure Composer, puoi impostare le chiavi della tabella scegliendo Dettagli nella scheda della tabella DynamoDB e modificando i valori delle chiavi. Infrastructure Composer imposta anche i valori predefiniti anche per una serie di altre proprietà, tra cui `BillingMode` e `StreamViewType`.

Per ulteriori informazioni su queste proprietà e sulle altre proprietà che puoi aggiungere al tuo modello AWS SAM, consulta la sezione AWS::DynamoDB::Table nella Guida per l'utente di CloudFormation.

- Una nuova policy IAM che autorizza la funzione a eseguire operazioni CRUD sulla tabella DynamoDB che hai aggiunto.

```
Policies:
...
  - DynamoDBCrudPolicy:
    TableName: !Ref LambdaIaCTable
```

Il modello AWS SAM finale completo avrà un aspetto simile al seguente.

```
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
```

Implementazione dell'applicazione serverless utilizzando AWS SAM (facoltativo)

Se desideri utilizzare AWS SAM per implementare un'applicazione serverless mediante il modello appena creato in Infrastructure Composer, devi prima installare la CLI di AWS SAM. Per farlo, segui le istruzioni riportate nella pagina Installing the AWS SAM CLI.

Inoltre, prima di implementare l'applicazione devi aggiornare il codice della funzione che Infrastructure Composer ha salvato insieme al modello. Al momento, il file `lambda_function.py` salvato da Infrastructure Composer contiene solo il codice di base "Hello world" fornito da Lambda al momento della creazione della funzione.

Per aggiornare il codice della funzione, copia il codice seguente e incollalo nel file `lambda_function.py` che Infrastructure Composer ha salvato sul computer di compilazione locale. Hai specificato la directory in cui Infrastructure Composer deve salvare questo file quando hai attivato la modalità di sincronizzazione locale.

Questo codice accetta una coppia chiave-valore in un messaggio dalla coda Amazon SQS creata in Infrastructure Composer. Se sia la chiave sia il valore sono stringhe, il codice li utilizza per scrivere un elemento nella tabella DynamoDB definita nel modello.

Codice della funzione Python aggiornato

```
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
```

Implementazione dell'applicazione serverless

Per implementare l'applicazione utilizzando la CLI di AWS SAM, effettua le seguenti operazioni. Affinché la funzione venga compilata e implementata correttamente, sul computer di compilazione e nel PATH deve essere installata la versione 3.11 di Python.

1. Esegui il seguente comando dalla directory in cui Infrastructure Composer ha salvato i tuoi file `template.yaml` e `lambda_function.py`.

   ```
   sam build
   ```

   Questo comando raccoglie gli artefatti di compilazione per l'applicazione e li colloca nel formato e nella posizione corretti per l'implementazione.

2. Per implementare l'applicazione e creare le risorse Lambda, Amazon SQS e DynamoDB specificate nel modello AWS SAM, esegui il comando seguente.

```
sam deploy --guided
```

L'utilizzo del flag `--guided` significa che AWS SAM mostrerà le istruzioni per guidarti attraverso il processo di implementazione. Per questa implementazione, accetta le opzioni predefinite premendo Invio.

Durante il processo di implementazione, AWS SAM crea le seguenti risorse nel tuo Account AWS:

- Uno stack CloudFormation denominato `sam-app`
- Una funzione Lambda con il formato del nome `sam-app-LambdaIaCDemo-`*`99VXPpYQVv1M`*
- Una coda Amazon SQS con il formato del nome `sam-app-LambdaIaCQueue-`*`xL87VeKsGiIo`*
- Una tabella DynamoDB con il formato del nome `sam-app-LambdaIaCTable-`*`CN0S66C0VLNV`*

AWS SAM crea anche i ruoli e le policy IAM necessari in modo che la funzione Lambda possa leggere i messaggi dalla coda Amazon SQS ed eseguire operazioni CRUD sulla tabella DynamoDB.

Test dell'applicazione implementata (facoltativo)

Per confermare che l'applicazione serverless sia stata implementata correttamente, invia un messaggio alla coda Amazon SQS contenente una coppia chiave-valore e verifica che Lambda scriva un elemento nella tabella DynamoDB utilizzando questi valori.

Test dell'applicazione serverless

1. Apri la pagina Code della console Amazon SQS e seleziona la coda che AWS SAM ha creato dal modello. Il formato del nome è `sam-app-LambdaIaCQueue-`*`xL87VeKsGiIo`*.
2. Scegli Invio e ricezione di messaggi e incolla il seguente JSON nel Corpo del messaggio nella sezione Invia messaggio.

   ```
   {
       "myKey": "myValue"
   }
   ```
3. Scegliere Invia messaggio.

   L'invio del messaggio alla coda farà sì che Lambda chiami la funzione tramite lo strumento di mappatura dell'origine degli eventi definito nel modello AWS SAM. Per verificare che Lambda

abbia chiamato la funzione come previsto, verifica che alla tabella DynamoDB sia stato aggiunto un elemento.

4. Apri la pagina Tabelle della console DynamoDB e scegli la tabella. Il formato del nome è `sam-app-LambdaIaCTable-CN0S66C0VLNV`.
5. Scegli Explore table items (Esplora elementi della tabella). Nel riquadro Elementi restituiti, dovresti vedere un elemento con l'id `myKey` e il valore `myValue`.

Implementazione delle funzioni Lambda con AWS CDK

AWS Cloud Development Kit (AWS CDK) Si tratta di un framework Infrastructure as Code (IaC) che puoi utilizzare per definire l'infrastruttura AWS cloud utilizzando un linguaggio di programmazione di tua scelta. Per definire la tua infrastruttura cloud, devi innanzitutto scrivere un'app (in uno dei linguaggi supportati dal CDK) contenente uno o più stack. Quindi, lo sintetizzi in un CloudFormation modello e distribuisci le tue risorse sul tuo. Account AWS Segui i passaggi descritti in questo argomento per implementare una funzione Lambda che restituisce un evento da un endpoint Gateway Amazon API.

La AWS Construct Library, inclusa nel CDK, fornisce moduli che è possibile utilizzare per modellare le risorse fornite. Servizi AWS Per i servizi più diffusi, la libreria fornisce costrutti curati con impostazioni predefinite intelligenti e best practice. Puoi utilizzare il modulo aws_lambda per definire la tua funzione e le risorse di supporto con poche righe di codice.

Prerequisiti

Prima di iniziare questo tutorial, installatelo AWS CDK eseguendo il seguente comando.

```
npm install -g aws-cdk
```

Fase 1: Configura il tuo AWS CDK progetto

Crea una directory per la tua nuova AWS CDK app e inizializza il progetto.

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
```

Una volta avviato il progetto, attiva l'ambiente virtuale del progetto e installa le dipendenze di base per AWS CDK.

```
source .venv/bin/activate
python -m pip install -r requirements.txt
```

Java

```
mkdir hello-lambda
cd hello-lambda
cdk init --language java
```

Importa questo progetto Maven nel tuo ambiente di sviluppo integrato (IDE) Java. Ad esempio, in Eclipse, usa File, Importa, Maven, Progetti Maven esistenti.

C#

```
mkdir hello-lambda
cd hello-lambda
cdk init --language csharp
```

> **Note**
>
> Il modello di AWS CDK applicazione utilizza il nome della directory del progetto per generare nomi per i file e le classi di origine. In questo esempio, la directory è denominata `hello-lambda`. Se utilizzi un nome di directory del progetto diverso, l'app non corrisponderà a queste istruzioni.

AWS CDK v2 include costrutti stabili per tutti Servizi AWS in un unico pacchetto chiamato. `aws-cdk-lib` Questo pacchetto viene installato come dipendenza quando inizializzi il progetto. Quando lavori con determinati linguaggi di programmazione, il pacchetto viene installato quando crei il progetto per la prima volta.

Fase 2: Definire lo stack AWS CDK

Uno stack CDK è una raccolta di uno o più costrutti che definiscono le risorse. AWS Ogni stack CDK rappresenta uno stack nell'app CDK. CloudFormation

Per definire lo stack CDK, segui le istruzioni relative al tuo linguaggio di programmazione preferito. Questo stack definisce quanto segue:

- Il nome logico della funzione: `MyFunction`
- La posizione del codice della funzione, specificata nella proprietà `code`. Per ulteriori informazioni, consulta Codice dell'handler nella Documentazione di riferimento API AWS Cloud Development Kit (AWS CDK) .
- Il nome logico della REST API: `HelloApi`
- Il nome logico dell'endpoint API Gateway: `ApiGwEndpoint`

Tieni presente che tutti gli stack CDK di questo tutorial utilizzano il runtime Node.js per la funzione Lambda. È possibile utilizzare diversi linguaggi di programmazione per lo stack CDK e la funzione Lambda per sfruttare i punti di forza di ogni linguaggio. Ad esempio, è possibile utilizzare lo stack CDK TypeScript per sfruttare i vantaggi della digitazione statica per il codice dell'infrastruttura. È possibile utilizzare la funzione Lambda JavaScript per sfruttare la flessibilità e il rapido sviluppo di un linguaggio digitato dinamicamente.

JavaScript

Apri il file `lib/hello-lambda-stack.js` e sostituisci il contenuto con quanto riportato di seguito.

```
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

Apri il file `lib/hello-lambda-stack.ts` e sostituisci il contenuto con quanto riportato di seguito.

```
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

Apri il file /hello-lambda/hello_lambda/hello_lambda_stack.py e sostituisci il contenuto con quanto riportato di seguito.

```
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

Apri il file /hello-lambda/src/main/java/com/myorg/HelloLambdaStack.java e sostituisci il contenuto con quanto riportato di seguito.

```
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#

Apri il file `src/HelloLambda/HelloLambdaStack.cs` e sostituisci il contenuto con quanto riportato di seguito.

```
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
            });
        }
    }
}
```

Fase 3: creazione del codice della funzione Lambda

1. Dalla root del tuo progetto (`hello-lambda`), crea la directory `/lib/lambda-handler` per il codice della funzione Lambda. Questa directory è specificata nella `code` proprietà dello stack. AWS CDK
2. Crea un nuovo file chiamato `index.js` nella `/lib/lambda-handler` directory. Incolla il codice seguente nel file. La funzione estrae proprietà specifiche dalla richiesta API e le restituisce come risposta 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,
  };
};
```

Fase 4: Distribuire lo stack AWS CDK

1. Dalla root del progetto, esegui il comando cdk synth:

   ```
   cdk synth
   ```

   Questo comando sintetizza un AWS CloudFormation modello dallo stack CDK. Il modello è un file YAML di circa 400 righe simile al seguente.

   > **Note**
   >
   > Se viene visualizzato il seguente errore, assicurarti di trovarti nella root della directory del tuo progetto.
   >
   > ```
   > --app is required either in command-line, in cdk.json or in ~/.cdk.json
   > ```

   Example CloudFormation modello

   ```
   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. Esegui il comando cdk deploy:

```
cdk deploy
```

Attendi che le tue risorse vengano create. L'output finale include l'URL per l'endpoint API Gateway. Esempio:

```
Outputs:
HelloLambdaStack.ApiGwEndpoint77F417B1 = https://abcd1234.execute-api.us-
east-1.amazonaws.com/prod/
```

Fase 5: esecuzione del test della funzione

Per richiamare la funzione Lambda, copia l'endpoint API Gateway e incollalo in un browser Web o esegui un comando `curl`:

```
curl -s https://abcd1234.execute-api.us-east-1.amazonaws.com/prod/
```

La risposta è una rappresentazione JSON delle proprietà selezionate dall'oggetto evento originale, che contiene informazioni sulla richiesta effettuata all'endpoint API Gateway. Esempio:

```
{
  "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",
     ...
```

Fase 6: eliminare le risorse

L'endpoint API Gateway è accessibile pubblicamente. Per evitare addebiti imprevisti, esegui il comando cdk destroy per eliminare lo stack e tutte le risorse associate.

```
cdk destroy
```

Fasi successive

Per informazioni sulla scrittura di AWS CDK app nella lingua che preferisci, consulta quanto segue:

TypeScript

- Lavorare con AWS CDK in TypeScript

JavaScript

- Lavorare con l' AWS CDK interno JavaScript

Python

Lavorare con il AWS CDK in Python

Java

Lavorare con il AWS CDK in Java

C#

Lavorare con il AWS CDK in C#

Go

Lavorare con AWS CDK in Go

Utensili elettrici per AWS Lambda

Powertools for AWS Lambda (noto anche come Powertools AWS for) fornisce funzioni di utilità, decoratori e middleware che gestiscono attività Lambda comuni come la registrazione strutturata, il tracciamento, la raccolta di metriche e la convalida degli input. Usa Powertools for AWS Lambda per implementare le migliori pratiche serverless e accelerare lo sviluppo di più funzioni Lambda. In questo modo si semplificano le attività di sviluppo più comuni nelle funzioni Lambda.

Vantaggi principali di Powertools per AWS

Sebbene lo sviluppo di Lambda sia possibile senza PowertoolsAWS, il suo utilizzo offre diversi vantaggi:

- Osservabilità integrata: registrazione strutturata, tracciamento e metriche personalizzate
- Gestione dei segreti: recupero dei parametri, gestione dei segreti e idempotenza
- Progressive Enhancement: scegli le utilità più adatte alle tue esigenze
- Sviluppo accelerato: analisi degli eventi, convalida ed elaborazione in batch
- Migliori pratiche: implementazione di modelli serverless AWS Well-Architected

Integrazione di Powertools con AWS

Powertools for ti AWS aiuta a creare applicazioni serverless pronte per la produzione con meno codice personalizzato. Disponibile in Python, TypeScript/Node.js, .NET e Java, Powertools for

AWS può essere incluso tramite Lambda Layers o utilizzando il gestore di pacchetti linguistici. Ogni implementazione linguistica fornisce funzionalità di base come la registrazione strutturata, il tracciamento, la raccolta di metriche e la gestione degli eventi, pur mantenendo gli idiomi naturali di ogni linguaggio di programmazione. Queste implementazioni sono integrate da componenti specializzati per l'integrazione dei AWS servizi, che supportano il recupero dei parametri, l'elaborazione in batch e la gestione delle API, oltre a best practice come la propagazione degli ID di correlazione, la gestione degli errori e i modelli di idempotenza. Insieme, queste funzionalità consentono agli sviluppatori di creare applicazioni serverless robuste e gestibili, riducendo al contempo il sovraccarico del codice personalizzato.

- Utensili elettrici per AWS Lambda (Python)
- Powertools per AWS Lambda (TypeScript)
- Utensili elettrici per AWS Lambda (Java)
- Utensili elettrici per AWS Lambda (.NET)

Passaggi successivi

Per ulteriori informazioni sull'utilizzo di Powertools perAWS, consulta i seguenti argomenti:

- Elettroutensili per officina AWS Lambda
- Pattern serverless che utilizzano Powertools per AWS
- AWSobiettivo serverless ben progettato
- Creazione di API serverless con Powertools for Lambda AWS

Gestione dei flussi di lavoro e degli eventi Lambda

Quando si creano applicazioni serverless con Lambda, spesso sono necessari modi per orchestrare l'esecuzione delle funzioni e gestire gli eventi. AWS fornisce diversi approcci per coordinare le funzioni Lambda:

- Funzioni durevoli Lambda per l'orchestrazione del flusso di lavoro incentrata sul codice all'interno di Lambda
- AWS Step Functions per l'orchestrazione visiva del flusso di lavoro su più servizi
- Amazon EventBridge Scheduler e Amazon EventBridge per architetture e pianificazioni basate sugli eventi

Puoi anche integrare questi approcci insieme. Ad esempio, è possibile utilizzare EventBridge Scheduler per attivare funzioni durevoli o flussi di lavoro Step Functions quando si verificano eventi specifici o configurare flussi di lavoro per pubblicare eventi su EventBridge Scheduler in punti di esecuzione definiti. I seguenti argomenti di questa sezione forniscono ulteriori informazioni su come utilizzare queste opzioni di orchestrazione.

Orchestrazione incentrata sul codice con funzioni durevoli

Le funzioni durevoli Lambda forniscono un approccio basato sul codice all'orchestrazione del flusso di lavoro, che consente di creare flussi di lavoro statici e di lunga durata direttamente all'interno delle funzioni Lambda. A differenza dei servizi di orchestrazione esterni, le funzioni durevoli mantengono la logica del flusso di lavoro all'interno del codice, semplificandone la versione, il test e la manutenzione insieme alla logica aziendale.

Le funzioni durevoli sono ideali quando hai bisogno di:

- Definizione del flusso di lavoro incentrata sul codice: definisci i flussi di lavoro utilizzando linguaggi di programmazione familiari anziché JSON o visual designer
- Processi a lunga durata: esegui flussi di lavoro che possono durare fino a un anno, ben oltre il limite di 15 minuti delle funzioni Lambda standard
- Sviluppo semplificato: mantieni la logica del flusso di lavoro e la logica aziendale nella stessa base di codice per semplificare la manutenzione e i test
- Attesa conveniente: sospende l'esecuzione durante gli stati di attesa senza consumare risorse di elaborazione
- Gestione dello stato integrata: checkpoint automatico e persistenza dello stato senza configurazione dello storage esterno

Scelta tra funzioni durevoli e Step Functions

Sia Durable Functions che Step Functions offrono funzionalità di orchestrazione del flusso di lavoro, ma sono adatte a diversi casi d'uso:

Considerazione	Funzioni durevoli	Step Functions
Definizione del workflow	Codice primario (, JavaScript Python, Java)	Amazon States Language o visual designer basato su JSON

Considerazione	Funzioni durevoli	Step Functions
Approccio allo sviluppo	Base di codice unica con logica aziendale	Definizione del flusso di lavoro e codice di funzione separati
Integrazione dei servizi	Tramite il codice della funzione Lambda e AWS SDKs	Integrazioni native con molti servizi AWS
Durata dell'esecuzione	Fino a 1 anno	Fino a 1 anno (Standard), 5 minuti (Express)
Elaborazione parallela	Promise.all () e modelli basati su codice	Stato parallelo e mappa distribuita
Gestione degli errori	Blocchi try-catch e logica di ripetizione personalizzata	Stati retry and catch integrati
Monitoraggio visivo	CloudWatch registri e dashboard personalizzati	Grafico di esecuzione visivo e cronologia dettagliata
Ideale per	Flussi di lavoro incentrati sugli sviluppatori, logica aziendale complessa, prototipazione rapida	Orchestrazione multiserv izio, flussi di lavoro visivi, governance aziendale

Utilizza funzioni durevoli quando:

- Il tuo team preferisce approcci di sviluppo incentrati sul codice
- La logica del flusso di lavoro è strettamente associata alla logica aziendale
- Hai bisogno di prototipazione e iterazione rapide
- I tuoi flussi di lavoro coinvolgono principalmente funzioni Lambda e semplici chiamate di servizio

Usa Step Functions quando:

- Hai bisogno di progettazione e monitoraggio visivi del flusso di lavoro
- I tuoi flussi di lavoro orchestrano più servizi in modo estensivo AWS

- Sono necessarie funzionalità di governance e conformità aziendali
- Le parti interessate non tecniche devono comprendere la logica del flusso di lavoro

Per ulteriori informazioni sulle funzioni durevoli, consulta Funzioni durevoli per Lambda.

Orchestrazione dei flussi di lavoro con Step Functions

AWS Step Functions è un servizio di orchestrazione del flusso di lavoro che consente di coordinare più funzioni Lambda e altri AWS servizi in flussi di lavoro strutturati. Questi flussi di lavoro possono mantenere lo stato, gestire gli errori con sofisticati meccanismi di ripetizione dei tentativi ed elaborare i dati su larga scala.

Step Functions offre due tipi di flussi di lavoro per soddisfare diverse esigenze di orchestrazione:

Flussi di lavoro standard

Ideale per flussi di lavoro verificabili di lunga durata che richiedono una semantica di esecuzione esatta. I flussi di lavoro standard possono durare fino a un anno, fornire una cronologia di esecuzione dettagliata e supportare il debug visivo. Sono adatti per processi come l'evasione degli ordini, le pipeline di elaborazione dei dati o i lavori di analisi in più fasi.

Flussi di lavoro rapidi

Progettato per high-event-rate carichi di lavoro di breve durata con at-least-once semantica di esecuzione. I flussi di lavoro Express possono durare fino a cinque minuti e sono ideali per l'elaborazione di eventi ad alto volume, le trasformazioni di dati in streaming o gli scenari di ingestione di dati IoT. Offrono un throughput più elevato e costi potenzialmente inferiori rispetto ai flussi di lavoro standard.

> **Note**
>
> Per ulteriori informazioni sui tipi di flusso di lavoro di Step Functions, vedere Scelta del tipo di flusso di lavoro in Step Functions.

All'interno di questi flussi di lavoro, Step Functions fornisce due tipi di stati della mappa per l'elaborazione parallela:

Mappa in linea

Elabora gli elementi di un array JSON all'interno della cronologia di esecuzione del flusso di lavoro principale. Inline Map supporta fino a 40 iterazioni simultanee ed è adatto per set di dati più piccoli o quando è necessario mantenere tutte le elaborazioni all'interno di un'unica esecuzione. Per ulteriori informazioni, vedere Utilizzo dello stato della mappa in modalità Inline.

Mappa distribuita

Consente l'elaborazione di carichi di lavoro paralleli su larga scala iterando su set di dati che superano i 256 KiB o richiedono più di 40 iterazioni simultanee. Con il supporto per un massimo di 10.000 esecuzioni parallele di flussi di lavoro secondari, Distributed Map eccelle nell'elaborazione di dati semistrutturati archiviati in Amazon S3, come file JSON o CSV, il che la rende ideale per l'elaborazione in batch e le operazioni ETL. Per ulteriori informazioni, consulta Utilizzo dello stato della mappa in modalità distribuita.

Combinando questi tipi di flusso di lavoro e gli stati della mappa, Step Functions fornisce un set di strumenti flessibile e potente per orchestrare applicazioni serverless complesse, da operazioni su piccola scala a pipeline di elaborazione dati su larga scala.

Per iniziare a usare Lambda con Step Functions, vedi Orchestrating Lambda functions with Step Functions.

Gestione degli eventi con Stand Scheduler EventBridge EventBridge

Amazon EventBridge è un servizio di bus per eventi che ti aiuta a creare architetture basate sugli eventi. Instrada gli eventi tra AWS servizi, applicazioni integrate e applicazioni SaaS (Software as a Service). EventBridge Scheduler è uno scheduler senza server che consente di creare, eseguire e gestire attività da un unico servizio centrale, che consente di richiamare le funzioni Lambda in base a una pianificazione utilizzando le espressioni cron e rate o di configurare chiamate singole.

Amazon EventBridge e EventBridge Scheduler ti aiutano a creare architetture basate sugli eventi con Lambda. EventBridge indirizza gli eventi tra AWS servizi, applicazioni integrate e applicazioni SaaS, mentre EventBridge Scheduler fornisce funzionalità di pianificazione specifiche per richiamare le funzioni Lambda su base ricorrente o una tantum.

Questi servizi offrono diverse funzionalità chiave per lavorare con le funzioni Lambda:

- Crea regole che corrispondono e instradano gli eventi alle funzioni Lambda utilizzando EventBridge

- Imposta chiamate di funzioni ricorrenti utilizzando le espressioni cron e rate con Scheduler EventBridge
- Configura le chiamate di funzioni una tantum in date e orari specifici
- Definisci finestre temporali flessibili e riprova le politiche per le chiamate pianificate

Per ulteriori informazioni, consulta Richiamo di una funzione Lambda su una pianificazione.

Funzioni durevoli Lambda

Le funzioni durevoli Lambda consentono di creare applicazioni resilienti in più fasi e flussi di lavoro AI che possono essere eseguiti fino a un anno mantenendo progressi affidabili nonostante le interruzioni. Quando viene eseguita una funzione durevole, questo ciclo di vita completo viene chiamato esecuzione durevole, che utilizza i checkpoint per tenere traccia dei progressi e ripristinare automaticamente i guasti tramite la replay, rieseguendola dall'inizio saltando il lavoro completato.

All'interno di ogni funzione, si utilizzano operazioni durevoli come elementi costitutivi fondamentali. I passaggi eseguono la logica aziendale con nuovi tentativi integrati e il monitoraggio dei progressi, mentre le attese sospendono l'esecuzione senza incorrere in costi di elaborazione, il che li rende ideali per processi di lunga durata come flussi di lavoro o analisi di dipendenze esterne. human-in-the-loop Che si tratti di elaborare ordini, coordinare microservizi o orchestrare applicazioni di intelligenza artificiale agentiche, le funzioni durevoli mantengono lo stato automaticamente e si ripristinano in caso di errori mentre si scrive codice in linguaggi di programmazione familiari.

Vantaggi principali

Scrivi codice resiliente in modo naturale: con costrutti di programmazione familiari, scrivi codice che gestisce automaticamente gli errori. Il checkpoint integrato, i nuovi tentativi trasparenti e il ripristino automatico garantiscono che la logica aziendale rimanga pulita e mirata.

Paghi solo in base all'uso effettivo: durante le operazioni di attesa, la funzione viene sospesa senza incorrere in costi di elaborazione. Per i flussi di lavoro di lunga durata che attendono ore o giorni, paghi solo per il tempo di elaborazione effettivo, non per le attese inattive.

Semplicità operativa: con il modello serverless di Lambda, ottieni la scalabilità automatica, inclusa la gestione dell'infrastruttura. scale-to-zero Le funzioni durevoli gestiscono automaticamente la gestione dello stato, la logica dei tentativi e il ripristino degli errori, riducendo il sovraccarico operativo.

Come funziona

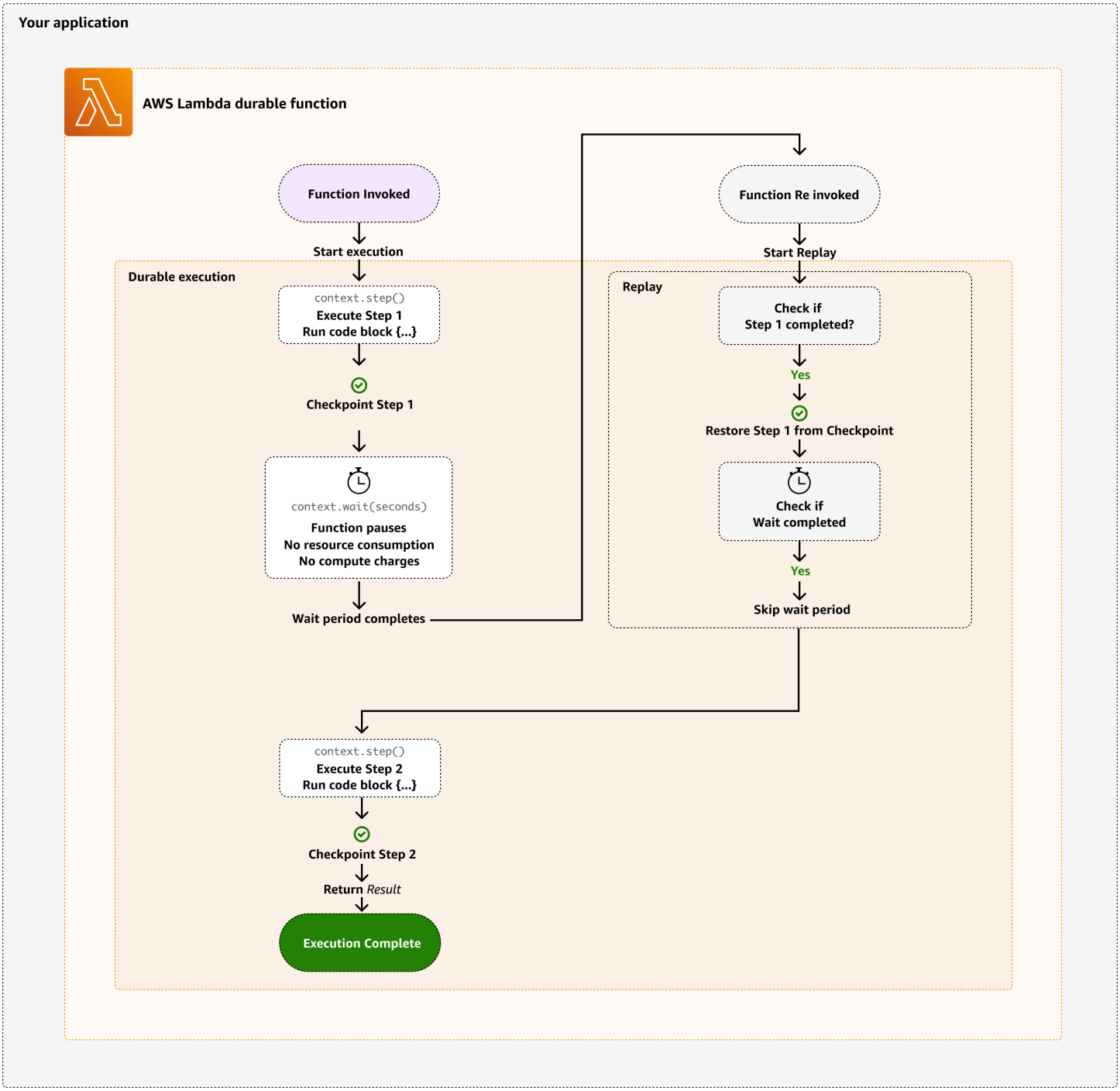
Sotto il cofano, le funzioni durevoli sono normali funzioni Lambda che utilizzano un checkpoint/replay meccanismo per tracciare i progressi e supportare operazioni di lunga durata attraverso punti di sospensione definiti dall'utente, comunemente denominati esecuzione durevole. Quando una funzione durevole viene ripristinata dopo un punto di attesa o un'interruzione, ad esempio dopo un

nuovo tentativo, il sistema esegue la riproduzione. Durante la riproduzione, il codice viene eseguito dall'inizio ma salta i checkpoint completati, utilizzando i risultati memorizzati anziché rieseguire le operazioni completate. Questo meccanismo di replay garantisce la coerenza e consente esecuzioni di lunga durata.

Dopo la ripresa della funzione dopo una pausa o un'interruzione, il sistema esegue la riproduzione. Durante la riproduzione, il codice viene eseguito dall'inizio ma salta i checkpoint completati, utilizzando i risultati memorizzati anziché rieseguire le operazioni completate. Questo meccanismo di replay garantisce la coerenza e consente esecuzioni di lunga durata.

Per sfruttare questo checkpoint-and-replay meccanismo nelle tue applicazioni, Lambda fornisce un SDK di esecuzione durevole. L'SDK elimina la complessità della gestione dei checkpoint e della riproduzione, esponendo semplici primitive chiamate operazioni durevoli utilizzate nel codice. L'SDK è disponibile per JavaScript Python e TypeScript si integra perfettamente con il flusso di lavoro di sviluppo Lambda esistente.

Con l'SDK, crei il tuo gestore di eventi Lambda, che a sua volta ti fornisce DurableContext un complemento all'evento. Questo contesto consente di accedere a operazioni durature come passaggi e attese. La logica delle funzioni viene scritta come un normale codice sequenziale, ma invece di chiamare direttamente i servizi, le chiamate vengono suddivise in fasi per il checkpoint automatico e i nuovi tentativi. Quando è necessario sospendere l'esecuzione, si aggiungono attese che sospendono la funzione senza incorrere in addebiti. L'SDK gestisce tutta la complessità della gestione dello stato e della riproduzione in background, in modo che il codice rimanga pulito e leggibile.



Quando utilizzare le funzioni durevoli

Coordinamento di breve durata: coordina i pagamenti, l'inventario e le spedizioni su più servizi con ripristino automatico in caso di guasti. Elabora gli ordini tramite convalida, autorizzazione al pagamento, allocazione dell'inventario ed evasione con completamento garantito.

Elabora i pagamenti con sicurezza: crea flussi di pagamento resilienti che mantengano lo stato delle transazioni anche in caso di errori e gestiscano automaticamente i nuovi tentativi. Coordina l'autorizzazione in più fasi, i controlli antifrode e il regolamento tra i fornitori di servizi di pagamento con la piena verificabilità in tutte le fasi.

Crea flussi di lavoro AI affidabili: crea flussi di lavoro di intelligenza artificiale in più fasi che concatenano le chiamate, incorporano il feedback umano e gestiscono in modo deterministico le attività di lunga durata in caso di guasto. Riprendi automaticamente dopo la sospensione e paghi solo per il tempo di esecuzione attivo.

Orchestra l'evasione degli ordini complessi: coordina l'elaborazione degli ordini tra sistemi di inventario, pagamento, spedizione e notifica con una resilienza integrata. Gestisci automaticamente gli errori parziali, preserva lo stato degli ordini nonostante le interruzioni e attendi in modo efficiente gli eventi esterni senza consumare risorse di elaborazione.

Automatizza i flussi di lavoro aziendali in più fasi: crea flussi di lavoro affidabili per l'onboarding dei dipendenti, l'approvazione dei prestiti e i processi di conformità che durano giorni o settimane. Mantieni lo stato del flusso di lavoro tra approvazioni umane, integrazioni di sistema e attività pianificate, fornendo al contempo una visibilità completa sullo stato e sulla cronologia dei processi.

Fasi successive

- Inizia con funzioni durevoli
- Esplora l'SDK di esecuzione durevole
- Monitora ed esegui il debug di funzioni durature
- Verifica la sicurezza e le autorizzazioni
- Segui le migliori pratiche

Concetti di base

Lambda fornisce un'esecuzione durevole SDKs per JavaScript TypeScript, e Python. Queste SDKs sono le basi per creare funzioni durevoli, che forniscono le primitive necessarie per controllare i progressi, gestire i nuovi tentativi e gestire il flusso di esecuzione. Per la documentazione e gli esempi completi sull'SDK, consulta JavaScript/TypeScript SDK e Python SDK on. GitHub

Esecuzione durevole

Un'esecuzione durevole rappresenta il ciclo di vita completo di una funzione durevole Lambda, che utilizza un meccanismo di checkpoint e replay per tracciare l'avanzamento della logica aziendale, sospendere l'esecuzione e ripristinare in caso di errori. Quando le funzioni riprendono dopo la sospensione o l'interruzione, i checkpoint precedentemente completati vengono riprodotti e la funzione continua l'esecuzione.

Il ciclo di vita può includere più invocazioni di una funzione Lambda per completare l'esecuzione, in particolare dopo sospensioni o guasti. Questo approccio consente alla funzione di funzionare per periodi prolungati (fino a un anno) mantenendo un progresso affidabile nonostante le interruzioni.

Come funziona il replay

Lambda mantiene un registro in esecuzione di tutte le operazioni durevoli (passaggi, attese e altre operazioni) durante l'esecuzione della funzione. Quando la funzione deve essere messa in pausa o incontra un'interruzione, Lambda salva questo registro dei checkpoint e interrompe l'esecuzione. Quando è il momento di riprendere, Lambda richiama nuovamente la funzione dall'inizio e riproduce il registro del checkpoint, sostituendo i valori memorizzati con le operazioni completate. Ciò significa che il codice viene eseguito nuovamente, ma i passaggi completati in precedenza non vengono rieseguiti. Al loro posto vengono utilizzati i risultati memorizzati.

Questo meccanismo di riproduzione è fondamentale per comprendere le funzioni durevoli. Il codice deve essere deterministico durante la riproduzione, il che significa che produce gli stessi risultati con gli stessi input. Evita le operazioni con effetti collaterali (come la generazione di numeri casuali o la visualizzazione dell'ora corrente) al di fuori dei passaggi, poiché queste possono produrre valori diversi durante la riproduzione e causare comportamenti non deterministici.

DurableContext

DurableContextè l'oggetto contestuale che riceve la tua funzione durevole. Fornisce metodi per operazioni durature come passaggi e attese che creano checkpoint e gestiscono il flusso di esecuzione.

La tua funzione duratura riceve un contesto Lambda `DurableContext` invece del contesto Lambda predefinito:

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
```

L'SDK Python per funzioni durevoli utilizza metodi sincroni e non supporta. `await` L' TypeScript SDK utilizza. `async/await`

Fasi

Steps esegue la logica aziendale con nuovi tentativi integrati e checkpoint automatico. Ogni passaggio salva il risultato, assicurando che la funzione possa riprendere da qualsiasi passaggio completato dopo le interruzioni.

TypeScript

```
// Each step is automatically checkpointed
const order = await context.step(async () => processOrder(event));
const payment = await context.step(async () => processPayment(order));
const result = await context.step(async () => completeOrder(payment));
```

Python

```
# Each step is automatically checkpointed
order = context.step(lambda: process_order(event))
payment = context.step(lambda: process_payment(order))
result = context.step(lambda: complete_order(payment))
```

Stati di attesa

Gli stati di attesa sono pause pianificate in cui la funzione smette di funzionare (e interrompe la ricarica) fino al momento di continuare. Utilizzali per attendere periodi di tempo, richiamate esterne o condizioni specifiche.

TypeScript

```
// Wait for 1 hour without consuming resources
await context.wait({ seconds:3600 });

// Wait for external callback
const approval = await context.waitForCallback(
  async (callbackId) => sendApprovalRequest(callbackId)
);
```

Python

```
# Wait for 1 hour without consuming resources
context.wait(3600)

# Wait for external callback
approval = context.wait_for_callback(
    lambda callback_id: send_approval_request(callback_id)
)
```

Quando la funzione incontra un'attesa o deve essere messa in pausa, Lambda salva il registro del checkpoint e interrompe l'esecuzione. Quando è il momento di riprendere, Lambda richiama nuovamente la funzione e riproduce il registro del checkpoint, sostituendo i valori memorizzati con le operazioni completate.

Per flussi di lavoro più complessi, le funzioni Lambda durevoli includono anche operazioni avanzate `parallel()` come l'esecuzione simultanea, l'elaborazione di array `map()`, `runInChildContext()`, le operazioni annidate e il polling. `waitForCondition()` Vedi Esempi per esempi dettagliati e indicazioni su quando utilizzare ciascuna operazione.

Richiamo di altre funzioni

Invoke consente a una funzione durevole di chiamare altre funzioni Lambda e attendere i loro risultati. La funzione chiamante si sospende durante l'esecuzione della funzione richiamata, creando un checkpoint che conserva il risultato. Ciò consente di creare flussi di lavoro modulari in cui funzioni specializzate gestiscono attività specifiche.

`context.invoke()`Utilizzatelo per richiamare altre funzioni dall'interno della vostra funzione duratura. La chiamata ha un checkpoint, quindi se la funzione viene interrotta dopo il completamento della funzione richiamata, riprende con il risultato memorizzato senza richiamare nuovamente la funzione.

TypeScript

```
// Invoke another function and wait for result
const customerData = await context.invoke(
  'validate-customer',
  'arn:aws:lambda:us-east-1:123456789012:function:customer-service:1',
  { customerId: event.customerId }
);

// Use the result in subsequent steps
const order = await context.step(async () => {
  return processOrder(customerData);
});
```

Python

```
# Invoke another function and wait for result
customer_data = context.invoke(
```

```
    'arn:aws:lambda:us-east-1:123456789012:function:customer-service:1',
    {'customerId': event['customerId']},
    name='validate-customer'
)

# Use the result in subsequent steps
order = context.step(
    lambda: process_order(customer_data),
    name='process-order'
)
```

La funzione richiamata può essere una funzione Lambda durevole o standard. Se richiamate una funzione duratura, la funzione chiamante attende il completamento dell'esecuzione duratura completa. Questo modello è comune nelle architetture di microservizi in cui ogni funzione gestisce un dominio specifico e consente di comporre flussi di lavoro complessi a partire da funzioni specializzate e riutilizzabili.

> **Note**
>
> Le chiamate tra account non sono supportate. La funzione richiamata deve trovarsi nello stesso AWS account della funzione chiamante.

Configurazione durevole delle funzioni

Le funzioni durevoli hanno impostazioni di configurazione specifiche che controllano il comportamento di esecuzione e la conservazione dei dati. Queste impostazioni sono separate dalla configurazione standard della funzione Lambda e si applicano all'intero ciclo di vita dell'esecuzione durevole.

L'DurableConfigoggetto definisce la configurazione per le funzioni durevoli:

```
{
  "ExecutionTimeout": Integer,
  "RetentionPeriodInDays": Integer
}
```

Execution timeout (Timeout di esecuzione)

Il timeout di esecuzione controlla per quanto tempo può durare un'esecuzione durevole dall'inizio al completamento. Questo è diverso dal timeout della funzione Lambda, che controlla per quanto tempo può essere eseguita una singola chiamata di funzione.

Un'esecuzione durevole può includere più chiamate di funzioni Lambda man mano che avanza attraverso checkpoint, attese e replay. Il timeout di esecuzione si applica al tempo totale trascorso dell'esecuzione durevole, non alle chiamate di singole funzioni.

Comprendere la differenza

Il timeout della funzione Lambda (massimo 15 minuti) limita ogni singola chiamata della funzione. Il timeout di esecuzione duraturo (massimo 1 anno) limita il tempo totale dall'inizio dell'esecuzione al completamento, all'errore o al timeout. Durante questo periodo, la funzione può essere richiamata più volte durante l'elaborazione delle fasi, le attese e il ripristino in caso di errori.

Ad esempio, se imposti un timeout di esecuzione duraturo di 24 ore e un timeout della funzione Lambda di 5 minuti:

- Ogni chiamata di funzione deve essere completata entro 5 minuti
- L'intera esecuzione durevole può durare fino a 24 ore
- La tua funzione può essere richiamata più volte durante quelle 24 ore
- Le operazioni di attesa non vengono conteggiate ai fini del timeout della funzione Lambda, ma ai fini del timeout di esecuzione

Puoi configurare il timeout di esecuzione quando crei una funzione durevole utilizzando la console Lambda AWS CLI, oppure. AWS SAM Nella console Lambda, scegli la tua funzione, quindi Configurazione, Esecuzione durevole. Imposta il valore del timeout di esecuzione in secondi (impostazione predefinita: 86400 secondi/24 ore, minimo: 60 secondi, massimo: 31536000 secondi/1 anno).

> **Note**
>
> Il timeout di esecuzione e il timeout della funzione Lambda sono impostazioni diverse. Il timeout della funzione Lambda controlla per quanto tempo può essere eseguita ogni singola chiamata (massimo 15 minuti). Il timeout di esecuzione controlla il tempo totale trascorso per l'intera esecuzione durevole (massimo 1 anno).

Periodo di conservazione

Il periodo di conservazione controlla per quanto tempo Lambda conserva la cronologia di esecuzione e i dati dei checkpoint dopo il completamento di un'esecuzione duratura. Questi dati includono i risultati delle fasi, lo stato di esecuzione e il registro completo del checkpoint.

Dopo la scadenza del periodo di conservazione, Lambda elimina la cronologia di esecuzione e i dati del checkpoint. Non è più possibile recuperare i dettagli dell'esecuzione o ripetere l'esecuzione. Il periodo di conservazione inizia quando l'esecuzione raggiunge uno stato terminale (SUCCEEDED, FAILED, STOPED o TIMED_OUT).

Puoi configurare il periodo di conservazione quando crei una funzione durevole utilizzando la console Lambda AWS CLI, oppure. AWS SAM Nella console Lambda, scegli la tua funzione, quindi Configurazione, Esecuzione durevole. Imposta il valore del periodo di conservazione in giorni (impostazione predefinita: 14 giorni, minimo: 1 giorno, massimo: 90 giorni).

Scegli un periodo di conservazione in base ai requisiti di conformità, alle esigenze di debug e alle considerazioni relative ai costi. Periodi di conservazione più lunghi offrono più tempo per il debug e il controllo, ma aumentano i costi di storage.

Creazione di funzioni Lambda durevoli

Per iniziare a usare le funzioni durevoli Lambda, usa la console Lambda per creare una funzione durevole. In pochi minuti, puoi creare e implementare una funzione durevole che utilizza passaggi e attese per dimostrare l'esecuzione basata su checkpoint.

Mentre svolgi il tutorial, imparerai i concetti fondamentali delle funzioni durevoli, come usare l'`DurableContext`oggetto, creare checkpoint con passaggi e mettere in pausa l'esecuzione con attese. Imparerai anche come funziona il replay quando la funzione riprende dopo un'attesa.

A scopo di semplificazione, crea la funzione utilizzando il runtime Python o Node.js. Con questi linguaggi interpretati, puoi modificare il codice della funzione direttamente nell'editor del codice integrato della console.

> **Tip**
>
> Per scoprire come creare soluzioni serverless, consulta la Guida allo sviluppo serverless.

Prerequisiti

Iscriviti per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la https://portal.aws.amazon.com/billing/registrazione.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata o un messaggio di testo e ti verrà chiesto di inserire un codice di verifica attraverso la tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

AWS ti invia un'email di conferma dopo il completamento della procedura di registrazione. In qualsiasi momento, puoi visualizzare l'attività corrente del tuo account e gestirlo accedendo a https://aws.amazon.com/e scegliendo Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi Console di gestione AWScome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .
2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita il Centro identità IAM.

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .

2. In IAM Identity Center, assegna l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con l'impostazione predefinita IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accesso come utente amministratore

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso a ulteriori utenti

1. In IAM Identity Center, crea un set di autorizzazioni conforme alla best practice dell'applicazione di autorizzazioni con il privilegio minimo.

   Segui le istruzioni riportate nella pagina Creazione di un set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .

2. Assegna al gruppo prima gli utenti e poi l'accesso con autenticazione unica (Single Sign-On).

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente di AWS IAM Identity Center .

Crea una funzione Lambda durevole con la console

In questo esempio, la tua funzione duratura elabora un ordine attraverso più passaggi con checkpoint automatico. La funzione accetta un oggetto JSON contenente un ID dell'ordine, convalida l'ordine, elabora il pagamento e conferma l'ordine. Ogni passaggio viene controllato automaticamente, quindi se la funzione viene interrotta, riprende dall'ultimo passaggio completato.

La funzione dimostra anche un'operazione di attesa, che sospende l'esecuzione per un breve periodo per simulare l'attesa di una conferma esterna.

Per creare una funzione duratura con la console

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli Crea funzione.
3. Scegli Crea da zero.
4. Nel riquadro Informazioni base, per Nome funzione inserisci **`myDurableFunction`**.
5. Per Runtime, scegli Node.js 24 o Python 3.14.
6. Seleziona Abilita esecuzione duratura.

Lambda crea una funzione durevole con un ruolo di esecuzione che include le autorizzazioni per le operazioni di checkpoint (e). `lambda:CheckpointDurableExecutions` `lambda:GetDurableExecutionState`

> **Note**
>
> I runtime Lambda includono l'SDK Durable Execution, che consente di testare funzioni durevoli senza dipendenze dai pacchetti. Tuttavia, consigliamo di includere l'SDK nel pacchetto di distribuzione per la produzione. Ciò garantisce la coerenza delle versioni ed evita potenziali aggiornamenti di runtime che potrebbero influire sulla funzionalità.

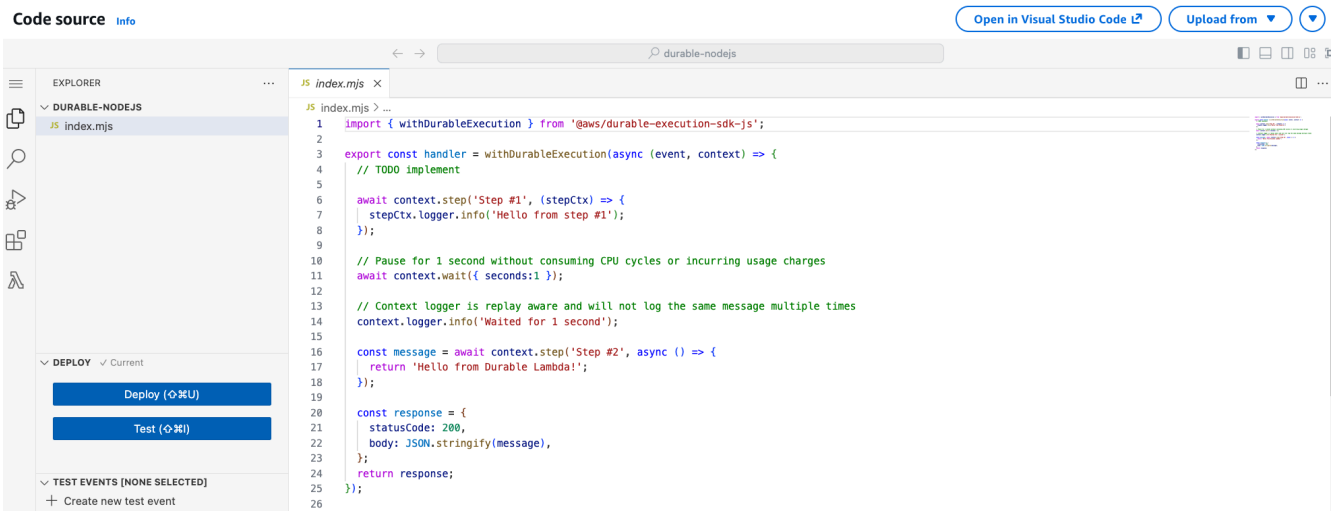
Usa l'editor di codice integrato nella console per aggiungere il tuo codice funzionale durevole.

Node.js

Per modificare il codice nella console

1. Scegli la scheda Codice.

   Nell'editor di codice integrato della console, dovresti vedere il codice della funzione creato da Lambda. Se non vedi la scheda index.js nell'editor di codice, seleziona index.js in Esplora file, come illustrato nel diagramma seguente.



2. Incolla il codice seguente nella scheda index.js, sostituendo il codice creato da Lambda.

```
import {
  DurableContext,
  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 };
    });

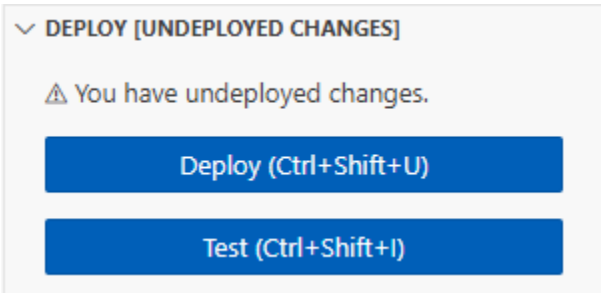
    // Wait for 10 seconds to simulate external confirmation
    await context.wait({ seconds: 10 });

    // 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. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



Comprensione del codice funzionale durevole

Prima di passare alla fase successiva, esaminiamo il codice della funzione e comprendiamo i concetti chiave relativi alle funzioni durevoli.

- L'`withDurableExecution`involucro:

  La tua funzione durevole è racchiusa in. `withDurableExecution` Questo wrapper consente un'esecuzione duratura fornendo l'`DurableContext`oggetto e gestendo le operazioni del checkpoint.

- L'oggetto: `DurableContext`

  Invece del contesto Lambda standard, la funzione riceve un. `DurableContext` Questo oggetto fornisce metodi per operazioni durevoli come `step()` e `wait()` che creano checkpoint.

- Passaggi e punti di controllo:

  Ogni `context.step()` chiamata crea un checkpoint prima e dopo l'esecuzione. Se la funzione viene interrotta, riprende dall'ultimo checkpoint completato. La funzione non riesegue i passaggi completati. Utilizza invece i risultati memorizzati.

- Operazioni di attesa:

La `context.wait()` chiamata sospende l'esecuzione senza consumare risorse di calcolo. Al termine dell'attesa, Lambda richiama nuovamente la funzione e riproduce il registro del checkpoint, sostituendo i valori memorizzati con i passaggi completati.

- Meccanismo di riproduzione:

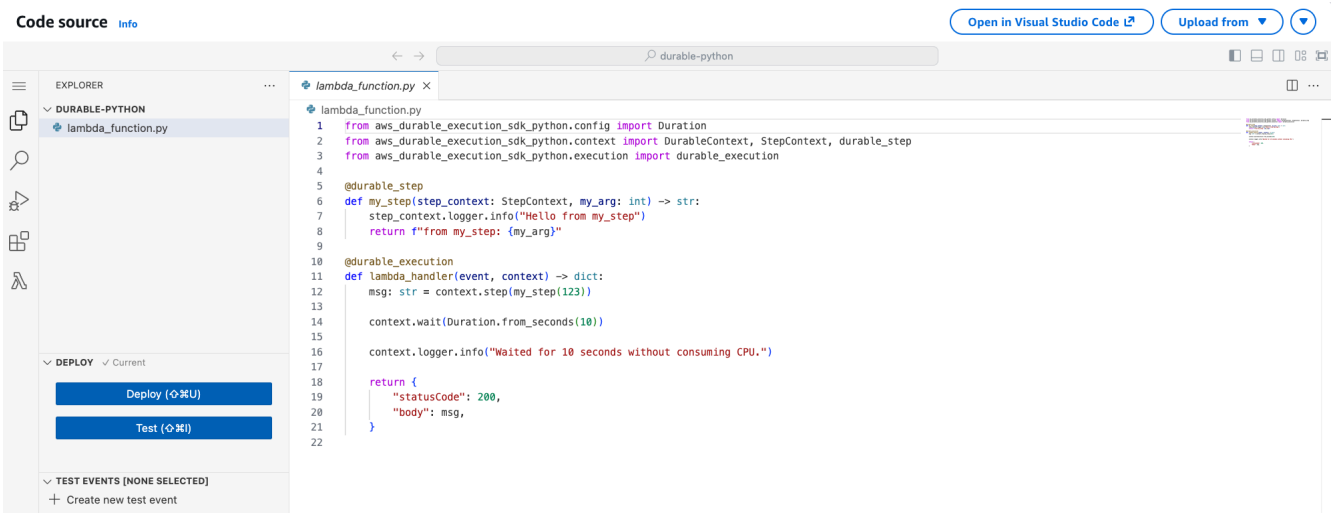
  Quando la funzione riprende dopo un'attesa o un'interruzione, Lambda esegue il codice dall'inizio. Tuttavia, i passaggi completati non vengono rieseguiti. Lambda riproduce i risultati dal registro dei checkpoint. Ecco perché il codice deve essere deterministico.

Python

Per modificare il codice nella console

1. Scegli la scheda Codice.

   Nell'editor di codice integrato della console, dovresti vedere il codice della funzione creato da Lambda. Se non vedi la scheda lambda_function.py nell'editor di codice, seleziona lambda_function.py in Esplora file, come illustrato nel diagramma seguente.



2. Incolla il codice seguente nella scheda lambda_function.py, sostituendo il codice creato da 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))

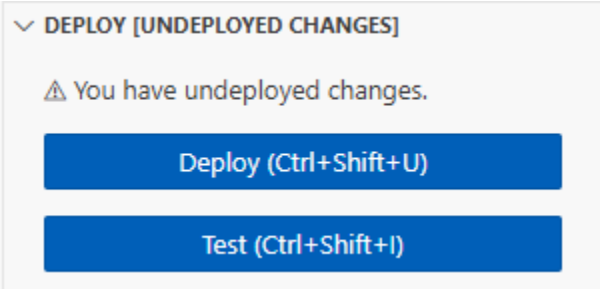
    # Step 2: Process payment
    payment_result = context.step(process_payment(order_id))

    # 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. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



Comprensione del codice funzionale durevole

Prima di passare alla fase successiva, esaminiamo il codice della funzione e comprendiamo i concetti chiave relativi alle funzioni durevoli.

- Il `@durable_execution` decoratore:

  La tua funzione di gestore è decorata con. `@durable_execution` Questo decoratore consente un'esecuzione duratura fornendo l'`DurableContext`oggetto e gestendo le operazioni del checkpoint.
- Il `@durable_step` decoratore:

  Ogni funzione Step è decorata con`@durable_step`. Questo decoratore contrassegna la funzione come un gradino durevole che crea posti di blocco.
- L'`DurableContext`oggetto:

  Invece del contesto Lambda standard, la funzione riceve un. `DurableContext` Questo oggetto fornisce metodi per operazioni durevoli come `step()` e `wait()` che creano checkpoint.
- Passaggi e punti di controllo:

  Ogni `context.step()` chiamata crea un checkpoint prima e dopo l'esecuzione. Se la funzione viene interrotta, riprende dall'ultimo checkpoint completato. La funzione non riesegue i passaggi completati. Utilizza invece i risultati memorizzati.
- Operazioni di attesa:

  La `context.wait()` chiamata sospende l'esecuzione senza consumare risorse di calcolo. Al termine dell'attesa, Lambda richiama nuovamente la funzione e riproduce il registro del checkpoint, sostituendo i valori memorizzati con i passaggi completati.
- Python SDK è sincrono:

Nota che l'SDK Python non lo usa. `await` Tutte le operazioni durevoli sono chiamate di metodo sincrone.

Invoca la funzione durable utilizzando l'editor di codice della console

Le funzioni durevoli richiedono un ARN qualificato per la chiamata. Prima di poter richiamare la funzione duratura, pubblicane una versione.

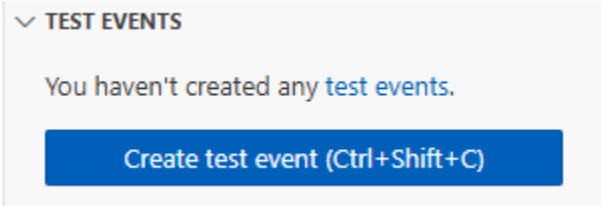
Per pubblicare una versione della tua funzione

1. Scegli la scheda Versioni.
2. Selezionare Publish new version (Pubblica nuova versione).
3. Per la descrizione della versione, inserisci **`Initial version`** (opzionale).
4. Seleziona Pubblica.
5. Lambda crea la versione 1 della tua funzione. Nota che la funzione ARN ora include `:1` alla fine, a indicare che si tratta della versione 1.

Ora crea un evento di test da inviare alla tua funzione. L'evento è un documento in formato JSON contenente un ID dell'ordine.

Per creare un evento di test

1. Nella sezione TEST EVENTS dell'editor di codice della console, scegli Crea evento di test.



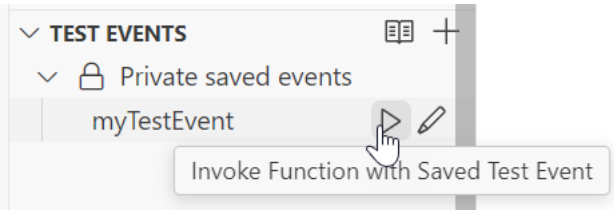
2. Per Event name (Nome evento), immettere **`myTestEvent`**.
3. Nella sezione JSON dell'evento, sostituisci il codice JSON predefinito con il seguente:

```
{
  "orderId": "order-12345"
}
```

4. Scegli Save (Salva).

Per testare la tua funzione duratura e visualizzarne l'esecuzione

Nella sezione TEST EVENTS dell'editor di codice della console, scegli l'icona di esecuzione accanto all'evento di test:



La tua funzione durevole inizia l'esecuzione. Poiché include un'attesa di 10 secondi, la chiamata iniziale viene completata rapidamente e la funzione riprende dopo il periodo di attesa. È possibile visualizzare l'avanzamento dell'esecuzione nella scheda Esecuzioni durevoli.

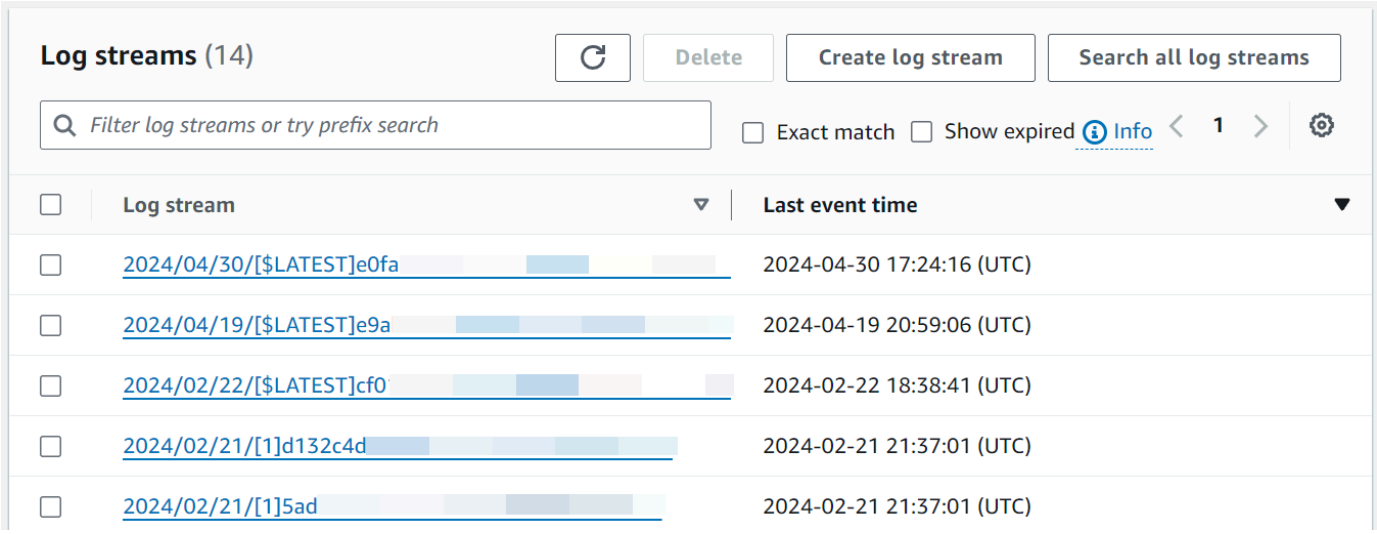
Per visualizzare l'esecuzione duratura della funzione

1. Scegli la scheda Esecuzioni durevoli.
2. Cerca la tua esecuzione nell'elenco. L'esecuzione mostra lo stato attuale (In esecuzione, Riuscito o Non riuscito).
3. Scegli l'ID di esecuzione per visualizzare i dettagli, tra cui:
   - Cronologia dell'esecuzione che mostra quando ogni passaggio è stato completato
   - Cronologia dei checkpoint
   - Periodi di attesa
   - Risultati delle fasi

Puoi anche visualizzare i log della tua funzione in CloudWatch Logs per vedere l'output della console di ogni passaggio.

Per visualizzare i record di invocazione della funzione in Logs CloudWatch

1. Apri la pagina Log groups della console. CloudWatch
2. Scegli il nome del gruppo di log per la funzione (`/aws/lambda/myDurableFunction`).
3. Scorri verso il basso e scegli il flusso di log per le invocazioni delle funzioni che desideri esaminare.



Dovrebbero essere visualizzate le voci di registro per ogni chiamata della funzione, inclusa l'esecuzione iniziale e la riproduzione dopo l'attesa.

Eliminazione

Quando hai finito di lavorare con la funzione durable di esempio, eliminala. Puoi anche eliminare il gruppo di log che memorizza i log della funzione e il ruolo di esecuzione creato dalla console.

Per eliminare la funzione Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **`confirm`** nel campo di immissione del testo, quindi scegli Elimina.

Come eliminare il gruppo di log

1. Apri la pagina Log groups page (Pagina dei gruppi di log) della console CloudWatch.
2. Scegli il gruppo di log della funzione (`/aws/lambda/myDurableFunction`).
3. Scegli Actions (Azioni), Delete log group(s) (Elimina gruppo/i di log).
4. Nella finestra di dialogo Delete log group(s) (Elimina gruppo/i di log) scegli Delete (Elimina).

Come eliminare il ruolo di esecuzione

1. Apri la pagina Ruoli della console AWS Identity and Access Management (IAM).
2. Seleziona il ruolo di esecuzione della funzione (ad esempio, `myDurableFunction-role-31exxmpl`).
3. Scegli Elimina.
4. Nella finestra di dialogo Elimina ruolo, immetti il nome del ruolo, quindi scegli Elimina.

Risorse aggiuntive e fasi successive

Ora che hai creato e testato una funzione semplice e duratura utilizzando la console, procedi nel seguente modo:

- Scopri i casi d'uso più comuni per funzioni durevoli, tra cui transazioni distribuite, elaborazione degli ordini e flussi di lavoro di revisione umana. Vedi esempi.
- Scopri come monitorare le esecuzioni durevoli delle funzioni con CloudWatch metriche e cronologia delle esecuzioni. Vedi Monitoraggio e debug.
- Scopri come richiamare funzioni durevoli in modo sincrono e asincrono e come gestire esecuzioni di lunga durata. Vedi Richiamo di funzioni durevoli.
- Segui le migliori pratiche per scrivere codice deterministico, gestire le dimensioni dei checkpoint e ottimizzare i costi. Consulta le migliori pratiche.
- Scopri come testare funzioni durevoli a livello locale e nel cloud. Vedi Testare funzioni durevoli.

Implementa e richiama le funzioni durevoli Lambda con AWS CLI

Usa il AWS CLI per creare e distribuire funzioni durevoli Lambda con comandi imperativi. Questo approccio ti offre il controllo diretto su ogni fase del processo di distribuzione.

Prerequisiti

- Istalla e configura la AWS CLI. Per istruzioni, vedere Installazione di AWS CLI.
- Crea un pacchetto di distribuzione con il codice della funzione e l'SDK di esecuzione durevole.
- Crea un ruolo di esecuzione IAM con autorizzazioni di checkpoint.

Creazione del ruolo di esecuzione

Crea un ruolo IAM con autorizzazioni per l'esecuzione Lambda di base e le operazioni di checkpoint.

Per creare il ruolo di esecuzione

1. Crea un documento sulla politica di fiducia che consenta a Lambda di assumere il ruolo. Salva questo come`trust-policy.json`:

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

2. Crea il ruolo:

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

3. Allega la politica di esecuzione di base:

```
aws iam attach-role-policy \
  --role-name durable-function-role \
  --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

4. Crea una politica per le operazioni ai checkpoint. Salva questo come`checkpoint-policy.json`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": [
        "lambda:CheckpointDurableExecutions",
        "lambda:GetDurableExecutionState"
      ],
      "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:myDurableFunction"
    }
  ]
}
```

5. Crea e allega la politica del checkpoint:

```
aws iam create-policy \
  --policy-name durable-checkpoint-policy \
  --policy-document file://checkpoint-policy.json

aws iam attach-role-policy \
  --role-name durable-function-role \
  --policy-arn arn:aws:iam::ACCOUNT_ID:policy/durable-checkpoint-policy
```

Crea la funzione durevole

Crea la tua funzione duratura con il `--durable-config` parametro.

Per creare una funzione duratura

1. Package del codice della funzione con le dipendenze in un file.zip:

```
zip -r function.zip index.mjs node_modules/
```

2. Crea la funzione con l'esecuzione duratura abilitata:

```
aws lambda create-function \
  --function-name myDurableFunction \
  --runtime nodejs22.x \
  --role arn:aws:iam::ACCOUNT_ID:role/durable-function-role \
  --handler index.handler \
  --zip-file fileb://function.zip \
  --durable-config '{"ExecutionTimeout": 10, "RetentionPeriodInDays":1}'
```

> **Note**
>
> È possibile abilitare l'esecuzione duratura solo durante la creazione della funzione. Non è possibile abilitarla su funzioni esistenti.

Pubblica una versione

Le funzioni durevoli richiedono un ARN qualificato (con versione o alias) per la chiamata. Pubblica una versione della tua funzione:

```
aws lambda publish-version \
  --function-name myDurableFunction \
  --description "Initial version"
```

Il comando restituisce la versione ARN. Annotate il numero di versione (ad esempio, `:1`) alla fine dell'ARN.

Facoltativamente, create un alias che punti alla versione:

```
aws lambda create-alias \
  --function-name myDurableFunction \
  --name prod \
  --function-version 1
```

Invoca la funzione duratura

Richiama la tua funzione duratura utilizzando l'ARN qualificato (versione o alias).

Invocazione sincrona

Per le esecuzioni che vengono completate entro 15 minuti, usa l'invocazione sincrona:

```
aws lambda invoke \
  --function-name myDurableFunction:1 \
  --payload '{"orderId": "order-12345"}' \
  --cli-binary-format raw-in-base64-out \
  response.json
```

O usando un alias:

```
aws lambda invoke \
  --function-name myDurableFunction:prod \
  --payload '{"orderId": "order-12345"}' \
  --cli-binary-format raw-in-base64-out \
  response.json
```

Invocazione asincrona

Per le esecuzioni di lunga durata, usa la chiamata asincrona:

```
aws lambda invoke \
  --function-name myDurableFunction:prod \
  --invocation-type Event \
  --payload '{"orderId": "order-12345"}' \
  --cli-binary-format raw-in-base64-out \
  response.json
```

Con la chiamata asincrona, Lambda ritorna immediatamente. L'esecuzione della funzione continua in background.

Gestisci esecuzioni durevoli

Utilizzate i seguenti comandi per gestire e monitorare le esecuzioni di funzioni durevoli.

Elenca le esecuzioni

Elenca tutte le esecuzioni per una funzione durevole:

```
aws lambda list-durable-executions \
  --function-name myDurableFunction:prod
```

Ottieni i dettagli dell'esecuzione

Ottieni dettagli su un'esecuzione specifica:

```
aws lambda get-durable-execution \
  --function-name myDurableFunction:prod \
  --execution-id exec-abc123
```

Ottieni la cronologia delle esecuzioni

Visualizza la cronologia dei checkpoint per un'esecuzione:

```
aws lambda get-durable-execution-history \
  --function-name myDurableFunction:prod \
  --execution-id exec-abc123
```

Interrompi un'esecuzione

Interrompi un'esecuzione durevole in esecuzione:

```
aws lambda stop-durable-execution \
  --function-name myDurableFunction:prod \
  --execution-id exec-abc123
```

Aggiornamento del codice della funzione

Aggiorna il codice della funzione durevole e pubblica una nuova versione:

Per aggiornare e pubblicare una nuova versione

1. Aggiorna il codice della funzione:

   ```
   aws lambda update-function-code \
     --function-name myDurableFunction \
     --zip-file fileb://function.zip
   ```

2. Attendi il completamento dell'aggiornamento:

   ```
   aws lambda wait function-updated \
     --function-name myDurableFunction
   ```

3. Pubblica una nuova versione:

   ```
   aws lambda publish-version \
     --function-name myDurableFunction \
     --description "Updated order processing logic"
   ```

4. Aggiorna l'alias in modo che punti alla nuova versione:

   ```
   aws lambda update-alias \
     --function-name myDurableFunction \
     --name prod \
   ```

```
  --function-version 2
```

> **Important**
>
> Le esecuzioni in esecuzione continuano a utilizzare la versione con cui sono iniziate. Le nuove chiamate utilizzano la versione aggiornata dell'alias.

Visualizzare i log delle funzioni

Visualizza i log della tua funzione durevole in Logs: CloudWatch

```
aws logs tail /aws/lambda/myDurableFunction --follow
```

Filtra i log per un'esecuzione specifica:

```
aws logs filter-log-events \
  --log-group-name /aws/lambda/myDurableFunction \
  --filter-pattern "exec-abc123"
```

Eseguire la pulizia delle risorse

Elimina la tua funzione durevole e le risorse associate:

```
# 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::ACCOUNT_ID:policy/durable-checkpoint-policy

aws iam delete-policy \
  --policy-arn arn:aws:iam::ACCOUNT_ID:policy/durable-checkpoint-policy

# Delete the role
```

```
aws iam delete-role --role-name durable-function-role
```

Fasi successive

Dopo aver implementato la funzione duratura con: AWS CLI

- Monitora le esecuzioni utilizzando i `list-durable-executions` comandi and `get-durable-execution`
- Visualizza le operazioni dei checkpoint negli eventi relativi ai dati AWS CloudTrail
- Imposta CloudWatch allarmi per errori di esecuzione o esecuzioni di lunga durata
- Automatizza le distribuzioni utilizzando script o pipeline di shell CI/CD

Per ulteriori informazioni sui AWS CLI comandi per Lambda, consulta il AWS CLI Command Reference.

Implementa le funzioni durevoli di Lambda con Infrastructure as Code

Puoi implementare funzioni durevoli Lambda utilizzando strumenti Infrastructure as Code (IaC) AWS CloudFormation come AWS CDK, o. AWS Serverless Application Model Questi strumenti consentono di definire la funzione, il ruolo di esecuzione e le autorizzazioni nel codice, rendendo le distribuzioni ripetibili e controllate dalla versione.

Tutti e tre gli strumenti richiedono di:

- Abilitare l'esecuzione duratura della funzione
- Concedi i permessi di checkpoint al ruolo di esecuzione
- Pubblica una versione o crea un alias (le funzioni durevoli richiedono competenze qualificate) ARNs

AWS CloudFormation

CloudFormation Utilizzatelo per definire la vostra funzione duratura in un modello. L'esempio seguente crea una funzione durevole con le autorizzazioni richieste.

```
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/AWSLambdaBasicExecutionRole
      Policies:
        - PolicyName: DurableExecutionPolicy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action:
                  - lambda:CheckpointDurableExecutions
                  - lambda:GetDurableExecutionState
                Resource: !GetAtt DurableFunction.Arn

  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: 10
        RetentionPeriodInDays: 1

  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
```

Per distribuire il modello

```
aws cloudformation deploy \
  --template-file template.yaml \
  --stack-name my-durable-function-stack \
  --capabilities CAPABILITY_IAM
```

AWS CDK

AWS CDK consente di definire l'infrastruttura utilizzando linguaggi di programmazione. I seguenti esempi mostrano come creare una funzione durevole usando TypeScript e 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) },
    });

    // Add checkpoint permissions
    durableFunction.addToRolePolicy(new iam.PolicyStatement({
      actions: [
        'lambda:CheckpointDurableExecutions',
        'lambda:GetDurableExecutionState',
      ],
      resources: [durableFunction.functionArn],
    }));

    // 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 checkpoint permissions
        durable_function.add_to_role_policy(iam.PolicyStatement(
            actions=[
                'lambda:CheckpointDurableExecutions',
                'lambda:GetDurableExecutionState',
            ],
            resources=[durable_function.function_arn]
        ))

        # 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'
        )
```

Per distribuire lo stack CDK

```
cdk deploy
```

AWS Serverless Application Model

AWS SAM semplifica i CloudFormation modelli per le applicazioni serverless. Il modello seguente crea una funzione durevole con. 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: 10
        RetentionPeriodInDays: 1
      Policies:
        - Version: '2012-10-17'
          Statement:
            - Effect: Allow
              Action:
                - lambda:CheckpointDurableExecutions
                - lambda:GetDurableExecutionState
              Resource: !Sub 'arn:aws:lambda:${AWS::Region}:${AWS::AccountId}:function:
${DurableFunction}'
      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
```

Per distribuire il modello SAM

```
sam build
```

```
sam deploy --guided
```

Modelli di configurazione comuni

Indipendentemente dallo strumento IaC utilizzato, segui questi schemi per funzioni durevoli:

Abilita un'esecuzione duratura

Impostare la proprietà `DurableExecution.Enabled` su `true`. Questa proprietà è disponibile solo durante la creazione della funzione, non è possibile abilitare l'esecuzione durevole su funzioni esistenti.

Concedi le autorizzazioni per il checkpoint

Aggiungi `lambda:CheckpointDurableExecutions` e `lambda:GetDurableExecutionState` al ruolo di esecuzione. Ambita queste autorizzazioni alla funzione specifica ARN.

Usa qualificati ARNs

Crea una versione o un alias per la tua funzione. Le funzioni durevoli richiedono una ARNs chiamata qualificata (con versione o alias). Utilizza AWS SAM o crea versioni esplicite `AutoPublishAlias` in e. CloudFormation AWS CDK

Dipendenze dei pacchetti

Includi l'SDK di esecuzione durevole nel tuo pacchetto di distribuzione. Per Node.js, installa`@aws/durable-execution-sdk-js`. Per Python, installa. `aws-durable-execution-sdk-python`

Fasi successive

Dopo aver implementato la tua funzione duratura:

- Verifica la tua funzione utilizzando l'ARN qualificato (versione o alias)
- Monitora l'avanzamento dell'esecuzione nella console Lambda nella scheda Esecuzioni durevoli
- Visualizza le operazioni dei checkpoint negli eventi relativi ai dati AWS CloudTrail
- Consulta CloudWatch i registri per verificare l'output delle funzioni e il comportamento di riproduzione

Per ulteriori informazioni sulla distribuzione delle funzioni Lambda con gli strumenti IAc, vedere:

- CloudFormation AWS::Lambda::Function riferimento
- AWS CDK Documentazione del modulo Lambda
- AWS SAM Guida per gli sviluppatori

Esempi e casi d'uso

Le funzioni durevoli Lambda consentono di creare applicazioni con tolleranza ai guasti e in più fasi utilizzando operazioni durevoli come passaggi e attese. Grazie al checkpoint automatico e al modello checkpoint-replay, in cui l'esecuzione riparte dall'inizio dopo un errore ma salta i checkpoint completati, le funzioni possono riprendersi dagli errori e riprendere l'esecuzione senza perdere i progressi.

Processi tolleranti ai guasti di breve durata

Utilizza funzioni durevoli per creare operazioni affidabili che in genere vengono completate in pochi minuti. Sebbene questi processi siano più brevi rispetto ai flussi di lavoro a lunga durata, traggono comunque vantaggio dal checkpoint automatico e dalla tolleranza ai guasti nei sistemi distribuiti. Le funzioni durevoli garantiscono il corretto completamento dei processi in più fasi anche quando le singole chiamate di servizio falliscono, senza richiedere una complessa gestione degli errori o un codice di gestione dello stato.

Gli scenari più comuni includono i sistemi di prenotazione alberghiera, le piattaforme di prenotazione di ristoranti, le richieste di viaggi in ride-sharing, l'acquisto di biglietti per eventi e gli upgrade degli abbonamenti SaaS. Questi scenari condividono caratteristiche comuni: più chiamate di servizio che devono essere completate contemporaneamente, la necessità di riprovare automaticamente in caso di guasti transitori e l'esigenza di mantenere lo stato coerente tra i sistemi distribuiti.

Transazioni distribuite tra microservizi

Coordina i pagamenti, l'inventario e la spedizione su più servizi con ripristino automatico in caso di guasti. Ogni operazione di servizio è conclusa in una fase, in modo da garantire che la transazione possa essere ripristinata da qualsiasi momento in caso di guasto del servizio.

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 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}
```

Se un passaggio fallisce, la funzione riprova automaticamente dall'ultimo checkpoint riuscito. La prenotazione dell'inventario persiste anche se l'elaborazione del pagamento fallisce temporaneamente. Quando la funzione riprova, salta la fase di inventario completata e procede direttamente all'elaborazione del pagamento. In questo modo si eliminano le prenotazioni duplicate e si garantisce lo stato coerente in tutto il sistema distribuito.

Elaborazione degli ordini con più passaggi

Elabora gli ordini tramite convalida, autorizzazione al pagamento, allocazione dell'inventario ed evasione con tentativi e ripristino automatici. Ogni passaggio è controllato, in modo da garantire che l'ordine proceda anche se i singoli passaggi falliscono e si riprova.

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 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']
    }
```

Questo schema assicura che gli ordini non rimangano mai bloccati in stati intermedi. Se la convalida fallisce, l'ordine viene rifiutato prima dell'autorizzazione del pagamento. Se l'autorizzazione al pagamento fallisce, l'inventario non viene allocato. Ogni passaggio si basa su quello precedente con riprova e ripristino automatici.

Nota

Il controllo condizionale `if (!validation.itemsValid)` non rientra in una fase e verrà rieseguito durante la riproduzione. Questo è sicuro perché è deterministico: produce sempre lo stesso risultato con lo stesso oggetto di convalida.

Processi a esecuzione prolungata

Utilizza funzioni durevoli per processi che durano ore, giorni o settimane. Le operazioni di attesa sospendono l'esecuzione senza incorrere in costi di elaborazione, rendendo convenienti i processi di lunga durata. Durante i periodi di attesa, la funzione smette di funzionare e Lambda ricicla l'ambiente di esecuzione. Quando è il momento di riprendere, Lambda richiama nuovamente la funzione e la riproduce dall'ultimo checkpoint.

Questo modello di esecuzione rende le funzioni durevoli ideali per i processi che devono essere sospesi per lunghi periodi, che si tratti di attese di decisioni umane, risposte esterne del sistema, finestre di elaborazione pianificate o ritardi basati sul tempo. Paghi solo per il tempo di elaborazione attivo, non per l'attesa.

Gli scenari più comuni includono i processi di approvazione dei documenti, l'elaborazione programmata in batch, i processi di onboarding per più giorni, i processi di prova in abbonamento e i sistemi di notifica ritardata. Questi scenari condividono caratteristiche comuni: lunghi periodi di attesa misurati in ore o giorni, la necessità di mantenere lo stato di esecuzione per tutte le attese e requisiti attenti ai costi in base ai quali è proibitivo pagare per i tempi di elaborazione inattivi.

Human-in-the-loop approvazioni

Sospendi l'esecuzione per le revisioni, le approvazioni o le decisioni dei documenti mantenendo lo stato di esecuzione. La funzione attende i callback esterni senza consumare risorse e riprende automaticamente quando viene ricevuta l'approvazione.

Questo modello è essenziale per i processi che richiedono il giudizio umano o la convalida esterna. La funzione si sospende al punto di callback e non comporta alcun addebito di calcolo durante l'attesa. Quando qualcuno invia la propria decisione tramite API, Lambda richiama nuovamente la funzione e la riproduce dal checkpoint, continuando con il risultato dell'approvazione.

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 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
    }
```

Quando la richiamata viene ricevuta e la funzione riprende, viene riprodotta dall'inizio. La fase di preparazione del documento restituisce immediatamente il risultato del checkpoint. Inoltre, l' waitForCallback operazione ritorna istantaneamente con il risultato dell'approvazione memorizzato anziché attendere nuovamente. L'esecuzione prosegue quindi fino alla fase di finalizzazione o archiviazione.

Pipeline di dati a più stadi

Elabora set di dati di grandi dimensioni attraverso fasi di estrazione, trasformazione e caricamento con punti di controllo tra le fasi. Il completamento di ogni fase può richiedere ore e i checkpoint assicurano che la pipeline possa riprendere da qualsiasi fase in caso di interruzione.

Questo modello è ideale per flussi di lavoro ETL, migrazioni di dati o processi di elaborazione in batch in cui è necessario elaborare i dati in fasi con punti di ripristino tra di loro. Se una fase fallisce, la pipeline riprende dall'ultima fase completata anziché ricominciare dall'inizio. È inoltre possibile utilizzare le operazioni di attesa per effettuare una pausa tra le fasi, rispettare i limiti di velocità, attendere che i sistemi a valle siano pronti o pianificare l'elaborazione durante le ore non di punta.

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 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(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']
    }
```

Ogni fase è racchiusa in una fase, che crea un checkpoint che consente alla pipeline di riprendere da qualsiasi fase in caso di interruzione. L'attesa di 5 minuti tra l'estrazione e la trasformazione rispetta i limiti di velocità del sistema di origine senza consumare risorse di elaborazione, mentre l'attesa fino alle 2 del mattino pianifica le costose operazioni di caricamento durante le ore non di punta.

Nota

La `new Date()` chiamata e la `calculateMsUntilHour()` funzione non rientrano nelle fasi iniziali e verranno rieseguite durante la riproduzione. Per le operazioni basate sul tempo che devono essere coerenti tra i replay, calcola il timestamp all'interno di un passaggio o utilizzalo solo per le durate di attesa (che sono punti di controllo).

Richiamazioni concatenate tra le funzioni

Richiama altre funzioni Lambda dall'interno di una funzione durevole utilizzando. `context.invoke()` La funzione chiamante si sospende in attesa del completamento della funzione richiamata, creando un checkpoint che preserva il risultato. Se la funzione chiamante viene interrotta dopo il completamento della funzione richiamata, riprende con il risultato memorizzato senza richiamare nuovamente la funzione.

Utilizzate questo modello quando avete funzioni specializzate che gestiscono domini specifici (convalida dei clienti, elaborazione dei pagamenti, gestione dell'inventario) e avete bisogno di coordinarle in un flusso di lavoro. Ogni funzione mantiene la propria logica e può essere richiamata da più funzioni di orchestrazione, evitando la duplicazione del codice.

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']
```

```
}
```

Ogni chiamata crea un checkpoint nella funzione orchestrator. Se l'orchestrator viene interrotto dopo il completamento della convalida del cliente, riprende da quel checkpoint con i dati del cliente memorizzati, saltando la chiamata di convalida. In questo modo si evitano chiamate duplicate ai servizi downstream e si garantisce un'esecuzione coerente anche dopo le interruzioni.

Le funzioni richiamate possono essere funzioni Lambda durevoli o standard. Se si richiama una funzione durevole, questa può avere un proprio flusso di lavoro in più fasi con attese e checkpoint. L'orchestratore attende semplicemente che l'esecuzione completa e duratura finisca, ricevendo il risultato finale.

> **Note**
>
> Le chiamate tra account non sono supportate. Tutte le funzioni richiamate devono trovarsi nello stesso AWS account della funzione chiamante.

Schemi avanzati

Usa funzioni durevoli per creare applicazioni complesse in più fasi che combinano più operazioni durevoli, esecuzione parallela, elaborazione di array, logica condizionale e polling. Questi modelli consentono di creare applicazioni sofisticate che coordinano molte attività mantenendo la tolleranza agli errori e il ripristino automatico.

I pattern avanzati vanno oltre i semplici passaggi sequenziali. È possibile eseguire operazioni contemporaneamente con`parallel()`, elaborare array con`map()`, attendere condizioni esterne e combinare queste primitive per creare applicazioni affidabili. `waitForCondition()` Ogni operazione durevole crea i propri checkpoint, in modo che l'applicazione possa essere ripristinata da qualsiasi punto in caso di interruzione.

Processi di onboarding degli utenti

Guida gli utenti attraverso la registrazione, la verifica delle e-mail, la configurazione del profilo e la configurazione iniziale con la gestione dei tentativi. Questo esempio combina passaggi sequenziali, callback e logica condizionale per creare un processo di onboarding completo.

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 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()
    }
```

Il processo combina passaggi sequenziali con punti di controllo per la creazione dell'account e l'invio di e-mail, quindi si interrompe per un massimo di 48 ore in attesa della verifica dell'e-mail senza consumare risorse. La logica condizionale gestisce percorsi diversi a seconda che la verifica sia completata o scaduta. Le attività di configurazione del profilo vengono eseguite contemporaneamente utilizzando operazioni parallele per ridurre il tempo totale di esecuzione e ogni passaggio riprova automaticamente in caso di errori transitori per garantire che l'onboarding venga completato in modo affidabile.

Elaborazione in batch con checkpoint

Elabora milioni di record con ripristino automatico dall'ultimo checkpoint riuscito dopo gli errori. Questo esempio dimostra come le funzioni durevoli combinino le `map()` operazioni con la suddivisione in blocchi e la limitazione della velocità per gestire l'elaborazione di dati su larga scala.

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 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
```

I record vengono suddivisi in batch gestibili per evitare di sovraccaricare la memoria o i servizi downstream, quindi vengono elaborati più batch contemporaneamente con il controllo del parallelismo. `maxConcurrency` Ogni batch ha il proprio checkpoint, quindi in caso di errore si limita a riprovare a rielaborare il batch fallito anziché rielaborare tutti i record. Questo modello è ideale per lavori ETL, migrazioni di dati o operazioni di massa in cui l'elaborazione può richiedere ore.

Fasi successive

- Esplora i concetti di base da comprendere DurableContext, i passaggi e le attese
- Esamina le best practice per la scrittura di codice deterministico e l'ottimizzazione delle prestazioni
- Scopri come testare funzioni durevoli a livello locale e nel cloud

Sicurezza e autorizzazioni per le funzioni durevoli di Lambda

Le funzioni durevoli Lambda richiedono autorizzazioni IAM specifiche per gestire le operazioni di checkpoint. Segui il principio del privilegio minimo concedendo solo le autorizzazioni necessarie alla tua funzione.

Autorizzazioni del ruolo di esecuzione

Il ruolo di esecuzione della tua funzione duratura richiede le autorizzazioni per creare checkpoint e recuperare lo stato di esecuzione. La seguente politica mostra le autorizzazioni minime richieste:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:CheckpointDurableExecution",
                "lambda:GetDurableExecutionState"
            ],
```

```
            "Resource": "arn:aws:lambda:region:account-id:function:function-name:*"
        }
    ]
}
```

Quando crei una funzione durevole utilizzando la console, Lambda aggiunge automaticamente queste autorizzazioni al ruolo di esecuzione. Se crei la funzione utilizzando AWS CLI o AWS CloudFormation, aggiungi queste autorizzazioni al tuo ruolo di esecuzione.

Principio del privilegio minimo

Inquadra l'`Resource`elemento in base a una funzione specifica ARNs invece di utilizzare caratteri jolly. Ciò limita il ruolo di esecuzione alle operazioni di checkpoint solo per le funzioni che le richiedono.

Esempio: autorizzazioni mirate per più funzioni

```
{
    "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:*"
            ]
        }
    ]
}
```

In alternativa, puoi utilizzare la policy AWS gestita `AWSLambdaBasicDurableExecutionRolePolicy` che include le autorizzazioni di esecuzione durature richieste insieme alle autorizzazioni di esecuzione Lambda di base per CloudWatch Amazon Logs.

Crittografia statale

Le funzioni durevoli Lambda abilitano automaticamente la crittografia a riposo utilizzando chiavi AWS di proprietà senza alcun costo. Ogni esecuzione di funzione mantiene uno stato isolato a cui altre esecuzioni non possono accedere. Le chiavi gestite dal cliente (CMK) non sono supportate.

I dati di Checkpoint includono:

- Risultati dei passaggi e valori restituiti
- Avanzamento e tempistica dell'esecuzione
- Informazioni sullo stato di attesa

Tutti i dati vengono crittografati in transito utilizzando TLS quando Lambda legge o scrive i dati del checkpoint.

Crittografia personalizzata con serializzatori e deserializzatori personalizzati

Per requisiti di sicurezza critici, puoi implementare il tuo meccanismo di crittografia e decrittografia utilizzando serializzatori e deserializzatori personalizzati () utilizzando un SDK durevole. SerDer Questo approccio offre il pieno controllo sulle chiavi e sugli algoritmi di crittografia utilizzati per proteggere i dati dei checkpoint.

> **Important**
>
> Quando utilizzi la crittografia personalizzata, perdi la visibilità dei risultati delle operazioni nella console Lambda e nelle risposte dell'API. I dati dei checkpoint appaiono crittografati nella cronologia di esecuzione e non possono essere ispezionati senza decriptarli.

Il ruolo di esecuzione della funzione richiede `kms:Encrypt` e `kms:Decrypt` autorizzazioni per la AWS KMS chiave utilizzata nell'implementazione personalizzata. SerDer

CloudTrail registrazione

Lambda registra le operazioni dei checkpoint come eventi relativi ai dati. AWS CloudTrailÈ possibile utilizzarlo CloudTrail per verificare quando vengono creati i checkpoint, tenere traccia delle modifiche dello stato di esecuzione e monitorare l'accesso a dati di esecuzione durevoli.

Le operazioni di checkpoint vengono visualizzate nei CloudTrail registri con i seguenti nomi di eventi:

- `CheckpointDurableExecution`- Registrato quando viene completato un passaggio e crea un checkpoint
- `GetDurableExecutionState`- Registrato quando Lambda recupera lo stato di esecuzione durante la riproduzione

Per abilitare la registrazione degli eventi relativi ai dati per funzioni durevoli, configura un CloudTrail percorso per registrare gli eventi relativi ai dati Lambda. Per ulteriori informazioni, consulta Registrazione degli eventi relativi ai dati nella Guida per l' CloudTrail utente.

Esempio: immissione del CloudTrail registro per il funzionamento del checkpoint

```
{
    "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"
}
```

Considerazioni relative a più account

Se richiamate funzioni durevoli su più AWS account, l'account chiamante necessita dell'`lambda:InvokeFunction`autorizzazione, ma le operazioni di checkpoint utilizzano sempre il ruolo di esecuzione nell'account della funzione. L'account chiamante non può accedere direttamente ai dati del checkpoint o allo stato di esecuzione.

Questo isolamento garantisce che i dati del checkpoint rimangano sicuri all'interno dell'account della funzione, anche se richiamati da account esterni.

Funzionalità di sicurezza Lambda ereditate

Le funzioni durevoli ereditano tutte le funzionalità di sicurezza, governance e conformità di Lambda, tra cui connettività VPC, crittografia delle variabili di ambiente, code di lettere morte, concorrenza riservata, URLs funzione, firma del codice e certificazioni di conformità (SOC, PCI DSS, HIPAA, ecc.).

Per informazioni dettagliate sulle funzionalità di sicurezza Lambda, consulta Security AWS Lambda nella Lambda Developer Guide. Le uniche considerazioni aggiuntive sulla sicurezza per le funzioni durevoli sono le autorizzazioni ai checkpoint documentate in questa guida.

SDK di esecuzione durevole

L'SDK di esecuzione durevole è la base per la creazione di funzioni durevoli. Fornisce le primitive necessarie per controllare i progressi, gestire i nuovi tentativi e gestire il flusso di esecuzione. L'SDK elimina la complessità della gestione e della riproduzione dei checkpoint, consentendoti di scrivere codice sequenziale che diventa automaticamente tollerante ai guasti.

L'SDK è disponibile per JavaScript TypeScript, e Python. Per la documentazione completa e gli esempi relativi alle API, consulta JavaScript/TypeScript SDK e Python SDK on. GitHub

DurableContext

L'SDK fornisce alla funzione un `DurableContext` oggetto che espone tutte le operazioni durevoli. Questo contesto sostituisce il contesto Lambda standard e fornisce metodi per creare checkpoint, gestire il flusso di esecuzione e coordinarsi con sistemi esterni.

Per utilizzare l'SDK, avvolgi il tuo gestore Lambda con il wrapper di esecuzione durevole:

TypeScript

```
import { withDurableExecution, DurableContext } from '@aws/durable-execution-sdk-
js';

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    // Your function receives DurableContext instead of Lambda context
    // Use context.step(), context.wait(), etc.
    return result;
  }
);
```

Python

```
from aws_durable_execution_sdk_python import durable_execution, DurableContext

@durable_execution
def handler(event: dict, context: DurableContext):
    # Your function receives DurableContext
    # Use context.step(), context.wait(), etc.
    return result
```

Il wrapper intercetta l'invocazione della funzione, carica qualsiasi registro di checkpoint esistente e fornisce il file che gestisce la riproduzione e il checkpoint. `DurableContext`

Cosa fa l'SDK

L'SDK gestisce tre responsabilità fondamentali che consentono un'esecuzione duratura:

Gestione dei checkpoint: l'SDK crea automaticamente i checkpoint man mano che la funzione esegue operazioni durevoli. Ogni checkpoint registra il tipo di operazione, gli input e i risultati. Quando la funzione completa un passaggio, l'SDK mantiene il checkpoint prima di continuare. Ciò garantisce che la funzione possa riprendere dopo qualsiasi operazione completata in caso di interruzione.

Coordinamento della riproduzione: quando la funzione riprende dopo una pausa o un'interruzione, l'SDK esegue la riproduzione. Esegue il codice dall'inizio ma salta le operazioni completate, utilizzando i risultati dei checkpoint memorizzati invece di rieseguirli. L'SDK garantisce che la riproduzione sia deterministica: con gli stessi input e lo stesso registro dei checkpoint, la funzione produce gli stessi risultati.

Isolamento dello stato: l'SDK mantiene lo stato di esecuzione separatamente dalla logica aziendale. Ogni esecuzione durevole ha il proprio registro dei checkpoint a cui le altre esecuzioni non possono accedere. L'SDK crittografa i dati dei checkpoint inattivi e garantisce che lo stato rimanga coerente tra i replay.

Come funziona il checkpoint

Quando si chiama un'operazione duratura, l'SDK segue questa sequenza:

1. Verifica la presenza di un checkpoint esistente: l'SDK verifica se questa operazione è già stata completata in una precedente chiamata. Se esiste un checkpoint, l'SDK restituisce il risultato memorizzato senza rieseguire l'operazione.

2. Esegui l'operazione: se non esiste alcun checkpoint, l'SDK esegue il codice operativo. Per quanto riguarda i passaggi, ciò significa chiamare la funzione. Per le attese, ciò significa la ripresa della pianificazione.
3. Crea checkpoint: una volta completata l'operazione, l'SDK serializza il risultato e crea un checkpoint. Il checkpoint include il tipo di operazione, il nome, gli input, il risultato e il timestamp.
4. Checkpoint persistente: l'SDK chiama l'API checkpoint Lambda per rendere persistente il checkpoint. Ciò garantisce che il checkpoint sia duraturo prima di continuare l'esecuzione.
5. Risultato di ritorno: l'SDK restituisce il risultato dell'operazione al codice, che continua con l'operazione successiva.

Questa sequenza assicura che, una volta completata un'operazione, il risultato venga archiviato in modo sicuro. Se la funzione viene interrotta in qualsiasi momento, l'SDK può riprodurla fino all'ultimo checkpoint completato.

Comportamento di riproduzione

Quando la funzione riprende dopo una pausa o un'interruzione, l'SDK esegue la riproduzione:

1. Carica il registro del checkpoint: l'SDK recupera il registro dei checkpoint per questa esecuzione da Lambda.
2. Esegui dall'inizio: l'SDK richiama la funzione di gestione dall'inizio, non dal punto in cui è stata messa in pausa.
3. Ignora le operazioni durevoli completate: man mano che il codice richiama operazioni durevoli, l'SDK confronta ciascuna di esse con il registro del checkpoint. Per operazioni durevoli completate, l'SDK restituisce il risultato memorizzato senza eseguire il codice operativo.

> **Note**
>
> Se il risultato di un contesto secondario era maggiore della dimensione massima del checkpoint (256 KB), il codice del contesto viene eseguito nuovamente durante la riproduzione. Ciò consente di ottenere risultati di grandi dimensioni a partire dalle operazioni durevoli eseguite all'interno del contesto, che verranno ricercate dal registro del checkpoint. Pertanto è imperativo eseguire codice deterministico solo nel contesto stesso. Quando si utilizzano contesti secondari con risultati di grandi dimensioni, è consigliabile eseguire un lavoro di lunga durata o non deterministico all'interno di passaggi ed eseguire solo attività di breve durata che combinano i risultati nel contesto stesso.

4. Riprendi al punto di interruzione: quando l'SDK raggiunge un'operazione senza checkpoint, viene eseguita normalmente e crea nuovi checkpoint man mano che le operazioni durevoli vengono completate.

Questo meccanismo di replay richiede che il codice sia deterministico. Con gli stessi input e lo stesso registro dei checkpoint, la funzione deve effettuare la stessa sequenza di chiamate operative durevoli. L'SDK lo impone convalidando che i nomi e i tipi di operazioni corrispondano al registro del checkpoint durante la riproduzione.

Operazioni durevoli disponibili

`DurableContext`Fornisce operazioni per diversi modelli di coordinamento. Ogni operazione durevole crea automaticamente dei checkpoint, assicurando che la funzione possa riprendere da qualsiasi momento.

Fasi

Esegue la logica aziendale con checkpoint e nuovi tentativi automatici. Utilizza i passaggi per le operazioni che richiamano servizi esterni, eseguono calcoli o eseguono qualsiasi logica che dovrebbe essere sottoposta a checkpoint. L'SDK crea un checkpoint prima e dopo il passaggio, memorizzando il risultato per la riproduzione.

TypeScript

```
const result = await context.step('process-payment', async () => {
  return await paymentService.charge(amount);
});
```

Python

```
result = context.step(
    lambda _: payment_service.charge(amount),
    name='process-payment'
)
```

I passaggi supportano strategie di ripetizione configurabili, semantica di esecuzione (at-most-once or) e serializzazione personalizzata. at-least-once

Stati di attesa

Sospende l'esecuzione per una durata specificata senza consumare risorse di elaborazione. L'SDK crea un checkpoint, termina l'invocazione della funzione e pianifica la ripresa. Al termine dell'attesa, Lambda richiama nuovamente la funzione e l'SDK viene riprodotto fino al punto di attesa prima di continuare.

TypeScript

```
// Wait 1 hour without charges
await context.wait({ seconds: 3600 });
```

Python

```
# Wait 1 hour without charges
context.wait(3600)
```

Callback

I callback consentono alla funzione di mettere in pausa e attendere che sistemi esterni forniscano input. Quando si crea un callback, l'SDK genera un ID di callback univoco e crea un checkpoint. La funzione viene quindi sospesa (termina la chiamata) senza incorrere in costi di calcolo. I sistemi esterni inviano i risultati del callback utilizzando `SendDurableExecutionCallbackSuccess` o `SendDurableExecutionCallbackFailure` Lambda APIs. Quando viene inviata una callback, Lambda richiama nuovamente la funzione, l'SDK viene riprodotto fino al punto di callback e la funzione continua con il risultato del callback.

L'SDK fornisce due metodi per lavorare con i callback:

CreateCallback: crea un callback e restituisce sia una promessa che un ID di callback. L'ID di callback viene inviato a un sistema esterno, che invia il risultato utilizzando l'API Lambda.

TypeScript

```
const [promise, callbackId] = await context.createCallback('approval', {
  timeout: { hours: 24 }
});

await sendApprovalRequest(callbackId, requestData);
const approval = await promise;
```

Python

```
callback = context.create_callback(
    name='approval',
    config=CallbackConfig(timeout_seconds=86400)
)

context.step(
    lambda _: send_approval_request(callback.callback_id),
    name='send_request'
)

approval = callback.result()
```

waitForCallback: semplifica la gestione delle callback combinando la creazione e l'invio delle callback in un'unica operazione. L'SDK crea il callback, esegue la funzione di invio con l'ID di callback e attende il risultato.

TypeScript

```
const result = await context.waitForCallback(
  'external-api',
  async (callbackId, ctx) => {
    await submitToExternalAPI(callbackId, requestData);
  },
  { timeout: { minutes: 30 } }
);
```

Python

```
result = context.wait_for_callback(
    lambda callback_id: submit_to_external_api(callback_id, request_data),
    name='external-api',
    config=WaitForCallbackConfig(timeout_seconds=1800)
)
```

Configura i timeout per evitare che le funzioni restino in attesa all'infinito. Se scade un callback, l'SDK genera un timeout `CallbackError` e la tua funzione può gestire il caso di timeout. Utilizza i timeout del battito cardiaco per le richiamate di lunga durata per rilevare quando i sistemi esterni smettono di rispondere.

Utilizza i callback per human-in-the-loop flussi di lavoro, integrazione di sistemi esterni, risposte webhook o qualsiasi scenario in cui l'esecuzione debba essere messa in pausa per l'input esterno.

Esecuzione parallela

Esegue più operazioni contemporaneamente con il controllo della concorrenza opzionale. L'SDK gestisce l'esecuzione parallela, crea checkpoint per ogni operazione e gestisce gli errori in base alla politica di completamento.

TypeScript

```
const results = await context.parallel([
  async (ctx) => ctx.step('task1', async () => processTask1()),
  async (ctx) => ctx.step('task2', async () => processTask2()),
  async (ctx) => ctx.step('task3', async () => processTask3())
]);
```

Python

```
results = context.parallel(
```

```
    lambda ctx: ctx.step(lambda _: process_task1(), name='task1'),
    lambda ctx: ctx.step(lambda _: process_task2(), name='task2'),
    lambda ctx: ctx.step(lambda _: process_task3(), name='task3')
)
```

Utilizzato `parallel` per eseguire operazioni indipendenti contemporaneamente.

Eseguire la mappatura

Esegue contemporaneamente un'operazione su ogni elemento di una matrice con controllo di concorrenza opzionale. L'SDK gestisce l'esecuzione simultanea, crea checkpoint per ogni operazione e gestisce gli errori in base alla politica di completamento.

TypeScript

```
const results = await context.map(itemArray, async (ctx, item, index) =>
  ctx.step('task', async () => processItem(item, index))
);
```

Python

```
results = context.map(
    item_array,
    lambda ctx, item, index: ctx.step(
        lambda _: process_item(item, index),
        name='task'
    )
)
```

Utilizzato `map` per elaborare gli array con controllo della concorrenza.

Contesti infantili

Crea un contesto di esecuzione isolato per le operazioni di raggruppamento. I contesti secondari dispongono di un proprio registro dei checkpoint e possono contenere più passaggi, attese e

altre operazioni. L'SDK tratta l'intero contesto secondario come una singola unità per riprovare e ripristinare.

Utilizza i contesti secondari per organizzare flussi di lavoro complessi, implementare flussi di lavoro secondari o isolare operazioni da riprovare contemporaneamente.

TypeScript

```
const result = await context.runInChildContext(
  'batch-processing',
  async (childCtx) => {
    return await processBatch(childCtx, items);
  }
);
```

Python

```
result = context.run_in_child_context(
    lambda child_ctx: process_batch(child_ctx, items),
    name='batch-processing'
)
```

Il meccanismo di replay richiede che le operazioni durevoli avvengano in un ordine deterministico. Utilizzando più contesti secondari è possibile eseguire contemporaneamente più flussi di lavoro e il determinismo si applica separatamente all'interno di ciascun contesto. Ciò consente di creare funzioni ad alte prestazioni che utilizzano in modo efficiente più core della CPU.

Ad esempio, immaginiamo di iniziare due contesti secondari, A e B. Nella chiamata iniziale, i passaggi all'interno dei contesti sono stati eseguiti in questo ordine, con i passaggi «A» eseguiti contemporaneamente ai passaggi «B»: A1, B1, B2, A2, A3. Dopo la riproduzione, i tempi sono molto più rapidi in quanto i risultati vengono recuperati dal registro dei checkpoint e i passaggi vengono eseguiti in un ordine diverso: B1, A1, A2, B2, A3. Poiché i passaggi «A» sono stati rilevati nell'ordine corretto (A1, A2, A3) e i passaggi «B» sono stati rilevati nell'ordine corretto (B1, B2), l'esigenza del determinismo è stata soddisfatta correttamente.

Attese condizionali

Sondaggi relativi a una condizione con checkpoint automatico tra un tentativo e l'altro. L'SDK esegue la funzione di controllo, crea un checkpoint con il risultato, attende in base alla strategia scelta e ripete l'operazione finché la condizione non viene soddisfatta.

TypeScript

```
const result = await context.waitForCondition(
  async (state, ctx) => {
    const status = await checkJobStatus(state.jobId);
    return { ...state, status };
  },
  {
    initialState: { jobId: 'job-123', status: 'pending' },
    waitStrategy: (state) =>
      state.status === 'completed'
        ? { shouldContinue: false }
        : { shouldContinue: true, delay: { seconds: 30 } }
  }
);
```

Python

```
result = context.wait_for_condition(
    lambda state, ctx: check_job_status(state['jobId']),
    config=WaitForConditionConfig(
        initial_state={'jobId': 'job-123', 'status': 'pending'},
        wait_strategy=lambda state, attempt:
            {'should_continue': False} if state['status'] == 'completed'
            else {'should_continue': True, 'delay': 30}
    )
)
```

Utilizzalo `waitForCondition` per eseguire il polling di sistemi esterni, attendere che le risorse siano pronte o implementare nuovi tentativi con backoff.

Invocazione della funzione

Richiama un'altra funzione Lambda e ne attende il risultato. L'SDK crea un checkpoint, richiama la funzione di destinazione e riprende la funzione al termine della chiamata. Ciò consente la composizione delle funzioni e la scomposizione del flusso di lavoro.

TypeScript

```
const result = await context.invoke(
  'invoke-processor',
  'arn:aws:lambda:us-east-1:123456789012:function:processor',
  { data: inputData }
);
```

Python

```
result = context.invoke(
    'arn:aws:lambda:us-east-1:123456789012:function:processor',
    {'data': input_data},
    name='invoke-processor'
)
```

Come viene misurata la durata delle operazioni

Ogni operazione duratura eseguita `DurableContext` crea punti di controllo per tenere traccia dell'avanzamento dell'esecuzione e archiviare i dati sullo stato. Queste operazioni comportano addebiti in base al loro utilizzo e i checkpoint possono contenere dati che contribuiscono ai costi di scrittura e conservazione dei dati. I dati archiviati includono i dati degli eventi di invocazione, i payload restituiti dai passaggi e i dati trasmessi durante il completamento dei callback. Comprendere come viene misurata la durata delle operazioni consente di stimare i costi di esecuzione e ottimizzare i flussi di lavoro. Per i dettagli sui prezzi, consulta la pagina dei prezzi di Lambda.

La dimensione del payload si riferisce alla dimensione dei dati serializzati necessari a garantire la persistenza di un'operazione durevole. I dati vengono misurati in byte e la dimensione può variare a seconda del serializzatore utilizzato dall'operazione. Il payload di un'operazione potrebbe essere il

risultato stesso del completamento con successo o l'oggetto di errore serializzato se l'operazione non è riuscita.

Operazioni di base

Le operazioni di base sono gli elementi costitutivi fondamentali per funzioni durevoli:

Operation	Tempistica dei checkpoint	Numero di operazioni	Dati persistenti
Esecuzione	Avviato	1	Dimensione del payload di input
Esecuzione	Completato () Succeeded/Failed/Stopped	0	Dimensione del payload in uscita
Fase	Retry/Succeeded/Failed	1 + Nessun tentativo	Dimensione del payload restituita da ogni tentativo
Attendi	Avviato	1	N/D
WaitForCondition	Ogni tentativo di sondaggio	1 + N sondaggi	Dimensione del payload restituita da ogni tentativo di sondaggio
Riprova a livello di chiamata	Avviato	1	Payload per l'oggetto di errore

Operazioni di callback

Le operazioni di callback consentono alla funzione di mettere in pausa e attendere che sistemi esterni forniscano input. Queste operazioni creano punti di controllo quando il callback viene creato e quando è completato:

Operation	Tempistica del checkpoint	Numero di operazioni	Dati persistenti
CreateCallback	Avviato	1	N/D
Completamento del callback tramite chiamata API	Completato	0	Payload di callback
WaitForCallback	Avviato	3+ N tentativi (contesto + callback + step)	Payload restituiti dai tentativi di invio, più due copie del payload di callback

Operazioni composte

Le operazioni composte combinano più operazioni durevoli per gestire modelli di coordinamento complessi come esecuzione parallela, elaborazione di array e contesti annidati:

Operation	tempistica dei checkpoint	Numero di operazioni	Dati persistenti
Parallela	Avviato	1+ N rami (1 contesto principale + N contesti secondari)	Fino a due copie della dimensione del payload restituita da ciascun ramo, più gli stati di ogni ramo

Operation	tempistica dei checkpoint	Numero di operazioni	Dati persistenti
Mappa	Avviato	1+ N rami (1 contesto genitore +N contesti secondari)	Fino a due copie della dimensione del payload restituita da ogni iterazione, più gli stati di ogni iterazione
Aiutanti di Promise	Completato	1	Dimensione del payload restituita dalla promessa
RunInChildContext	Riuscito/non riuscito	1	Dimensione del payload restituita dal contesto secondario

Per i contesti, ad esempio provenienti da `runInChildContext` o utilizzati internamente da operazioni composte, i risultati inferiori a 256 KB vengono controllati direttamente. I risultati più grandi non vengono archiviati, ma vengono ricostruiti durante la riproduzione rielaborando le operazioni del contesto.

Runtime supportati per funzioni durevoli

Sono disponibili funzioni durevoli per i runtime di Node.js e Python. Puoi creare funzioni durevoli utilizzando runtime gestiti nella console Lambda o distribuirle utilizzando immagini dei container per una maggiore flessibilità della versione di runtime.

Runtime gestiti da Lambda

I seguenti runtime gestiti supportano funzioni durevoli quando si creano funzioni nella console Lambda o si utilizza AWS CLI il parametro con `--durable-config '{"ExecutionTimeout": 10, "RetentionPeriodInDays":1}'` il parametro. Per informazioni complete sui runtime Lambda, consulta Lambda runtimes.

Lingua	Runtime
Node.js	nodejs 22.x
Node.js	nodejs24.x
Python	python 3.13
Python	python3.14

> **Note**
>
> I runtime Lambda includono l'SDK di esecuzione durevole per test e sviluppo. Tuttavia, consigliamo di includere l'SDK nel pacchetto di distribuzione per la produzione. Ciò garantisce la coerenza delle versioni ed evita potenziali aggiornamenti di runtime che potrebbero influire sul comportamento delle funzioni.

Node.js

Installa l'SDK nel tuo progetto Node.js:

```
npm install @aws/durable-execution-sdk-js
```

L'SDK supporta JavaScript e. TypeScript Per TypeScript i progetti, l'SDK include definizioni dei tipi.

Python

Installa l'SDK nel tuo progetto Python:

```
pip install aws-durable-execution-sdk-python
```

L'SDK Python utilizza metodi sincroni e non richiede. `async/await`

Immagini di container

È possibile utilizzare funzioni durevoli con immagini dei contenitori per supportare versioni di runtime aggiuntive o configurazioni di runtime personalizzate. Le immagini dei container consentono di utilizzare versioni di runtime non disponibili come runtime gestiti o di personalizzare l'ambiente di runtime.

Per creare una funzione durevole utilizzando un'immagine del contenitore:

1. Crea un Dockerfile basato su un'immagine di base Lambda
2. Installa l'SDK di esecuzione durevole nel tuo contenitore
3. Crea e invia l'immagine del contenitore ad Amazon Elastic Container Registry
4. Crea la funzione Lambda dall'immagine del contenitore con l'esecuzione durevole abilitata

Esempio di contenitore Python

Crea un Dockerfile per 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" ]
```

Crea un file: `requirements.txt`

```
aws-durable-execution-sdk-python
```

Crea e invia l'immagine:

```
# 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
```

Crea la funzione con l'esecuzione duratura abilitata:

```
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": 10, "RetentionPeriodInDays":1}'
```

Per ulteriori informazioni sull'utilizzo delle immagini dei container con Lambda, consulta Creazione di immagini di container Lambda nella Lambda Developer Guide.

Considerazioni sul runtime

Gestione delle versioni SDK: includi l'SDK di esecuzione durevole nel pacchetto di distribuzione o nell'immagine del contenitore. Ciò garantisce che la funzione utilizzi una versione SDK specifica e non sia influenzata dagli aggiornamenti di runtime. Aggiungi le versioni SDK a `package.json` o `requirements.txt` per controllare quando esegui l'aggiornamento.

Aggiornamenti in fase di esecuzione: AWS aggiorna i runtime gestiti per includere patch di sicurezza e correzioni di bug. Questi aggiornamenti possono includere nuove versioni dell'SDK. Per evitare comportamenti imprevisti, includi l'SDK nel pacchetto di distribuzione e testalo accuratamente prima di distribuirlo in produzione.

Dimensioni dell'immagine del contenitore: le immagini dei contenitori hanno una dimensione massima non compressa di 10 GB. L'SDK di esecuzione durevole aggiunge dimensioni minime all'immagine. Ottimizza il tuo contenitore utilizzando build in più fasi e rimuovendo le dipendenze non necessarie.

Prestazioni di avvio a freddo: le immagini dei container possono avere tempi di avvio a freddo più lunghi rispetto ai runtime gestiti. L'SDK di esecuzione durevole ha un impatto minimo sulle prestazioni di avvio a freddo. Utilizza la concorrenza fornita se la latenza di avvio a freddo è fondamentale per la tua applicazione.

Richiamo di funzioni Lambda durevoli

Le funzioni Lambda durevoli supportano gli stessi metodi di invocazione delle funzioni Lambda standard. È possibile richiamare funzioni durevoli in modo sincrono, asincrono o tramite mappature delle sorgenti degli eventi. Il processo di invocazione è identico alle funzioni standard, ma le funzioni durevoli forniscono funzionalità aggiuntive per le esecuzioni di lunga durata e la gestione automatica dello stato.

Metodi di invocazione

Invocazione sincrona: richiama una funzione durevole e attendi la risposta. Le chiamate sincrone sono limitate dalla Lambda a 15 minuti (o meno, a seconda della funzione configurata e del timeout di esecuzione). Utilizza la chiamata sincrona quando hai bisogno di risultati immediati o quando esegui l'integrazione con servizi che prevedono una risposta. APIs È possibile utilizzare le operazioni di attesa per un calcolo efficiente senza interrompere il chiamante: la chiamata attende il completamento dell'intera esecuzione durevole. Per gli avvii di esecuzione idempotenti, utilizzate il parametro del nome dell'esecuzione come descritto in Idempotency.

```
aws lambda invoke \
  --function-name my-durable-function:1 \
  --cli-binary-format raw-in-base64-out \
  --payload '{"orderId": "12345"}' \
  response.json
```

Richiamata asincrona: mette in coda un evento per l'elaborazione senza attendere una risposta. Lambda mette l'evento in coda e ritorna immediatamente. Le chiamate asincrone supportano durate di esecuzione fino a 1 anno. Utilizza la chiamata asincrona per scenari o quando l'elaborazione può

avvenire in background. fire-and-forget Per avvii di esecuzione idempotenti, utilizzate il parametro del nome dell'esecuzione come descritto in Idempotenza.

```
aws lambda invoke \
  --function-name my-durable-function:1 \
  --invocation-type Event \
  --cli-binary-format raw-in-base64-out \
  --payload '{"orderId": "12345"}' \
  response.json
```

Mappature delle sorgenti degli eventi: configura Lambda per richiamare automaticamente la tua funzione duratura quando i record sono disponibili da servizi basati su stream o code come Amazon SQS, Kinesis o DynamoDB. Le mappature delle sorgenti degli eventi analizzano l'origine dell'evento e richiamano la funzione con batch di record. Per i dettagli sull'utilizzo delle mappature delle sorgenti degli eventi con funzioni durevoli, inclusi i limiti di durata dell'esecuzione, consulta Mappature delle sorgenti degli eventi con funzioni durevoli.

Per dettagli completi su ciascun metodo di invocazione, vedere invocazione sincrona e chiamata asincrona.

> **Note**
>
> Le funzioni durevoli supportano dead-letter queues (DLQs) per la gestione degli errori, ma non supportano le destinazioni Lambda. Configura un DLQ per acquisire i record delle chiamate non riuscite.

ARNs Requisito qualificato

Le funzioni durevoli richiedono identificatori qualificati per la chiamata. È necessario richiamare funzioni durevoli utilizzando un numero di versione, un alias o. `$LATEST` È possibile utilizzare un ARN completo o un nome di funzione con version/alias suffisso. Non è possibile utilizzare un identificatore non qualificato (senza una versione o un suffisso di alias).

Invocazioni valide:

```
# 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
```

Invocazioni non valide:

```

# Unqualified ARN (not allowed)
arn:aws:lambda:us-east-1:123456789012:function:my-durable-function

# Unqualified function name (not allowed)
my-durable-function
```

Questo requisito garantisce che le esecuzioni durevoli rimangano coerenti per tutto il loro ciclo di vita. Quando inizia un'esecuzione durevole, viene associata alla versione della funzione specifica. Se la funzione viene messa in pausa e riprende ore o giorni dopo, Lambda richiama la stessa versione che ha avviato l'esecuzione, garantendo la coerenza del codice nell'intero flusso di lavoro.

Best practice

Utilizza versioni numerate o alias per funzioni durevoli di produzione anziché. `$LATEST` Le versioni numerate sono immutabili e garantiscono una riproduzione deterministica. Facoltativamente, gli alias forniscono un riferimento stabile che è possibile aggiornare in modo da puntare a nuove versioni senza modificare il codice di invocazione. Quando si aggiorna un alias, le nuove esecuzioni utilizzano la nuova versione, mentre le esecuzioni in corso continuano con la versione originale. È possibile utilizzarla `$LATEST` per la prototipazione o per abbreviare i tempi di implementazione durante lo sviluppo, tenendo presente che le esecuzioni potrebbero non essere riprodotte correttamente (o addirittura fallire) se il codice sottostante cambia durante l'esecuzione.

Comprendere il ciclo di vita dell'esecuzione

Quando richiami una funzione duratura, Lambda crea un'esecuzione durevole che può includere più chiamate di funzioni:

1. Richiamata iniziale: la richiesta di invocazione crea una nuova esecuzione durevole. Lambda assegna un ID di esecuzione univoco e avvia l'elaborazione.
2. Esecuzione e checkpoint: man mano che la funzione esegue operazioni durature, l'SDK crea checkpoint che tengono traccia dei progressi.
3. Sospensione (se necessaria): se la funzione utilizza attese durature, ad esempio or, `wait` o ripetizioni automatiche dei passaggi`waitForCallback`, Lambda sospende l'esecuzione e interrompe l'addebito del tempo di elaborazione.
4. Ripresa: quando è il momento di riprendere (anche dopo nuovi tentativi), Lambda richiama nuovamente la funzione. L'SDK riproduce il registro del checkpoint e continua dal punto in cui l'esecuzione si è interrotta.
5. Completamento: quando la funzione restituisce un risultato finale o genera un errore non gestito, l'esecuzione durevole viene completata.

Per le chiamate sincrone, il chiamante attende il completamento dell'intera esecuzione durevole, comprese le eventuali operazioni di attesa. Se l'esecuzione supera il timeout di chiamata (15 minuti o meno), la chiamata scade. Per le chiamate asincrone, Lambda ritorna immediatamente e l'esecuzione continua indipendentemente. Usa l'esecuzione durevole APIs per tenere traccia dello stato di esecuzione e recuperare i risultati finali.

Richiamo dal codice dell'applicazione

Utilizzate il AWS SDKs per richiamare funzioni durevoli dal codice dell'applicazione. Il processo di invocazione è identico alle funzioni standard:

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'})
)
```

Invocazioni concatenate

Le funzioni durevoli possono richiamare altre funzioni durevoli e non durevoli utilizzando l'operazione da. `invoke` `DurableContext` Questo crea una chiamata concatenata in cui la funzione chiamante attende (sospende) il completamento della funzione invocata:

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)}
```

Le invocazioni concatenate creano un checkpoint nella funzione chiamante. Se la funzione chiamante viene interrotta, riprende dal checkpoint con il risultato della funzione invocata, senza richiamare nuovamente la funzione.

> **Note**
>
> Le chiamate concatenate tra più account non sono supportate. La funzione richiamata deve trovarsi nello stesso AWS account della funzione chiamante.

Mappature delle sorgenti degli eventi con funzioni durevoli

Le funzioni durevoli funzionano con tutte le mappature delle sorgenti degli eventi Lambda. Configura le mappature delle sorgenti degli eventi per funzioni durevoli nello stesso modo in cui le configuri per le funzioni standard. Le mappature delle sorgenti degli eventi analizzano automaticamente sorgenti di eventi come Amazon SQS, Kinesis e DynamoDB Streams e richiamano la tua funzione con batch di record.

Le mappature delle sorgenti degli eventi sono utili per funzioni durevoli che elaborano flussi o code con flussi di lavoro complessi e in più fasi. Ad esempio, puoi creare una funzione durevole che elabora i messaggi Amazon SQS con nuovi tentativi, chiamate API esterne e approvazioni umane.

In che modo le mappature delle sorgenti degli eventi richiamano funzioni durevoli

Le mappature delle sorgenti degli eventi richiamano funzioni durevoli in modo sincrono, in attesa del completamento dell'esecuzione duratura completa prima di elaborare il batch successivo o di contrassegnare i record come elaborati. Se il tempo di esecuzione duraturo totale supera i 15 minuti, l'esecuzione scade e fallisce. La mappatura dell'origine degli eventi riceve un'eccezione di timeout e la gestisce in base alla configurazione dei nuovi tentativi.

Limite di esecuzione di 15 minuti

Quando le funzioni durevoli vengono richiamate dalle mappature delle sorgenti degli eventi, la durata totale dell'esecuzione durevole non può superare i 15 minuti. Questo limite si applica all'intera esecuzione durevole dall'inizio al completamento, non solo alle chiamate di singole funzioni.

Questo limite di 15 minuti è separato dal timeout della funzione Lambda (anch'esso massimo 15 minuti). Il timeout della funzione controlla per quanto tempo può essere eseguita ogni singola chiamata, mentre il timeout di esecuzione duraturo controlla il tempo totale trascorso dall'inizio dell'esecuzione al completamento.

Scenari di esempio:

- Valido: una funzione durevole elabora un messaggio Amazon SQS in tre passaggi, ciascuno dei quali richiede 2 minuti, quindi attende 5 minuti prima di completare un passaggio finale. Tempo di esecuzione totale: 11 minuti. Funziona perché il totale è inferiore a 15 minuti.
- Non valido: una funzione durevole elabora un messaggio Amazon SQS, completa l'elaborazione iniziale in 2 minuti, quindi attende 20 minuti per una richiamata esterna prima del completamento. Tempo di esecuzione totale: 22 minuti. Questa operazione supera il limite di 15 minuti e avrà esito negativo.
- Non valido: una funzione durevole elabora un record Kinesis con più operazioni di attesa per un totale di 30 minuti tra i passaggi. Anche se ogni singola chiamata viene completata rapidamente, il tempo di esecuzione totale supera i 15 minuti.

> **Important**
>
> Configura il timeout di esecuzione duraturo su 15 minuti o meno quando utilizzi le mappature delle sorgenti degli eventi, altrimenti la creazione della mappatura delle sorgenti degli eventi avrà esito negativo. Se il flusso di lavoro richiede tempi di esecuzione più lunghi, utilizza lo schema di funzioni intermedie descritto di seguito.

Configurazione delle mappature delle sorgenti degli eventi

Configura le mappature delle sorgenti degli eventi per funzioni durevoli utilizzando la console Lambda, oppure. AWS CLI AWS SDKs Tutte le proprietà standard di mappatura delle sorgenti degli eventi si applicano alle funzioni durevoli:

```
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
```

Ricordarsi di utilizzare un ARN qualificato (con numero di versione o alias) quando si configurano le mappature delle sorgenti degli eventi per funzioni durevoli.

Gestione degli errori con le mappature delle sorgenti degli eventi

Le mappature delle sorgenti degli eventi forniscono una gestione degli errori integrata che funziona con funzioni durevoli:

- Comportamento dei nuovi tentativi: se la chiamata iniziale fallisce, la mappatura delle sorgenti degli eventi riprova in base alla configurazione dei nuovi tentativi. Configura il numero massimo di tentativi e gli intervalli di nuovi tentativi in base alle tue esigenze.
- Code di lettere scadenti: configura una coda di lettere scadenti per acquisire i record che hanno esito negativo dopo tutti i nuovi tentativi. Ciò impedisce la perdita di messaggi e consente l'ispezione manuale dei record non riusciti.
- Errori parziali dei batch: per Amazon SQS e Kinesis, utilizza la segnalazione parziale degli errori dei batch per elaborare i record singolarmente e riprovare solo i record non riusciti.
- Bisect in caso di errore: per Kinesis e DynamoDB Streams, abilita bisect on error per dividere i batch non riusciti e isolare i record problematici.

> **Note**
>
> Le funzioni durevoli supportano dead-letter queues (DLQs) per la gestione degli errori, ma non supportano le destinazioni Lambda. Configura un DLQ per acquisire i record delle chiamate non riuscite.

Per informazioni complete sulla gestione degli errori di mappatura delle sorgenti degli eventi, consulta la sezione Mappature delle sorgenti degli eventi.

Utilizzo di una funzione intermedia per flussi di lavoro di lunga durata

Se il completamento del flusso di lavoro richiede più di 15 minuti, utilizzate una funzione Lambda standard intermedia tra la mappatura della sorgente dell'evento e la funzione duratura. La funzione intermediaria riceve gli eventi dalla mappatura delle sorgenti degli eventi e richiama la funzione duratura in modo asincrono, rimuovendo il limite di esecuzione di 15 minuti.

Questo modello disaccoppia il modello di invocazione sincrona della mappatura della sorgente degli eventi dal modello di esecuzione a lunga durata della funzione duratura. La mappatura delle sorgenti degli eventi richiama la funzione intermedia, che ritorna rapidamente dopo l'avvio dell'esecuzione duratura. La funzione durable viene quindi eseguita in modo indipendente per tutto il tempo necessario (fino a 1 anno).

Architecture

Il modello di funzione intermedia utilizza tre componenti:

1. Mappatura delle sorgenti degli eventi: interroga l'origine dell'evento (Amazon SQS, Kinesis, DynamoDB Streams) e richiama la funzione intermediaria in modo sincrono con i batch di record.
2. Funzione intermedia: una funzione Lambda standard che riceve gli eventi dalla mappatura dell'origine degli eventi, convalida e trasforma i dati se necessario e richiama la funzione duratura in modo asincrono. Questa funzione si completa rapidamente (in genere in meno di 1 secondo) e restituisce il controllo alla mappatura delle sorgenti degli eventi.
3. Funzione durevole: elabora l'evento con una logica complessa a più fasi che può essere eseguita per periodi prolungati. Richiamato in modo asincrono, quindi non è vincolato dal limite di 15 minuti.

Implementazione

La funzione intermediaria riceve l'intero evento dalla mappatura della sorgente dell'evento e richiama la funzione durevole in modo asincrono. Utilizzate il parametro del nome di esecuzione per garantire l'avvio di un'esecuzione idempotente, evitando l'elaborazione duplicata se la mappatura dell'origine dell'evento riprova:

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}
```

Per l'idempotenza nella funzione intermedia stessa, usate Powertools per evitare invocazioni duplicate della funzione durable se la AWS Lambda mappatura della sorgente dell'evento riprova la funzione intermediaria.

La funzione durable riceve il payload con il nome di esecuzione ed elabora tutti i record con una logica di lunga durata:

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
import json

@durable_execution
def handler(payload, context: DurableContext):
    sqs_event = payload['event']

    # Process each record with complex, multi-step logic
```

```
    def process_record(ctx, record):
        validated = ctx.step(
            lambda _: validate_order(json.loads(record['body'])),
            name='validate'
        )

        # Wait for external approval (could take hours or days)
        approval = ctx.wait_for_callback(
            lambda callback_id: request_approval(callback_id, validated),
            name='approval',
            config=WaitForCallbackConfig(timeout_seconds=172800)  # 48 hours
        )

        # Complete processing
        return ctx.step(
            lambda _: complete_order(validated, approval),
            name='complete'
        )

    results = context.map(sqs_event['Records'], process_record)

    return {'statusCode': 200, 'processed': len(results.get_results())}
```

Considerazioni chiave

Questo modello rimuove il limite di esecuzione di 15 minuti disaccoppiando la mappatura della sorgente dell'evento dall'esecuzione durevole. La funzione intermedia ritorna immediatamente dopo l'avvio dell'esecuzione duratura, consentendo alla mappatura della fonte dell'evento di continuare l'elaborazione. La funzione durable viene quindi eseguita in modo indipendente per tutto il tempo necessario.

La funzione intermedia ha successo quando richiama la funzione durable, non quando l'esecuzione duratura viene completata. Se l'esecuzione duratura fallisce in un secondo momento, la mappatura della fonte dell'evento non riproverà perché il batch è già stato elaborato correttamente. Implementa la gestione degli errori nella funzione durable e configura le code di lettere morte per le esecuzioni non riuscite.

Utilizzate il parametro del nome di esecuzione per garantire l'avvio di un'esecuzione idempotente. Se la mappatura dell'origine dell'evento riprova la funzione intermedia, la funzione durable non avvierà un'esecuzione duplicata perché il nome dell'esecuzione esiste già.

Fonti di eventi supportate

Le funzioni durevoli supportano tutte le sorgenti di eventi Lambda che utilizzano le mappature delle sorgenti degli eventi:

- Code Amazon SQS (standard e FIFO)
- Flussi Kinesis
- DynamoDB Streams
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)
- Apache Kafka gestito dal cliente
- Amazon MQ (ActiveMQ e RabbitMQ)
- Flussi di modifica di Amazon DocumentDB

Tutti i tipi di sorgenti di eventi sono soggetti al limite di esecuzione durevole di 15 minuti quando si richiamano funzioni durevoli.

Riprova per le funzioni durevoli Lambda

Le funzioni durevoli offrono funzionalità di ripetizione automatica che rendono le applicazioni resilienti ai guasti transitori. L'SDK gestisce i nuovi tentativi a due livelli: ripetuti passaggi per gli errori della logica aziendale e i tentativi di backend per gli errori dell'infrastruttura.

Fase 3 nuovi tentativi

Quando si verifica un'eccezione non rilevata in un passaggio, l'SDK riprova automaticamente il passaggio in base alla strategia di ripetizione configurata. I nuovi tentativi di passaggio sono operazioni bloccate che consentono all'SDK di sospendere l'esecuzione e riprenderla in un secondo momento senza perdere i progressi.

Comportamento dei tentativi successivi

La tabella seguente descrive come l'SDK gestisce le eccezioni all'interno dei passaggi:

Scenario	Cosa succede	Impatto della misurazione
Eccezione relativa ai tentativi di nuovi tentativi rimanenti	L'SDK crea un checkpoint per il nuovo tentativo e sospende	1 operazione più errore, dimensione del payload

Scenario	Cosa succede	Impatto della misurazione
	la funzione. Alla chiamata successiva, il passaggio riprova con il ritardo di backoff configurato.	
Eccezione in fase iniziale senza tentativi di riprova rimanenti	Il passaggio ha esito negativo e genera un'eccezione. Se il codice del gestore non rileva questa eccezione, l'intera esecuzione fallisce.	1 operazione più errore (dimensione del payload)

Quando è necessario riprovare un passaggio, l'SDK controlla lo stato del nuovo tentativo ed esce dalla chiamata Lambda se non è in esecuzione nessun altro lavoro. Ciò consente all'SDK di implementare ritardi di backoff senza consumare risorse di elaborazione. La funzione riprende automaticamente dopo il periodo di backoff.

Configurazione delle strategie di ripetizione dei passaggi

Configura le strategie di ripetizione dei tentativi per controllare il modo in cui le fasi gestiscono gli errori. È possibile specificare il numero massimo di tentativi, gli intervalli di backoff e le condizioni per i nuovi tentativi.

Backoff esponenziale con numero massimo di tentativi:

TypeScript

```
const result = await context.step('call-api', async () => {
  const response = await fetch('https://api.example.com/data');
  if (!response.ok) throw new Error(`API error: ${response.status}`);
  return await response.json();
}, {
  retryStrategy: (error, attemptCount) => {
    if (attemptCount >= 5) {
      return { shouldRetry: false };
    }
    // Exponential backoff: 2s, 4s, 8s, 16s, 32s (capped at 300s)
    const delay = Math.min(2 * Math.pow(2, attemptCount - 1), 300);
```

```
    return { shouldRetry: true, delay: { seconds: delay } };
  }
});
```

Python

```
def retry_strategy(error, attempt_count):
    if attempt_count >= 5:
        return {'should_retry': False}
    # Exponential backoff: 2s, 4s, 8s, 16s, 32s (capped at 300s)
    delay = min(2 * (2 ** (attempt_count - 1)), 300)
    return {'should_retry': True, 'delay': delay}

result = context.step(
    lambda _: call_external_api(),
    name='call-api',
    config=StepConfig(retry_strategy=retry_strategy)
)
```

Backoff a intervallo fisso:

TypeScript

```
const orders = await context.step('query-orders', async () => {
  return await queryDatabase(event.userId);
}, {
  retryStrategy: (error, attemptCount) => {
    if (attemptCount >= 3) {
      return { shouldRetry: false };
    }
    return { shouldRetry: true, delay: { seconds: 5 } };
  }
});
```

Python

```
def retry_strategy(error, attempt_count):
    if attempt_count >= 3:
        return {'should_retry': False}
    return {'should_retry': True, 'delay': 5}

orders = context.step(
    lambda _: query_database(event['userId']),
    name='query-orders',
    config=StepConfig(retry_strategy=retry_strategy)
)
```

Riprova condizionale (riprova solo errori specifici):

TypeScript

```
const result = await context.step('call-rate-limited-api', async () => {
  const response = await fetch('https://api.example.com/data');

  if (response.status === 429) throw new Error('RATE_LIMIT');
  if (response.status === 504) throw new Error('TIMEOUT');
  if (!response.ok) throw new Error(`API_ERROR_${response.status}`);

  return await response.json();
}, {
  retryStrategy: (error, attemptCount) => {
    // Only retry rate limits and timeouts
    const isRetryable = error.message === 'RATE_LIMIT' || error.message ===
 'TIMEOUT';

    if (!isRetryable || attemptCount >= 3) {
      return { shouldRetry: false };
    }

    // Exponential backoff: 1s, 2s, 4s (capped at 30s)
    const delay = Math.min(Math.pow(2, attemptCount - 1), 30);
    return { shouldRetry: true, delay: { seconds: delay } };
  }
});
```

Python

```
def retry_strategy(error, attempt_count):
    # Only retry rate limits and timeouts
    is_retryable = str(error) in ['RATE_LIMIT', 'TIMEOUT']

    if not is_retryable or attempt_count >= 3:
        return {'should_retry': False}

    # Exponential backoff: 1s, 2s, 4s (capped at 30s)
    delay = min(2 ** (attempt_count - 1), 30)
    return {'should_retry': True, 'delay': delay}

result = context.step(
    lambda _: call_rate_limited_api(),
    name='call-rate-limited-api',
    config=StepConfig(retry_strategy=retry_strategy)
)
```

Disabilita i tentativi:

TypeScript

```
const isDuplicate = await context.step('check-duplicate', async () => {
  return await checkIfOrderExists(event.orderId);
}, {
  retryStrategy: () => ({ shouldRetry: false })
});
```

Python

```
is_duplicate = context.step(
    lambda _: check_if_order_exists(event['orderId']),
    name='check-duplicate',
    config=StepConfig(
        retry_strategy=lambda error, attempt: {'should_retry': False}
    )
```

```
)
```

Quando viene ripristinata la strategia di nuovi tentativi`shouldRetry: false`, il passaggio fallisce immediatamente senza nuovi tentativi. Usalo per operazioni che non devono essere ripetute, come i controlli dell'idempotenza o le operazioni con effetti collaterali che non possono essere ripetuti in sicurezza.

Eccezioni all'esterno dei gradini

Quando si verifica un'eccezione non rilevata nel codice del gestore ma al di fuori di qualsiasi passaggio, l'SDK contrassegna l'esecuzione come non riuscita. Ciò garantisce che gli errori nella logica dell'applicazione vengano rilevati e segnalati correttamente.

Scenario	Cosa succede	Impatto della misurazione
Eccezione nel codice del gestore al di fuori di qualsiasi passaggio	L'SDK contrassegna l'esecuzione come NON RIUSCITA e restituisce l'errore. L'eccezione non viene ritentata automaticamente.	Errore nella dimensione del payload

Per abilitare un nuovo tentativo automatico per il codice soggetto a errori, inseriscilo in un unico passaggio con una strategia di riprova. I passaggi prevedono un nuovo tentativo automatico con backoff configurabile, mentre il codice al di fuori dei passaggi fallisce immediatamente.

Tentativi di backend

I nuovi tentativi di backend si verificano quando Lambda rileva guasti dell'infrastruttura, errori di runtime o quando l'SDK non è in grado di comunicare con il servizio di esecuzione durevole. Lambda riprova automaticamente questi errori per garantire che le funzioni durevoli possano essere ripristinate da problemi transitori dell'infrastruttura.

Scenari di nuovi tentativi nel backend

Lambda riprova automaticamente la funzione quando si verificano i seguenti scenari:

- Errori interni del servizio: quando Lambda o il servizio di esecuzione durevole restituisce un errore 5xx, che indica un problema temporaneo del servizio.
- Limitazione: quando la funzione viene limitata a causa di limiti di concorrenza o quote di servizio.
- Timeout: quando l'SDK non riesce a raggiungere il servizio di esecuzione durevole entro il periodo di timeout.
- Errori di inizializzazione della sandbox: quando Lambda non è in grado di inizializzare l'ambiente di esecuzione.
- Errori di runtime: quando il runtime Lambda rileva errori esterni al codice della funzione, come out-of-memory errori o arresti anomali del processo.
- Errori del token di checkpoint non valido: quando il token di checkpoint non è più valido, in genere a causa di modifiche dello stato sul lato del servizio.

La tabella seguente descrive come l'SDK gestisce questi scenari:

Scenario	Cosa succede	Impatto della misurazione
Errore di runtime esterno al gestore durevole (OOM, timeout, crash)	Lambda riprova automaticamente la chiamata. L'SDK riproduce a partire dall'ultimo checkpoint, saltando i passaggi completati.	Errore: dimensione del payload + 1 operazione per nuovo tentativo
Errore di servizio (5xx) o timeout durante la chiamata/ `CheckpointDurableExecution` `GetDurableExecutionState` APIs	Lambda riprova automaticamente la chiamata. L'SDK viene riprodotto a partire dall'ultimo checkpoint.	Errore: dimensione del payload + 1 operazione per nuovo tentativo
Limitazione (429) o token di checkpoint non valido durante la chiamata/ `CheckpointDurableExecution` `GetDurableExecutionState` APIs	Lambda ritenta automaticamente la chiamata con un backoff esponenziale. L'SDK viene riprodotto a partire dall'ultimo checkpoint.	Errore: dimensione del payload + 1 operazione per nuovo tentativo

Scenario	Cosa succede	Impatto della misurazione
Errore del client (4xx, tranne 429 e token non valido) quando/`CheckpointDurableExecution` `GetDurableExecutionState` APIs	L'SDK contrassegna l'esecuzione come NON RIUSCITA. Non si verifica alcun nuovo tentativo automatico perché l'errore indica un problema permanente.	Errore: dimensione del payload.

I tentativi di backend utilizzano il backoff esponenziale e continuano fino al successo della funzione o al raggiungimento del timeout di esecuzione. Durante la riproduzione, l'SDK salta i checkpoint completati e continua l'esecuzione dell'ultima operazione riuscita, assicurando che la funzione non riesegua il lavoro completato.

Riprova le best practice

Segui queste best practice per configurare le strategie di riprova:

- Configura strategie esplicite di nuovi tentativi: non fare affidamento sul comportamento predefinito dei nuovi tentativi in produzione. Configura strategie di riprova esplicite con un numero massimo di tentativi e intervalli di backoff appropriati per il tuo caso d'uso.
- Utilizza tentativi condizionali: implementa la **`shouldRetry`** logica per riprovare solo gli errori transitori (limiti di velocità, timeout) e fallire rapidamente in caso di errori permanenti (errori di convalida, non rilevati).
- Imposta il numero massimo di tentativi appropriato: equilibrio tra resilienza e tempo di esecuzione. Troppi tentativi possono ritardare il rilevamento degli errori, mentre troppo pochi possono causare errori non necessari.
- Utilizza il backoff esponenziale: il backoff esponenziale riduce il carico sui servizi a valle e aumenta la probabilità di ripristino da guasti transitori.
- Raccogli il codice soggetto a errori in fasi: il codice al di fuori dei passaggi non può essere riprovato automaticamente. Raccogli le chiamate API esterne, le query sul database e altre operazioni soggette a errori in fasi con strategie di riprova.

- Monitora le metriche dei tentativi: monitora le operazioni di ripetizione dei tentativi e gli errori di esecuzione in CloudWatch Amazon per identificare modelli e ottimizzare le strategie di ripetizione dei tentativi.

Idempotenza

Le funzioni durevoli forniscono un'idempotenza integrata per gli avvii di esecuzione tramite i nomi di esecuzione. Quando fornisci un nome di esecuzione, Lambda lo utilizza per prevenire esecuzioni duplicate e consentire nuovi tentativi sicuri delle richieste di invocazione. Per impostazione predefinita, le fasi hanno una semantica di at-least-once esecuzione: durante la riproduzione, l'SDK restituisce risultati puntuali senza rieseguire i passaggi completati, ma la logica aziendale deve essere idempotente per gestire potenziali tentativi prima del completamento.

> **Note**
>
> Le mappature delle sorgenti degli eventi Lambda (ESM) non supportano l'idempotenza al momento del lancio. Pertanto, ogni chiamata (inclusi i nuovi tentativi) avvia una nuova esecuzione durevole. Per garantire un'esecuzione idempotente con mappature delle sorgenti degli eventi, implementate la logica di idempotenza nel codice della funzione, ad esempio con Powertools for, oppure AWS Lambda utilizzate una normale funzione Lambda come proxy (dispatcher) per richiamare una funzione durevole con una chiave di idempotenza (parametro del nome di esecuzione).

Nomi di esecuzione

È possibile fornire un nome di esecuzione quando si richiama una funzione durevole. Il nome di esecuzione funge da chiave di idempotenza, che consente di riprovare in sicurezza le richieste di invocazione senza creare esecuzioni duplicate. Se non fornisci un nome, Lambda genera automaticamente un ID di esecuzione univoco.

I nomi di esecuzione devono essere univoci all'interno del tuo account e della tua regione. Quando si richiama una funzione con un nome di esecuzione già esistente, il comportamento di Lambda dipende dallo stato dell'esecuzione esistente e dalla corrispondenza del payload.

Comportamento di idempotenza

La tabella seguente descrive come Lambda gestisce le richieste di chiamata in base al fatto che tu fornisca un nome di esecuzione, lo stato di esecuzione esistente e se il payload corrisponde:

Scenario	Nome fornito?	Stato di esecuzione esistente	Payload identico?	Comportamento
1	No	N/D	N/D	Nuova esecuzione iniziata: Lambda genera un ID di esecuzione univoco e avvia una nuova esecuzione
2	Sì	Non è mai esistito o la conservazione è scaduta	N/D	Nuova esecuzione iniziata: Lambda avvia una nuova esecuzione con il nome fornito
3	Sì	In esecuzione	Sì	Avvio idempotente: Lambda restituisce le informazioni di esecuzione esistenti senza avviare un duplicato. Per le chiamate sincrone, ciò funge da ricollegamento all'esecuzione in esecuzione
4	Sì	In esecuzione	No	Errore: Lambda restituisce un `DurableExecutionAlreadyExists` errore perché un'esecuzione con

Scenario	Nome fornito?	Stato di esecuzione esistente	Payload identico?	Comportamento
				questo nome è già in esecuzione con un payload diverso
5	Sì	Chiuso (riuscito, non riuscito, interrotto o scaduto)	Sì	Avvio idempotente: Lambda restituisce le informazioni di esecuzione esistenti senza iniziare una nuova esecuzion e. Viene restituito il risultato dell'esec uzione chiusa
6	Sì	Chiuso (riuscito, fallito, interrotto o scaduto)	No	Errore: Lambda restituisce un `DurableEx ecutionAl readyExis ts` errore perché un'esecuzione con questo nome è già stata completata con un payload diverso

Nota

Gli scenari 3 e 5 dimostrano un comportamento idempotente in cui Lambda gestisce in modo sicuro le richieste di invocazione duplicate restituendo le informazioni di esecuzione esistenti anziché creare duplicati.

Fase: idempotenza

Per impostazione predefinita, i passaggi hanno una semantica di at-least-once esecuzione. Quando la funzione viene riprodotta dopo un'attesa, un callback o un errore, l'SDK confronta ogni passaggio

con il registro del checkpoint. Per i passaggi già completati, l'SDK restituisce il risultato del checkpoint senza rieseguire la logica dei passaggi. Tuttavia, se un passaggio fallisce o la funzione viene interrotta prima del completamento del passaggio, il passaggio può essere eseguito più volte.

La logica aziendale racchiusa in fasi deve essere idempotente per gestire potenziali nuovi tentativi. Utilizza le chiavi di idempotenza per garantire che operazioni come pagamenti o scritture di database vengano eseguite una sola volta, anche se il passaggio riprova.

Esempio: utilizzo delle chiavi di idempotenza in fasi

TypeScript

```
import { withDurableExecution, DurableContext } from '@aws/durable-execution-sdk-
js';
import { randomUUID } from 'crypto';

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    // Generate idempotency key once
    const idempotencyKey = await context.step('generate-key', async () => {
      return randomUUID();
    });

    // Use idempotency key in payment API to prevent duplicate charges
    const payment = await context.step('process-payment', async () => {
      return paymentAPI.charge({
        amount: event.amount,
        idempotencyKey: idempotencyKey
      });
    });

    return { statusCode: 200, payment };
  }
);
```

Python

```
from aws_durable_execution_sdk_python import durable_execution, DurableContext
import uuid
```

```
@durable_execution
def handler(event, context: DurableContext):
    # Generate idempotency key once
    idempotency_key = context.step(
        lambda _: str(uuid.uuid4()),
        name='generate-key'
    )

    # Use idempotency key in payment API to prevent duplicate charges
    payment = context.step(
        lambda _: payment_api.charge(
            amount=event['amount'],
            idempotency_key=idempotency_key
        ),
        name='process-payment'
    )

    return {'statusCode': 200, 'payment': payment}
```

È possibile configurare i passaggi per utilizzare la semantica di at-most-once esecuzione impostando la modalità di esecuzione su. `AT_MOST_ONCE_PER_RETRY` Ciò garantisce che il passaggio venga eseguito al massimo una volta per ogni nuovo tentativo, ma potrebbe non essere eseguito affatto se la funzione viene interrotta prima del completamento del passaggio.

L'SDK applica la replay deterministica convalidando che i nomi e l'ordine delle fasi corrispondano al registro del checkpoint durante la ripetizione. Se il codice tenta di eseguire i passaggi in un ordine diverso o con nomi diversi, l'SDK genera un. `NonDeterministicExecutionError`

Come funziona il replay con i passaggi completati:

1. Prima chiamata: la funzione esegue il passaggio A, crea il checkpoint, quindi attende
2. Seconda invocazione (dopo l'attesa): la funzione viene riprodotta dall'inizio, la fase A restituisce immediatamente il risultato del checkpoint senza rieseguirla, quindi continua con la fase B
3. Terza invocazione (dopo un'altra attesa): la funzione viene riprodotta dall'inizio, i passaggi A e B restituiscono istantaneamente i risultati del checkpoint, quindi prosegue con la fase C

Questo meccanismo di replay assicura che i passaggi completati non vengano rieseguiti, ma la logica aziendale deve comunque essere idempotente per gestire i nuovi tentativi prima del completamento.

Test delle funzioni durevoli di Lambda

AWS fornisce test dedicati SDKs per funzioni durevoli che consentono di eseguire e ispezionare le esecuzioni sia localmente che nel cloud. Installa l'SDK di test per la tua lingua:

TypeScript

```
npm install --save-dev @aws/aws-durable-execution-sdk-js-testing
```

Per la documentazione completa e gli esempi, consulta l'SDK TypeScript di test su. GitHub

Python

```
pip install aws-durable-execution-sdk-python-testing
```

Per la documentazione completa e gli esempi, consulta l'SDK per i test di Python su. GitHub

L'SDK di test offre due modalità di test: test locali per test unitari rapidi e test su cloud per test di integrazione rispetto alle funzioni distribuite.

Test locale

I test locali eseguono le funzioni durevoli nel tuo ambiente di sviluppo senza richiedere risorse distribuite. Il test runner esegue direttamente il codice della funzione e acquisisce tutte le operazioni per l'ispezione.

Utilizza i test locali per i test unitari, lo sviluppo basato sui test e le pipeline. CI/CD I test vengono eseguiti localmente senza latenza di rete o costi aggiuntivi.

Test di esempio:

TypeScript

```
import { withDurableExecution } from '@aws/aws-durable-execution-sdk-js';
import { DurableFunctionTestRunner } from '@aws/aws-durable-execution-sdk-js-
testing';
```

```
const handler = withDurableExecution(async (event, context) => {
  const result = await context.step('calculate', async () => {
    return event.a + event.b;
  });
  return result;
});

test('addition works correctly', async () => {
  const runner = new DurableFunctionTestRunner({ handler });
  const result = await runner.run({ a: 5, b: 3 });

  expect(result.status).toBe('SUCCEEDED');
  expect(result.result).toBe(8);

  const step = result.getStep('calculate');
  expect(step.result).toBe(8);
});
```

Python

```
from aws_durable_execution_sdk_python import durable_execution, DurableContext
from aws_durable_execution_sdk_python_testing import DurableFunctionTestRunner
from aws_durable_execution_sdk_python.execution import InvocationStatus

@durable_execution
def handler(event: dict, context: DurableContext) -> int:
    result = context.step(lambda _: event["a"] + event["b"], name="calculate")
    return result

def test_addition():
    runner = DurableFunctionTestRunner(handler=handler)
    with runner:
        result = runner.run(input={"a": 5, "b": 3}, timeout=10)

    assert result.status is InvocationStatus.SUCCEEDED
    assert result.result == 8

    step = result.get_step("calculate")
    assert step.result == 8
```

Il test runner acquisisce lo stato di esecuzione, inclusi il risultato finale, i risultati dei singoli passaggi, le operazioni di attesa, i callback ed eventuali errori. È possibile ispezionare le operazioni per nome o ripetere tutte le operazioni per verificare il comportamento di esecuzione.

Archivi di esecuzione

L'SDK di test utilizza gli archivi di esecuzione per rendere persistenti i dati di esecuzione dei test. Per impostazione predefinita, i test utilizzano un archivio in memoria che è veloce e non richiede alcuna pulizia. Per il debug o l'analisi della cronologia delle esecuzioni, puoi utilizzare un archivio di file system che salva le esecuzioni come file JSON.

Archivio in memoria (predefinito):

L'archivio in memoria mantiene i dati di esecuzione in memoria durante le esecuzioni dei test. I dati vengono persi al termine dei test, il che li rende ideali per i test unitari standard e CI/CD le pipeline in cui non è necessario ispezionare le esecuzioni al termine dei test.

Archivio del file system:

L'archivio del file system memorizza i dati di esecuzione su disco come file JSON. Ogni esecuzione viene salvata in un file separato, semplificando l'ispezione della cronologia di esecuzione dopo il completamento dei test. Utilizza l'archivio del file system per eseguire il debug di test falliti complessi o per analizzare i modelli di esecuzione nel tempo.

Configura lo store utilizzando le variabili di ambiente:

```
# Use filesystem store
export AWS_DEX_STORE_TYPE=filesystem
export AWS_DEX_STORE_PATH=./test-executions

# Run tests
pytest tests/
```

I file di esecuzione sono archiviati con nomi anonimi e contengono lo stato di esecuzione completo, comprese le operazioni, i checkpoint e i risultati. L'archivio del filesystem crea automaticamente la directory di archiviazione se non esiste.

Test sul cloud

Il test nel cloud richiama le funzioni durevoli implementate AWS e ne recupera la cronologia di esecuzione utilizzando l'API Lambda. Utilizza i test sul cloud per verificare il comportamento in ambienti simili a quelli di produzione con servizi e configurazioni reali. AWS

Il cloud testing richiede una funzione distribuita e AWS credenziali con autorizzazioni per richiamare funzioni e leggere la cronologia di esecuzione:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction",
                "lambda:GetDurableExecution",
                "lambda:GetDurableExecutionHistory"
            ],
            "Resource": "arn:aws:lambda:region:account-id:function:function-name"
        }
    ]
}
```

Esempio di test nel cloud:

TypeScript

```
import { DurableFunctionCloudTestRunner } from '@aws/aws-durable-execution-sdk-js-
testing';

test('deployed function processes orders', async () => {
  const runner = new DurableFunctionCloudTestRunner({
    functionName: 'order-processor',
    region: 'us-east-1'
  });

  const result = await runner.run({ orderId: 'order-123' });
```

```
  expect(result.status).toBe('SUCCEEDED');
  expect(result.result.status).toBe('completed');
});
```

Python

```
from aws_durable_execution_sdk_python_testing import (
    DurableFunctionCloudTestRunner,
    DurableFunctionCloudTestRunnerConfig
)

def test_deployed_function():
    config = DurableFunctionCloudTestRunnerConfig(
        function_name="order-processor",
        region="us-east-1"
    )
    runner = DurableFunctionCloudTestRunner(config=config)

    result = runner.run(input={"orderId": "order-123"})

    assert result.status is InvocationStatus.SUCCEEDED
    assert result.result["status"] == "completed"
```

I test nel cloud richiamano la funzione effettivamente implementata e recuperano la cronologia di esecuzione da. AWS Ciò consente di verificare l'integrazione con altri AWS servizi, convalidare le caratteristiche prestazionali e testare con dati e configurazioni simili a quelli di produzione.

Cosa testare

Testa le funzioni durevoli verificando i risultati di esecuzione, il comportamento operativo e la gestione degli errori. Concentrati sulla correttezza della logica aziendale piuttosto che sui dettagli di implementazione.

Verifica i risultati dell'esecuzione: verifica che le funzioni restituiscano i valori previsti per determinati input. Verifica sia le esecuzioni riuscite che i casi di errore per garantire che le funzioni gestiscano gli input non validi in modo appropriato.

Ispeziona l'esecuzione dell'operazione: verifica che i passaggi, le attese e i callback vengano eseguiti come previsto. Controllate i risultati delle fasi per assicurarvi che le operazioni intermedie producano valori corretti. Verifica che le operazioni di attesa siano configurate con i timeout appropriati e che i callback siano creati con le impostazioni corrette.

Gestione degli errori di test: verifica che le funzioni falliscano correttamente con messaggi di errore descrittivi quando viene fornito un input non valido. Verifica il comportamento dei nuovi tentativi simulando gli errori temporanei e confermando che le operazioni siano ripetute correttamente. Verificate che i guasti permanenti non generino tentativi non necessari.

Convalida i flussi di lavoro: per i flussi di lavoro in più fasi, verifica che le operazioni vengano eseguite nell'ordine corretto. Verifica la ramificazione condizionale per assicurarti che i diversi percorsi di esecuzione funzionino correttamente. Convalida le operazioni parallele, eseguite contemporaneamente e producono i risultati previsti.

Gli archivi di documentazione SDK contengono numerosi esempi di modelli di test, tra cui flussi di lavoro in più fasi, scenari di errore, gestione dei timeout e modelli di polling.

Strategia di test

Utilizza i test locali per i test unitari durante lo sviluppo e nelle CI/CD pipeline. I test locali vengono eseguiti rapidamente, non richiedono AWS credenziali e forniscono un feedback immediato sulle modifiche al codice. Scrivi test locali per verificare la logica aziendale, la gestione degli errori e il comportamento operativo.

Usa i test sul cloud per i test di integrazione prima di passare alla produzione. I test sul cloud verificano il comportamento con AWS servizi e configurazioni reali, convalidano le caratteristiche prestazionali e testano i flussi di lavoro. end-to-end Esegui test sul cloud in ambienti di staging per individuare i problemi di integrazione prima che raggiungano la produzione.

Simula le dipendenze esterne nei test locali per isolare la logica delle funzioni e mantenere veloci i test. Utilizza i test sul cloud per verificare l'effettiva integrazione con servizi esterni come database e altri APIs servizi. AWS

Scrivi test mirati che verifichino un comportamento specifico. Usa nomi di test descrittivi che spieghino cosa viene testato. Raggruppa i test correlati e utilizza i dispositivi di test per il codice di configurazione comune. Mantieni i test semplici ed evita logiche di test complesse e difficili da capire.

Errori di debug

Quando i test falliscono, controlla il risultato dell'esecuzione per capire cosa è andato storto. Controlla lo stato di esecuzione per vedere se la funzione è riuscita, non riuscita o è scaduta. Leggi i messaggi di errore per comprendere la causa dell'errore.

Esamina i risultati delle singole operazioni per scoprire dove il comportamento si discosta dalle aspettative. Controlla i risultati delle fasi per vedere quali valori sono stati prodotti. Verifica l'ordine delle operazioni per confermare le operazioni eseguite nella sequenza prevista. Conta le operazioni per garantire che sia stato creato il giusto numero di passaggi, attese e callback.

I problemi più comuni includono il codice non deterministico che produce risultati diversi durante la riproduzione, lo stato condiviso tramite variabili globali che si interrompe durante la riproduzione e le operazioni mancanti a causa di errori di logica condizionale. Usa i debugger e la registrazione standard per esaminare il codice delle funzioni e tenere traccia del flusso di esecuzione.

Per i test sul cloud, controlla la cronologia di esecuzione in CloudWatch Logs per visualizzare i registri operativi dettagliati. Utilizza il tracciamento per tracciare il flusso di esecuzione tra i servizi e identificare i punti deboli.

Monitoraggio di funzioni durature

Puoi monitorare le tue funzioni durevoli utilizzando CloudWatch metriche, CloudWatch registri e tracciamento. Poiché le funzioni durevoli possono funzionare per periodi prolungati e richiamare più funzioni, il loro monitoraggio richiede la comprensione dei loro modelli di esecuzione unici, inclusi i checkpoint, le transizioni di stato e il comportamento di riproduzione.

CloudWatch metriche

Lambda pubblica automaticamente le metriche senza costi CloudWatch aggiuntivi. Le funzioni durevoli forniscono metriche aggiuntive oltre alle metriche Lambda standard per aiutarti a monitorare i flussi di lavoro di lunga durata, la gestione dello stato e l'utilizzo delle risorse.

Metriche di esecuzione durevoli

Lambda emette le seguenti metriche per esecuzioni durevoli:

Metrica	Description
`ApproximateRunning DurableExecutions`	Numero di esecuzioni durevoli nello stato RUNNING
`ApproximateRunning DurableExecutionsU tilization`	Percentuale della quota massima di esecuzioni durevoli in esecuzione del tuo account attualmente in uso
`DurableExecutionDuration`	Tempo trascorso dall'orologio da parete, espresso in millisecondi, durante il quale un'esecuzione durevole è rimasta nello stato RUNNING
`DurableExecutionStarted`	Numero di esecuzioni durevoli avviate
`DurableExecutionStopped`	Numero di esecuzioni durevoli interrotte utilizzando l'API StopDurableExecution
`DurableExecutionSucceeded`	Numero di esecuzioni durevoli completate con successo
`DurableExecutionFailed`	Numero di esecuzioni durevoli completate con un errore
`DurableExecutionTimedOut`	Numero di esecuzioni durevoli che hanno superato il timeout di esecuzione configurato
`DurableExecutionOp erations`	Numero cumulativo di operazioni eseguite nell'ambito di un'esecuzione durevole (max: 3.000)
`DurableExecutionSt orageWrittenBytes`	Quantità cumulativa di dati in byte resa persistente da un'esecuzione durevole (max: 100 MB)

CloudWatch metriche

Lambda emette metriche standard di invocazione, prestazioni e concorrenza per funzioni durevoli. Poiché un'esecuzione durevole può includere più chiamate di funzioni man mano che avanza attraverso checkpoint e replay, queste metriche si comportano in modo diverso rispetto alle funzioni standard:

- Invocazioni: conta ogni chiamata di funzione, inclusi i replay. Una singola esecuzione durevole può generare più punti dati di invocazione.
- Durata: misura ogni chiamata di funzione separatamente. Da utilizzare `DurableExecutionDuration` per il tempo totale impiegato da una singola esecuzione durevole.
- Errori: tiene traccia degli errori di invocazione delle funzioni. Da utilizzare per gli errori a livello `DurableExecutionFailed` di esecuzione.

Per un elenco completo delle metriche Lambda standard, consulta Tipi di metriche per le funzioni Lambda.

Creazione di allarmi CloudWatch

Crea CloudWatch allarmi per avvisarti quando le metriche superano le soglie. Gli allarmi più comuni includono:

- `ApproximateRunningDurableExecutionsUtilization`supera l'80% della tua quota
- `DurableExecutionFailed`aumenta al di sopra di una soglia
- `DurableExecutionTimedOut`indica che le esecuzioni stanno scadendo
- `DurableExecutionStorageWrittenBytes`si avvicina ai limiti di archiviazione

Per ulteriori informazioni, consulta Uso degli CloudWatch allarmi. .

EventBridge eventi

Lambda pubblica eventi duraturi di modifica dello stato di esecuzione su. EventBridge Puoi utilizzare questi eventi per attivare flussi di lavoro, inviare notifiche o tenere traccia delle modifiche del ciclo di vita dell'esecuzione tra le tue funzioni durevoli.

Eventi durevoli di modifica dello stato di esecuzione

Lambda emette un evento EventBridge ogni volta che un'esecuzione durevole cambia lo stato. Questi eventi hanno le seguenti caratteristiche:

- Origine: `aws.lambda`
- Tipo di dettaglio: `Durable Execution Status Change`

Gli eventi di modifica dello stato vengono pubblicati per i seguenti stati di esecuzione:

- `RUNNING`- Esecuzione iniziata
- `SUCCEEDED`- Esecuzione completata con successo
- `STOPPED`- Esecuzione interrotta utilizzando l' StopDurableExecution API
- `FAILED`- Esecuzione non riuscita con un errore
- `TIMED_OUT`- L'esecuzione ha superato il timeout configurato

L'esempio seguente mostra un evento duraturo di modifica dello stato di esecuzione:

```
{
  "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"
  }
}
```

Per gli stati terminali (`SUCCEEDED`STOPPED,,`FAILED`,`TIMED_OUT`), l'evento include un `endTimestamp` campo che indica quando l'esecuzione è stata completata.

Creazione di EventBridge regole

Crea regole per indirizzare eventi duraturi di modifica dello stato di esecuzione verso obiettivi come Amazon Simple Notification Service, Amazon Simple Queue Service o altre funzioni Lambda.

L'esempio seguente crea una regola che corrisponde a tutte le modifiche durature dello stato di esecuzione:

```
{
  "source": ["aws.lambda"],
  "detail-type": ["Durable Execution Status Change"]
}
```

L'esempio seguente crea una regola che corrisponde solo alle esecuzioni non riuscite:

```
{
  "source": ["aws.lambda"],
  "detail-type": ["Durable Execution Status Change"],
  "detail": {
    "status": ["FAILED"]
  }
}
```

L'esempio seguente crea una regola che corrisponde alle modifiche di stato per una funzione specifica:

```
{
  "source": ["aws.lambda"],
  "detail-type": ["Durable Execution Status Change"],
  "detail": {
    "functionArn": [{
      "prefix": "arn:aws:lambda:us-east-1:123456789012:function:my-function"
    }]
  }
}
```

Per ulteriori informazioni sulla creazione di regole, consulta EventBridge i tutorial di Amazon nella Guida per l' EventBridge utente.

AWS X-Ray tracciamento

È possibile abilitare il tracciamento a raggi X sulle funzioni durevoli. Lambda passa l'intestazione di traccia X-Ray all'esecuzione durevole, consentendoti di tracciare le richieste in tutto il flusso di lavoro.

Per abilitare il tracciamento a raggi X tramite la console Lambda, scegli la tua funzione, quindi scegli Strumenti di configurazione, monitoraggio e operazioni e attiva il tracciamento attivo in X-Ray.

Per abilitare il tracciamento X-Ray utilizzando: AWS CLI

```
aws lambda update-function-configuration \
    --function-name my-durable-function \
    --tracing-config Mode=Active
```

Per abilitare il AWS X-Ray tracciamento utilizzando: AWS SAM

```
Resources:
  MyDurableFunction:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      DurableConfig:
        ExecutionTimeout: 3600
```

Per ulteriori informazioni su X-Ray, consulta la AWS X-Ray Developer Guide.

Le migliori pratiche per le funzioni durevoli Lambda

Le funzioni durevoli utilizzano un modello di esecuzione basato sulla riproduzione che richiede modelli diversi rispetto alle tradizionali funzioni Lambda. Segui queste best practice per creare flussi di lavoro affidabili e convenienti.

Scrivi codice deterministico

Durante la riproduzione, la funzione viene eseguita dall'inizio e deve seguire lo stesso percorso di esecuzione dell'esecuzione originale. Il codice al di fuori delle operazioni durevoli deve essere deterministico e produrre gli stessi risultati con gli stessi input.

Suddividi le operazioni non deterministiche in fasi:

- Generazione di numeri casuali e UUIDs
- Ora o timestamp correnti

- Chiamate API esterne e interrogazioni al database
- Operazioni sul file system

TypeScript

```
import { withDurableExecution, DurableContext } from '@aws/durable-execution-sdk-
js';
import { randomUUID } from 'crypto';

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    // Generate transaction ID inside a step
    const transactionId = await context.step('generate-transaction-id', async () =>
 {
      return randomUUID();
    });

    // Use the same ID throughout execution, even during replay
    const payment = await context.step('process-payment', async () => {
      return processPayment(event.amount, transactionId);
    });

    return { statusCode: 200, transactionId, payment };
  }
);
```

Python

```
from aws_durable_execution_sdk_python import durable_execution, DurableContext
import uuid

@durable_execution
def handler(event, context: DurableContext):
    # Generate transaction ID inside a step
    transaction_id = context.step(
        lambda _: str(uuid.uuid4()),
        name='generate-transaction-id'
    )
```

```
    # Use the same ID throughout execution, even during replay
    payment = context.step(
        lambda _: process_payment(event['amount'], transaction_id),
        name='process-payment'
    )

    return {'statusCode': 200, 'transactionId': transaction_id, 'payment': payment}
```

Importante

Non utilizzare variabili o chiusure globali per condividere lo stato tra i passaggi. Passa i dati attraverso i valori restituiti. Lo stato globale si interrompe durante la riproduzione perché i passaggi restituiscono risultati memorizzati nella cache ma le variabili globali vengono reimpostate.

Evita le mutazioni di chiusura: le variabili catturate nelle chiusure possono perdere mutazioni durante la riproduzione. I passaggi restituiscono risultati memorizzati nella cache, ma gli aggiornamenti delle variabili al di fuori del passaggio non vengono riprodotti.

TypeScript

```
// # WRONG: Mutations lost on replay
export const handler = withDurableExecution(async (event, context) => {
  let total = 0;

  for (const item of items) {
    await context.step(async () => {
      total += item.price; // ## Mutation lost on replay!
      return saveItem(item);
    });
  }

  return { total }; // Inconsistent value!
});

// # CORRECT: Accumulate with return values
export const handler = withDurableExecution(async (event, context) => {
  let total = 0;

  for (const item of items) {
    total = await context.step(async () => {
```

```
      const newTotal = total + item.price;
      await saveItem(item);
      return newTotal; // Return updated value
    });
  }

  return { total }; // Consistent!
});

// # EVEN BETTER: Use map for parallel processing
export const handler = withDurableExecution(async (event, context) => {
  const results = await context.map(
    items,
    async (ctx, item) => {
      await ctx.step(async () => saveItem(item));
      return item.price;
    }
  );

  const total = results.getResults().reduce((sum, price) => sum + price, 0);
  return { total };
});
```

Python

```
# # WRONG: Mutations lost on replay
@durable_execution
def handler(event, context: DurableContext):
    total = 0

    for item in items:
        context.step(
            lambda _: save_item_and_mutate(item, total),  # ## Mutation lost on
 replay!
            name=f'save-item-{item["id"]}'
        )

    return {'total': total}  # Inconsistent value!

# # CORRECT: Accumulate with return values
@durable_execution
```

```
def handler(event, context: DurableContext):
    total = 0

    for item in items:
        total = context.step(
            lambda _: save_item_and_return_total(item, total),
            name=f'save-item-{item["id"]}'
        )

    return {'total': total}  # Consistent!

# # EVEN BETTER: Use map for parallel processing
@durable_execution
def handler(event, context: DurableContext):
    def process_item(ctx, item):
        ctx.step(lambda _: save_item(item))
        return item['price']

    results = context.map(items, process_item)
    total = sum(results.get_results())

    return {'total': total}
```

Progettato per l'idempotenza

Le operazioni possono essere eseguite più volte a causa di nuovi tentativi o ripetizioni. Le operazioni non idempotenti causano effetti collaterali duplicati, come addebitare due volte i clienti o inviare più e-mail.

Usa token di idempotenza: genera token all'interno dei passaggi e includili nelle chiamate API esterne per evitare operazioni duplicate.

TypeScript

```
import { withDurableExecution, DurableContext } from '@aws/durable-execution-sdk-
js';

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    // Generate idempotency token once
```

```
    const idempotencyToken = await context.step('generate-idempotency-token', async
 () => {
      return crypto.randomUUID();
    });

    // Use token to prevent duplicate charges
    const charge = await context.step('charge-payment', async () => {
      return paymentService.charge({
        amount: event.amount,
        cardToken: event.cardToken,
        idempotencyKey: idempotencyToken
      });
    });

    return { statusCode: 200, charge };
  }
);
```

Python

```
from aws_durable_execution_sdk_python import durable_execution, DurableContext
import uuid

@durable_execution
def handler(event, context: DurableContext):
    # Generate idempotency token once
    idempotency_token = context.step(
        lambda _: str(uuid.uuid4()),
        name='generate-idempotency-token'
    )

    # Use token to prevent duplicate charges
    def charge_payment(_):
        return payment_service.charge(
            amount=event['amount'],
            card_token=event['cardToken'],
            idempotency_key=idempotency_token
        )

    charge = context.step(charge_payment, name='charge-payment')
```

```
    return {'statusCode': 200, 'charge': charge}
```

Usa la at-most-once semantica: per operazioni critiche che non devono mai essere duplicate (transazioni finanziarie, detrazioni di inventario), configura la modalità di esecuzione. at-most-once

TypeScript

```
// Critical operation that must not duplicate
await context.step('deduct-inventory', async () => {
  return inventoryService.deduct(event.productId, event.quantity);
}, {
  executionMode: 'AT_MOST_ONCE_PER_RETRY'
});
```

Python

```
# Critical operation that must not duplicate
context.step(
    lambda _: inventory_service.deduct(event['productId'], event['quantity']),
    name='deduct-inventory',
    config=StepConfig(execution_mode='AT_MOST_ONCE_PER_RETRY')
)
```

Idempotenza del database: utilizza check-before-write modelli, aggiornamenti condizionali o operazioni di alterazione per evitare la duplicazione dei record.

Gestisci lo stato in modo efficiente

Ogni checkpoint salva lo stato nell'archiviazione persistente. Gli oggetti a stato di grandi dimensioni aumentano i costi, rallentano il checkpoint e influiscono sulle prestazioni. Archivia solo i dati essenziali di coordinamento del flusso di lavoro.

Mantieni lo stato minimo:

- Archivio IDs e riferimenti, non oggetti completi

- Recupera dati dettagliati in pochi passaggi, se necessario
- Usa Amazon S3 o DynamoDB per dati di grandi dimensioni, trasferisci i riferimenti nello stato
- Evita di far passare carichi utili di grandi dimensioni tra i passaggi

TypeScript

```
import { withDurableExecution, DurableContext } from '@aws/durable-execution-sdk-
js';

export const handler = withDurableExecution(
  async (event: any, context: DurableContext) => {
    // Store only the order ID, not the full order object
    const orderId = event.orderId;

    // Fetch data within each step as needed
    await context.step('validate-order', async () => {
      const order = await orderService.getOrder(orderId);
      return validateOrder(order);
    });

    await context.step('process-payment', async () => {
      const order = await orderService.getOrder(orderId);
      return processPayment(order);
    });

    return { statusCode: 200, orderId };
  }
);
```

Python

```
from aws_durable_execution_sdk_python import durable_execution, DurableContext

@durable_execution
def handler(event, context: DurableContext):
    # Store only the order ID, not the full order object
    order_id = event['orderId']
```

```
    # Fetch data within each step as needed
    context.step(
        lambda _: validate_order(order_service.get_order(order_id)),
        name='validate-order'
    )

    context.step(
        lambda _: process_payment(order_service.get_order(order_id)),
        name='process-payment'
    )

    return {'statusCode': 200, 'orderId': order_id}
```

Progetta passaggi efficaci

I gradini sono l'unità di lavoro fondamentale nelle funzioni durevoli. I passaggi ben progettati semplificano la comprensione, il debug e la manutenzione dei flussi di lavoro.

Principi di progettazione Step:

- Usa nomi descrittivi, ad esempio `validate-order` invece di `step1` semplificare la comprensione di log ed errori
- Mantieni i nomi statici: non utilizzare nomi dinamici con timestamp o valori casuali. I nomi delle fasi devono essere deterministici per la riproduzione
- Equilibra la granularità: suddividi le operazioni complesse in fasi mirate, ma evita passaggi troppo piccoli che aumentano il sovraccarico dei checkpoint
- Operazioni relative al gruppo: le operazioni che dovrebbero avere successo o fallire rientrano nella stessa fase

Usa le operazioni di attesa in modo efficiente

Le operazioni di attesa sospendono l'esecuzione senza consumare risorse o incorrere in costi. Usali invece di mantenere Lambda in funzione.

Attese basate sul tempo: utilizzale `context.wait()` per i ritardi anziché o. `setTimeout` `sleep`

Callback esterni: da utilizzare in attesa di `context.waitForCallback()` sistemi esterni. Imposta sempre dei timeout per evitare attese indefinite.

Sondaggi: utilizza `context.waitForCondition()` il backoff esponenziale per interrogare i servizi esterni senza sovraccaricarli.

TypeScript

```
// Wait 24 hours without cost
await context.wait({ seconds: 86400 });

// Wait for external callback with timeout
const result = await context.waitForCallback(
  'external-job',
  async (callbackId) => {
    await externalService.submitJob({
      data: event.data,
      webhookUrl: `https://api.example.com/callbacks/${callbackId}`
    });
  },
  { timeout: { seconds: 3600 } }
);
```

Python

```
# Wait 24 hours without cost
context.wait(86400)

# Wait for external callback with timeout
result = context.wait_for_callback(
    lambda callback_id: external_service.submit_job(
        data=event['data'],
        webhook_url=f'https://api.example.com/callbacks/{callback_id}'
    ),
    name='external-job',
    config=WaitForCallbackConfig(timeout_seconds=3600)
)
```

Ulteriori considerazioni

Gestione degli errori: riprova gli errori temporanei come i timeout di rete e i limiti di velocità. Non riprovare con errori permanenti come errori di input o di autenticazione non validi. Configura le strategie di riprova con un numero massimo di tentativi e tassi di backoff appropriati. Per esempi dettagliati, consulta Gestione degli errori e nuovi tentativi.

Prestazioni: riduci al minimo le dimensioni del checkpoint memorizzando i riferimenti anziché i payload completi. Usa `context.parallel()` ed `context.map()` esegui operazioni indipendenti contemporaneamente. Operazioni relative ai batch per ridurre il sovraccarico dei checkpoint.

Controllo delle versioni: richiama funzioni con numeri di versione o alias per associare le esecuzioni a versioni di codice specifiche. Assicurati che le nuove versioni del codice siano in grado di gestire lo stato delle versioni precedenti. Non rinominate i passaggi né modificate il loro comportamento in modo da interrompere la riproduzione.

Serializzazione: utilizza tipi compatibili con JSON per gli input e i risultati delle operazioni. Converti le date in stringhe ISO e gli oggetti personalizzati in oggetti semplici prima di passarli a operazioni durevoli.

Monitoraggio: abilita la registrazione strutturata con nomi di esecuzione IDs e passaggi. Imposta CloudWatch allarmi per i tassi di errore e la durata dell'esecuzione. Usa il tracciamento per identificare i punti deboli. Per una guida dettagliata, consulta Monitoraggio e debug.

Test: verifica il percorso felice, la gestione degli errori e il comportamento di riproduzione. Prova gli scenari di timeout per callback e attese. Utilizza i test locali per ridurre i tempi di iterazione. Per una guida dettagliata, consulta Testare le funzioni durevoli.

Errori comuni da evitare: non `context.step()` annidate le chiamate, ma utilizzate invece contesti secondari. Suddividi le operazioni non deterministiche in fasi. Imposta sempre i timeout per i callback. Bilancia la granularità dei passaggi con il sovraccarico dei checkpoint. Memorizza i riferimenti anziché gli oggetti di grandi dimensioni nello stato.

Risorse aggiuntive

- Documentazione Python SDK: riferimento completo alle API, modelli di test ed esempi avanzati
- TypeScript Documentazione SDK: riferimento completo alle API, modelli di test ed esempi avanzati

Istanze gestite Lambda

Lambda Managed Instances ti consente di eseguire funzioni Lambda sulle tue istanze Amazon di attuale generazione, tra cui Graviton4, EC2 istanze ottimizzate per la rete e altre opzioni di elaborazione specializzate, senza gestire i cicli di vita delle istanze, le patch di runtime del sistema operativo e del linguaggio, il routing, il bilanciamento del carico o le politiche di scalabilità. Con Lambda Managed Instances, usufruisci di vantaggi in termini di EC2 prezzo, tra cui EC2 Savings Plans e Reserved Instances.

Per un elenco dei tipi di istanze supportati, vai alla pagina AWS Lambda dei prezzi e seleziona la tua. Regione AWS

Funzionalità chiave

Lambda Managed Instances offre le seguenti funzionalità:

- Scegli le istanze adatte: seleziona le istanze appropriate in base ai requisiti di prestazioni e costi, incluso l'accesso alle ultime novità CPUs come Graviton4, rapporti memoria-CPU configurabili e reti a larghezza di banda elevata.
- Provisioning automatico: effettua automaticamente il provisioning delle istanze adatte e attiva gli ambienti di esecuzione delle funzioni. AWS
- Scalabilità dinamica: le istanze si ridimensionano dinamicamente in base ai modelli di traffico delle funzioni.
- Esperienza completamente gestita: AWS gestisce la gestione dell'infrastruttura, il ridimensionamento, l'applicazione di patch e il routing, con le stesse ampie integrazioni event-source che conosci.

Quando usare le istanze gestite Lambda

Prendi in considerazione le istanze gestite Lambda per i seguenti casi d'uso:

- Carichi di lavoro prevedibili in base ai volumi elevati: la soluzione ideale per carichi di lavoro stazionari senza picchi di traffico imprevisti. Le istanze gestite Lambda sono scalabili per gestire il raddoppio del traffico entro cinque minuti per impostazione predefinita.
- Applicazioni critiche per le prestazioni: accesso ai rapporti memoria-CPU più recenti e variabili e a un CPUs throughput di rete elevato

- Requisiti normativi: esigenze di governance granulare con controllo sul VPC e sul posizionamento delle istanze
- Varietà di applicazioni: migrazione delle applicazioni basate sugli eventi, dell' media/data elaborazione, delle applicazioni Web e dei carichi di lavoro legacy alla modalità serverless

Come funziona

Lambda Managed Instances utilizza i provider di capacità come base per l'esecuzione delle tue funzioni:

1. Crea un provider di capacità: definisci dove vengono eseguite le tue funzioni specificando la configurazione del VPC e, facoltativamente, i requisiti dell'istanza e la configurazione di scalabilità
2. Crea la tua funzione: crea le funzioni Lambda come al solito e collegale a un provider di capacità
3. Pubblica una versione della funzione: le versioni delle funzioni diventano attive sulle istanze del provider di capacità una volta pubblicate

Quando pubblichi una versione della funzione con un provider di capacità, Lambda avvia Managed Instances nel tuo account. Per impostazione predefinita, avvia tre istanze per la resilienza AZ e avvia tre ambienti di esecuzione prima di contrassegnare la versione della funzione come ATTIVA. Se colleghi una funzione a un provider di capacità esistente che esegue già altre funzioni, Lambda potrebbe non generare nuove istanze se le istanze disponibili hanno già la capacità di adattarsi agli ambienti di esecuzione della nuova funzione.

Modello di concorrenza

Le istanze gestite Lambda supportano le chiamate simultanee, in cui un ambiente di esecuzione può gestire più chiamate contemporaneamente. Ciò differisce dal tipo di calcolo Lambda (predefinito), che fornisce un unico modello di concorrenza in cui un ambiente di esecuzione può eseguire al massimo una chiamata alla volta. La concorrenza multipla consente un migliore utilizzo delle EC2 istanze sottostanti ed è particolarmente utile per le applicazioni che richiedono un elevato livello di I/O come i servizi Web o i processi in batch. Questa modifica del modello di esecuzione significa che la sicurezza dei thread, la gestione dello stato e l'isolamento del contesto devono essere gestiti in modo diverso a seconda del runtime.

Locazione e isolamento

Il tipo di elaborazione Lambda (predefinito) è multi-tenant e utilizza la tecnologia Firecracker MicroVM per fornire isolamento tra gli ambienti di esecuzione in esecuzione su flotte Lambda condivise. Le istanze gestite Lambda vengono eseguite nel tuo account e forniscono l' EC2 hardware e le opzioni di prezzo più recenti. Le istanze gestite utilizzano contenitori in esecuzione su istanze EC2 Nitro per fornire isolamento anziché Firecracker. I provider di capacità fungono da limite di sicurezza per le funzioni Lambda. Le funzioni vengono eseguite in contenitori all'interno delle istanze.

Comprensione delle istanze gestite

Le funzioni Lambda Managed Instances vengono eseguite sulle istanze EC2 gestite del tuo account. Queste istanze sono gestite completamente da Lambda, il che significa che hai autorizzazioni limitate su di esse rispetto alle istanze standard. EC2 Puoi identificare le istanze gestite Lambda nel tuo account tramite:

- La presenza del `Operator` campo nell'output EC2 `DescribeInstances`
- Il `aws:lambda:capacity-provider` tag sull'istanza

Non è possibile eseguire EC2 operazioni standard direttamente su queste istanze, ad esempio interromperle manualmente. Per eliminare le istanze gestite, elimina il provider di capacità associato. Lambda interromperà quindi le istanze come parte del processo di eliminazione del provider di capacità.

Prezzi

Lambda Managed Instances utilizza prezzi EC2 basati su una commissione di gestione del 15% in aggiunta al costo dell' EC2 istanza. Questo modello di prezzo supporta EC2 Savings Plans, Reserved Instances e qualsiasi altro sconto sui prezzi applicato all' EC2 utilizzo. Per ulteriori dettagli, consulta la pagina dei prezzi: https://aws.amazon.com/lambda/ prezzi/

Importante: gli sconti EC2 sui prezzi si applicano solo all' EC2 elaborazione sottostante, non alla commissione di gestione.

In che modo le istanze gestite Lambda differiscono dal tipo di calcolo Lambda (predefinito)

Lambda Managed Instances modifica il modo in cui Lambda elabora le richieste rispetto a Lambda (impostazione predefinita).

Principali differenze:

	Lambda (impostazione predefinita)	Istanze gestite Lambda
Modello di concorrenza	Modello di concorrenza singolo in cui un ambiente di esecuzione può supportare al massimo una chiamata alla volta	Richiamazioni simultanee in cui un unico ambiente di esecuzione è in grado di gestire più chiamate contemporaneamente, aumentando la produttività soprattutto per le applicazioni che richiedono un elevato livello di I/O
Locazione e isolamento	Multi-tenant, che utilizza la tecnologia Firecracker MicroVM per fornire isolamento tra gli ambienti di esecuzione in esecuzione su flotte Lambda condivise	Esegui nel tuo account, utilizzando Nitro per fornire isolamento. EC2 I provider di capacità fungono da limite di sicurezza, con funzioni eseguite in contenitori all'interno delle istanze
Modello tariffario	Tariffe relative alla durata della richiesta	Prezzi basati su istanze con modelli di EC2 prezzo, tra cui istanze On-Demand e riservate , e opzioni di risparmio come Compute Savings Plans
Comportamento del dimensionamento	Scalabilità quando non esiste un ambiente di esecuzione libero per gestire una chiamata in entrata (avvio a freddo). Scalabilità fino a zero senza traffico	Scalabilità asincrona in base solo all'utilizzo delle risorse della CPU, senza avviamenti a freddo. Scalabilità fino ad ambienti di esecuzione minimi configurati senza traffico

	Lambda (impostazione predefinita)	Istanze gestite Lambda
Ideale per	Funzioni con traffico intenso in grado di gestire alcuni orari di avvio a freddo o applicazioni senza carico sostenuto che traggono vantaggio dalla scalabilità a zero	Funzioni di traffico prevedibile ad alto volume quando si desidera la flessibilità, i piani tariffari e le opzioni hardware di EC2

Fasi successive

- Scopri i provider di capacità per le istanze gestite Lambda
- Comprendi la scalabilità per le istanze gestite Lambda
- Consulta le guide specifiche di runtime per Java, Node.js e Python
- Configura la connettività VPC per i tuoi provider di capacità
- Comprendi la sicurezza e le autorizzazioni per le istanze gestite Lambda

Guida introduttiva alle istanze gestite Lambda

Creazione di una funzione Lambda Managed Instance (console)

Puoi utilizzare la console Lambda per creare una funzione di istanza gestita che viene eseguita su EC2 istanze Amazon gestite da un provider di capacità.

Importante: prima di creare una funzione di istanza gestita, devi prima creare un provider di capacità. Queste funzioni richiedono che un provider di capacità definisca l' EC2 infrastruttura Amazon che eseguirà le tue funzioni.

Per creare una funzione Lambda Managed Instance (console)

1. Apri la console Lambda.
2. Scegli Capacity providers dal riquadro di navigazione a sinistra.
3. Scegli Crea fornitore di capacità.
4. Nella sezione Impostazioni del fornitore di capacità, inserisci un nome per il tuo fornitore di capacità.

5. Seleziona VPC e autorizzazioni per il tuo provider di capacità. Puoi usarne uno esistente o crearne uno nuovo. Per informazioni sulla creazione del ruolo operatore richiesto, consulta Ruolo operatore Lambda per istanze gestite Lambda.
6. Espandere Advanced settings (Impostazioni avanzate).
7. Definisci i requisiti dell'istanza scegliendo l'architettura del processore e i tipi di istanza.
8. In Auto scaling, specifica il numero massimo di EC2 v CPUs per il tuo provider di capacità. Puoi anche scegliere la modalità di ridimensionamento manuale dell'istanza per impostare il tuo valore di ridimensionamento per un controllo preciso.
9. Scegli Create Capacity Provider per crearne uno nuovo.
10. Quindi, scegli Crea funzione.
11. Scegli Crea da zero.
12. Nel riquadro Informazioni di base, fornisci un nome per la funzione.
13. Per Runtime, scegli uno dei Runtime supportati.
14. Scegliete l'architettura per la vostra funzione (la stessa che avete selezionato per il provider di capacità). Per impostazione predefinita, x86_64.
15. In Autorizzazioni, assicurati di disporre dell'autorizzazione per il ruolo di esecuzione scelto. Altrimenti, puoi creare un nuovo ruolo.
16. In Configurazioni aggiuntive, scegli il tipo di elaborazione come Istanze gestite Lambda.
17. L'ARN del fornitore di capacità creato nei passaggi precedenti deve essere preselezionato.
18. Scegli la dimensione della memoria e la memoria dell'ambiente di esecuzione (GiB) per rapporto vCPU.
19. Scegli Crea funzione.

La funzione Lambda Managed Instance viene creata e fornirà capacità al provider di capacità specificato. La creazione della funzione richiede in genere diversi minuti. Una volta completata, è possibile modificare il codice della funzione ed eseguire il primo test.

Creazione di una funzione Lambda Managed Instance (CLI AWS)

Prerequisiti

Prima di iniziare, assicurati di disporre di:

- AWS CLI: installa e configura la CLI AWS . Per ulteriori informazioni, consulta Installazione o aggiornamento della versione più recente della AWS CLI.

- Autorizzazioni IAM: il tuo utente o ruolo IAM deve disporre delle autorizzazioni per creare funzioni Lambda, fornitori di capacità e passare ruoli IAM. Tieni presente che ti servirà anche `iam:CreateServiceLinkedRole` se è la prima volta che crei un provider di capacità nell'account o se il Service Linked Role (SLR) è stato eliminato.

Fase 1: Creare i ruoli IAM richiesti

Le istanze gestite Lambda richiedono due ruoli IAM: un ruolo di esecuzione per la funzione e un ruolo di operatore per il provider di capacità. Il ruolo di operatore consente a Lambda di avviare, terminare e monitorare le EC2 istanze Amazon per tuo conto. Il ruolo di esecuzione della funzione concede alla funzione le autorizzazioni per accedere ad altri servizi e risorse. AWS

Per creare il ruolo di esecuzione Lambda

1. Crea un documento sulla politica di fiducia che consenta a Lambda di assumere il ruolo:

```
cat > lambda-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

2. Crea il ruolo di esecuzione:

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

3. Allega la politica di esecuzione di base:

```
aws iam attach-role-policy \
  --role-name MyLambdaExecutionRole \
```

```
  --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

Per creare il ruolo di gestore di capacità

1. Crea un documento sulla politica di fiducia che consenta a Lambda di assumere il ruolo di operatore:

```
cat > operator-trust-policy.json << 'EOF'
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

2. Crea il ruolo dell'operatore:

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

3. Allega la politica di EC2 autorizzazioni richiesta:

```
aws iam attach-role-policy \
  --role-name MyCapacityProviderOperatorRole \
  --policy-arn arn:aws:iam::aws:policy/AWSLambdaManagedEC2ResourceOperator
```

Fase 2: Configurazione delle risorse VPC

Le istanze gestite Lambda vengono eseguite nel tuo VPC e richiedono una sottorete e un gruppo di sicurezza.

Per creare risorse VPC

1. Crea un VPC:

```
VPC_ID=$(aws ec2 create-vpc \
  --cidr-block 10.0.0.0/16 \
  --query 'Vpc.VpcId' \
  --output text)
```

2. Crea una sottorete:

```
SUBNET_ID=$(aws ec2 create-subnet \
  --vpc-id $VPC_ID \
  --cidr-block 10.0.1.0/24 \
  --query 'Subnet.SubnetId' \
  --output text)
```

3. Crea un gruppo di sicurezza:

```
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)
```

Nota: le funzioni Lambda Managed Instances richiedono la configurazione VPC per accedere alle risorse esterne al VPC e trasmettere i dati di telemetria a Logs e X-Ray. CloudWatch Per i dettagli sulla configurazione, consulta Networking for Lambda Managed Instances.

Fase 3: creazione di un provider di capacità

Un provider di capacità gestisce le EC2 istanze che eseguono le funzioni Lambda.

Come creare un provider di capacità

```
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
```

Questo comando crea un provider di capacità con la seguente configurazione:

- Configurazione VPC: specifica la sottorete e il gruppo di sicurezza per le istanze EC2
- Autorizzazioni: definisce il ruolo IAM utilizzato da Lambda per gestire le istanze EC2
- Requisiti dell'istanza: specifica l'architettura x86_64
- Configurazione di scalabilità: imposta un massimo di 30 v per il provider di capacità CPUs

Fase 4: Creare una funzione Lambda con codice in linea

Per creare una funzione con codice in linea

1. Innanzitutto, crea una semplice funzione Python e impacchettala in linea:

```
# 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. Crea la funzione Lambda utilizzando il file ZIP in linea:

```
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}
```

La funzione viene creata con:

- Runtime — Python 3.13
- Handler — La funzione in `lambda_handler` `lambda_function.py`
- Memoria: 2048 MB
- Archiviazione effimera: 512 MB
- Provider di capacità: collegamenti al provider di capacità creato

Fase 5: Pubblicare una versione della funzione

Per eseguire la funzione su Lambda Managed Instances, devi pubblicare una versione.

Per pubblicare una versione della funzione

```
aws lambda publish-version \
  --function-name my-managed-instance-function
```

Questo comando pubblica la versione 1 della funzione e la distribuisce al provider di capacità.

Passaggio 6: richiama la funzione

Dopo la pubblicazione, puoi richiamare la tua funzione.

Per richiamare la tua funzione

```
aws lambda invoke \
```

```
  --function-name my-managed-instance-function:1 \
  --payload '{"name": "World"}' \
  response.json

# View the response
cat response.json
```

La funzione viene eseguita sulle EC2 istanze gestite dal fornitore di capacità e restituisce una risposta.

Eliminazione

Per evitare di incorrere in addebiti, elimina le risorse che hai creato:

1. Eliminare la funzione:

```
aws lambda delete-function --function-name my-managed-instance-function
```

2. Eliminare il fornitore di capacità:

```
aws lambda delete-capacity-provider --capacity-provider-name my-capacity-provider
```

3. Eliminare le risorse 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. Elimina i ruoli 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
```

Concetti principali

Lambda Managed Instances introduce diversi concetti fondamentali che differiscono dalle tradizionali funzioni Lambda. La comprensione di questi concetti è essenziale per implementare e gestire efficacemente le funzioni sull'infrastruttura. EC2

I fornitori di capacità costituiscono la base delle istanze gestite Lambda. Un provider di capacità definisce l'infrastruttura di elaborazione su cui vengono eseguite le tue funzioni, inclusa la configurazione del VPC, i requisiti delle istanze e le politiche di scalabilità. I provider di capacità fungono anche da limite di sicurezza per le vostre funzioni, il che significa che tutte le funzioni assegnate allo stesso fornitore di capacità devono essere considerate reciprocamente affidabili.

Il comportamento di scalabilità differisce in modo significativo dalle tradizionali funzioni Lambda. Invece di scalare su richiesta all'arrivo delle chiamate, le istanze gestite si scalano in modo asincrono in base all'utilizzo delle risorse della CPU. Questo approccio elimina le partenze a freddo ma richiede una pianificazione della crescita del traffico. Se il traffico raddoppia più che nel giro di 5 minuti, potresti riscontrare rallentamenti man mano che Lambda aumenta la capacità per soddisfare la domanda.

La sicurezza e le autorizzazioni richiedono un'attenta considerazione. Sono necessarie le autorizzazioni del ruolo di operatore per consentire a Lambda di EC2 gestire le risorse nei tuoi provider di capacità. Inoltre, gli utenti hanno bisogno dell'`lambda:PassCapacityProvider`autorizzazione per assegnare funzioni ai fornitori di capacità, che fungono da barriera di sicurezza per controllare quali funzioni possono essere eseguite su un'infrastruttura specifica.

L'esecuzione simultanea è una caratteristica fondamentale delle istanze gestite. Ogni ambiente di esecuzione è in grado di gestire più chiamate contemporaneamente, massimizzando l'utilizzo delle risorse per le applicazioni che richiedono un elevato livello di I/O. Ciò differisce dalla Lambda tradizionale in cui ogni ambiente elabora una richiesta alla volta. Questo modello di esecuzione richiede attenzione alla sicurezza dei thread, alla gestione dello stato e all'isolamento del contesto a seconda del runtime.

Le seguenti sezioni forniscono informazioni dettagliate su ogni concetto fondamentale.

fornitori di capacità

Un provider di capacità è la base per l'esecuzione di istanze gestite Lambda. Funge da limite di sicurezza per le tue funzioni e definisce le risorse di calcolo che Lambda fornirà e gestirà per tuo conto.

Quando crei un fornitore di capacità, specifichi:

- Configurazione VPC: le sottoreti e i gruppi di sicurezza in cui verranno eseguite le istanze
- Autorizzazioni: ruoli IAM per Lambda per la gestione delle risorse EC2
- Requisiti dell'istanza (facoltativi): preferenze relative all'architettura e al tipo di istanza
- Configurazione di scalabilità (opzionale): in che modo Lambda ridimensiona le tue istanze

Comprendere i provider di capacità come limite di sicurezza

I provider di capacità fungono da limite di sicurezza per le funzioni Lambda all'interno del VPC, sostituendo l'isolamento basato su Firecracker. Le funzioni vengono eseguite in contenitori all'interno delle istanze, ma i contenitori non forniscono un forte isolamento di sicurezza tra le funzioni, a differenza di Firecracker micro. VMs

Concetti chiave sulla sicurezza:

- Capacity Provider: il limite di sicurezza che definisce i livelli di fiducia per le funzioni Lambda
- Isolamento dei container: i container NON sono un fornitore di sicurezza: non fate affidamento su di essi per la sicurezza tra carichi di lavoro non affidabili
- Trust Separation: separa i carichi di lavoro che non sono reciprocamente affidabili utilizzando diversi provider di capacità

Creazione di un provider di capacità

È possibile creare un provider di capacità utilizzando la AWS CLI, la console di AWS gestione o. AWS SDKs

Utilizzo della 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
```

Parametri obbligatori

CapacityProviderName

- Un nome univoco per il tuo fornitore di capacità
- Deve essere univoco all'interno del tuo AWS account

VpcConfig

- SubnetIds(obbligatorio): Almeno una sottorete, massimo 16. Utilizza sottoreti in più zone di disponibilità per garantire la resilienza
- SecurityGroupIds(opzionale): gruppi di sicurezza per le tue istanze. Il valore predefinito è il gruppo di sicurezza VPC predefinito se non specificato

PermissionsConfig

- CapacityProviderOperatorRoleArn(richiesto): ruolo IAM che consente a Lambda di gestire EC2 le risorse nel tuo provider di capacità

Parametri facoltativi

InstanceRequirements

Specificate l'architettura e i tipi di istanza per il vostro provider di capacità:

- Architetture: scegli `x86_64` o. `arm64` L'impostazione predefinita è `x86_64`
- AllowedInstanceTypes: Specificate i tipi di istanza consentiti. Ad esempio: `m5.8xlarge`
- ExcludedInstanceTypes: Specificate i tipi di istanza esclusi utilizzando i caratteri jolly. È possibile specificare solo uno dei o AllowedInstanceTypes ExcludedInstanceTypes

Per impostazione predefinita, Lambda sceglie i tipi di istanza migliori per il tuo carico di lavoro. Ti consigliamo di lasciare che le istanze gestite da Lambda scelgano i tipi di istanza per te, poiché limitare il numero di tipi di istanze possibili può comportare una minore disponibilità.

CapacityProviderScalingConfig

Configura il modo in cui Lambda ridimensiona le tue istanze:

- ScalingMode: impostato per il ridimensionamento automatico o `Auto` `Manual` per il controllo manuale. L'impostazione predefinita è `Auto`
- VCpuNumero massimo: numero massimo di v CPUs per il provider di capacità. Il valore predefinito è 400.
- ScalingPolicies: definisci le politiche di scalabilità del tracciamento degli obiettivi per l'utilizzo di CPU e memoria

KmsKeyArn

Specificare una AWS KMS chiave per la crittografia EBS. Il valore predefinito è la chiave AWS gestita se non specificata.

Tag

Aggiungi tag per organizzare e gestire i tuoi capacity provider.

Gestione dei fornitori di capacità

Aggiornare un provider di capacità

È possibile aggiornare determinate proprietà di un provider di capacità utilizzando l'`UpdateCapacityProvider`API.

```
aws lambda update-capacity-provider \
  --capacity-provider-name my-capacity-provider \
  --capacity-provider-scaling-config ScalingMode=Auto
```

Eliminare un provider di capacità

Puoi eliminare un fornitore di capacità quando non è più necessario utilizzando l'`DeleteCapacityProvider`API.

```
aws lambda delete-capacity-provider \
```

```
  --capacity-provider-name my-capacity-provider
```

Nota: non è possibile eliminare un fornitore di capacità a cui sono associate versioni di funzioni.

Visualizzazione dei dettagli del fornitore di capacità

Recupera informazioni su un fornitore di capacità utilizzando l'`GetCapacityProvider`API.

```
aws lambda get-capacity-provider \
  --capacity-provider-name my-capacity-provider
```

Stati del fornitore di capacità

Un provider di capacità può trovarsi in uno dei seguenti stati:

- In sospeso: il provider di capacità è in fase di creazione
- Attivo: il provider di capacità è pronto per l'uso
- Fallita: la creazione del provider di capacità non è riuscita
- Eliminazione: il provider di capacità viene eliminato

Quote

- Fornitori di capacità massima per account: 1.000
- Numero massimo di versioni funzionali per provider di capacità: 100 (non può essere aumentato)

Best practice

1. Separati per livello di affidabilità: crea diversi provider di capacità per carichi di lavoro con requisiti di sicurezza diversi
2. Usa nomi descrittivi: nomina i fornitori di capacità per indicare chiaramente l'uso previsto e il livello di affidabilità (ad esempio,`production-trusted`) `dev-sandbox`
3. Utilizza più zone di disponibilità: specifica le sottoreti su più AZs zone per un'elevata disponibilità
4. Consenti a Lambda di scegliere i tipi di istanza: a meno che tu non abbia requisiti hardware specifici, consenti a Lambda di selezionare i tipi di istanza migliori per una disponibilità ottimale
5. Monitora l'utilizzo: AWS CloudTrail da utilizzare per monitorare le assegnazioni e i modelli di accesso dei provider di capacità

Fasi successive

- Scopri come scalare le istanze gestite Lambda
- Comprendi la sicurezza e le autorizzazioni per le istanze gestite Lambda
- Configura la connettività VPC per i tuoi provider di capacità
- Consulta le guide specifiche di runtime per Java, Node.js e Python

Scalabilità delle istanze gestite Lambda

Le istanze gestite Lambda non sono scalabili all'arrivo delle chiamate e non supportano gli avviamenti a freddo. Al contrario, è scalabile in modo asincrono utilizzando segnali di consumo delle risorse. Le istanze gestite attualmente sono scalabili in base all'utilizzo delle risorse della CPU e alla saturazione di più valute.

Principali differenze:

- Lambda (impostazione predefinita): scala quando non esiste un ambiente di esecuzione libero per gestire una chiamata in entrata (avvio a freddo)
- Istanze gestite Lambda: scalabilità asincrona in base all'utilizzo delle risorse della CPU e alla saturazione multiconcorrenza degli ambienti di esecuzione

Se il traffico raddoppia più che nel giro di 5 minuti, potresti riscontrare rallentamenti man mano che Lambda ridimensiona le istanze e gli ambienti di esecuzione per soddisfare la domanda.

Il ciclo di vita della scalabilità

Lambda Managed Instances utilizza un'architettura distribuita per gestire la scalabilità:

Componenti:

- Istanze gestite: esegui nel tuo account nelle sottoreti che fornisci
- Router e scaler: componenti Lambda condivisi che indirizzano le chiamate e gestiscono la scalabilità
- Lambda Agent: viene eseguito su ogni istanza gestita per gestire il ciclo di vita dell'ambiente di esecuzione e monitorare il consumo di risorse

Come funziona:

1. Quando pubblichi una versione della funzione con un provider di capacità, Lambda avvia Managed Instances nel tuo account. Per impostazione predefinita, ne avvia tre per la resilienza AZ e avvia tre ambienti di esecuzione prima di contrassegnare la versione della funzione come ATTIVA.
2. Ogni istanza gestita può eseguire ambienti di esecuzione per più funzioni mappate allo stesso provider di capacità.
3. Man mano che il traffico fluisce verso l'applicazione, gli ambienti di esecuzione consumano risorse. L'agente Lambda invia una notifica allo Scaler, che decide se scalare nuovi ambienti di esecuzione o istanze gestite.
4. Se Router tenta di inviare una chiamata a un ambiente di esecuzione con un elevato consumo di risorse, l'agente Lambda su quell'istanza gli notifica di riprovare su un'altra istanza.
5. Quando il traffico diminuisce, Lambda Agent invia una notifica a Scaler, che decide di ridimensionare gli ambienti di esecuzione e scalare nelle istanze gestite.

Regolazione del comportamento di scalabilità

È possibile personalizzare il comportamento di scalabilità delle istanze gestite tramite quattro controlli:

Controlli a livello di funzione

1. Memoria delle funzioni e v CPUs

Scegli la dimensione della memoria e l'allocazione della vCPU per la tua funzione. La dimensione minima della funzione supportata è 2 GB e 1 vCPU.

Considerazioni:

- Scegli un'impostazione di memoria e vCPU che supporti le esecuzioni simultanee della tua funzione
- Non è possibile configurare una funzione con meno di 1 vCPU, poiché le funzioni in esecuzione su istanze gestite dovrebbero supportare carichi di lavoro multi-concorrenti.
- Non puoi scegliere meno di 2 GB perché corrisponde al rapporto 2 a 1 tra memoria e vCPU delle istanze c, che hanno il rapporto più basso
- Per le applicazioni Python, potrebbe essere necessario scegliere un rapporto più elevato tra memoria e vCPUs, ad esempio 4 a 1 o 8 a 1, a causa del modo in cui Python gestisce la multi-concorrenza

- Se si eseguono operazioni che richiedono un uso intensivo della CPU o si eseguono operazioni di I/O limitate, è consigliabile scegliere più di una vCPU

2. Simultaneità massima

Imposta la concorrenza massima per ambiente di esecuzione.

Comportamento predefinito: Lambda sceglie impostazioni predefinite ragionevoli che bilanciano il consumo di risorse e la velocità effettiva, adatte a un'ampia varietà di applicazioni.

Linee guida per la regolazione:

- Aumentare la concorrenza: se le chiamate alle funzioni utilizzano pochissima CPU, è possibile aumentare la concorrenza massima fino a un massimo di 64 per vCPU
- Riduzione della concorrenza: se l'applicazione consuma una grande quantità di memoria e pochissima CPU, è possibile ridurre la concorrenza massima

Importante: poiché le istanze gestite Lambda sono pensate per applicazioni multi-concorrenti, gli ambienti di esecuzione con una concorrenza molto bassa possono presentare problemi di scalabilità.

Controlli a livello di provider di capacità

3. Obiettivo di utilizzo delle risorse

Scegli il tuo obiettivo per il consumo di utilizzo della CPU.

Comportamento predefinito: Lambda mantiene un margine di manovra sufficiente per raddoppiare il traffico entro 5 minuti senza acceleratori.

Opzioni di ottimizzazione:

- Se il carico di lavoro è molto costante o se l'applicazione non è sensibile alle limitazioni, è possibile impostare l'obiettivo su un livello elevato per ottenere un maggiore utilizzo e ridurre i costi
- Se si desidera mantenere un margine di manovra sufficiente in caso di picchi di traffico, è possibile impostare obiettivi di risorse a un livello basso, il che richiederà una maggiore capacità

4. Selezione del tipo di istanza

Imposta i tipi di istanza consentiti o esclusi.

Comportamento predefinito: Lambda sceglie i tipi di istanza migliori per il tuo carico di lavoro. Ti consigliamo di lasciare che le istanze gestite da Lambda scelgano i tipi di istanza per te, poiché limitare il numero di tipi di istanze possibili può comportare una minore disponibilità.

Configurazione personalizzata:

- Requisiti hardware specifici: imposta i tipi di istanze consentiti su un elenco di istanze compatibili. Ad esempio, se disponi di un'applicazione che richiede un'elevata larghezza di banda di rete, puoi selezionare diversi tipi di istanze n
- Ottimizzazione dei costi: per gli ambienti di test o sviluppo, puoi scegliere tipi di istanze più piccoli, come i tipi di istanze m7a.large

Fasi successive

- Scopri i provider di capacità per le istanze gestite Lambda
- Consulta le guide specifiche per il runtime per la gestione della concorrenza multipla
- Configura la connettività VPC per i tuoi provider di capacità
- Monitora le metriche di scalabilità per ottimizzare il comportamento di scalabilità

Sicurezza e autorizzazioni

Le istanze gestite Lambda utilizzano i provider di capacità come limiti di fiducia. Le funzioni vengono eseguite in contenitori all'interno di queste istanze, ma i contenitori non forniscono un isolamento di sicurezza tra i carichi di lavoro. Tutte le funzioni assegnate allo stesso provider di capacità devono essere considerate affidabili da entrambe le parti.

Concetti chiave di sicurezza

- Capacity Provider: il limite di sicurezza che definisce i livelli di fiducia per le funzioni Lambda
- Isolamento dei container: i container non sono un limite di sicurezza: non fate affidamento su di essi per la sicurezza tra carichi di lavoro non affidabili
- Trust Separation: separa i carichi di lavoro che non sono reciprocamente affidabili utilizzando diversi provider di capacità

Autorizzazioni richieste

PassCapacityProvider Azione

Gli utenti devono essere `lambda:PassCapacityProvider` autorizzati ad assegnare funzioni ai fornitori di capacità. Questa autorizzazione funge da barriera di sicurezza, garantendo che solo gli utenti autorizzati possano collocare funzioni in fornitori di capacità specifiche.

Gli amministratori degli account controllano quali funzioni possono utilizzare fornitori di capacità specifici tramite l'azione `lambda:PassCapacityProvider` IAM. Questa azione è necessaria quando:

- Creazione di funzioni che utilizzano istanze gestite Lambda
- Aggiornamento delle configurazioni delle funzioni per utilizzare un provider di capacità
- Implementazione di funzioni tramite infrastruttura come codice

Esempio di politica IAM

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "lambda:PassCapacityProvider",
      "Resource": "arn:aws:lambda:*:*:capacity-provider:trusted-workloads-*"
    }
  ]
}
```

Ruolo collegato ai servizi

AWS Lambda utilizza il ruolo `AWSServiceRoleForLambda` collegato ai servizi per gestire le risorse ec2 di Lambda Managed Instances nei tuoi provider di capacità.

Best practice

1. Separati per livello di fiducia: crea diversi provider di capacità per carichi di lavoro con requisiti di sicurezza diversi
2. Usa nomi descrittivi: nomina i fornitori di capacità per indicare chiaramente l'uso previsto e il livello di affidabilità (ad esempio,`production-trusted`) `dev-sandbox`

3. Applica il privilegio minimo: concedi `PassCapacityProvider` le autorizzazioni solo ai fornitori di capacità necessari
4. Monitora l'utilizzo: AWS CloudTrail da utilizzare per monitorare le assegnazioni e i modelli di accesso dei fornitori di capacità

Fasi successive

- Scopri i provider di capacità per le istanze gestite Lambda
- Comprendi la scalabilità per le istanze gestite Lambda
- Configura la connettività VPC per i tuoi provider di capacità
- Consulta le guide specifiche di runtime per Java, Node.js e Python

Ruolo dell'operatore Lambda per le istanze gestite Lambda

Quando utilizzi Lambda Managed Instances, Lambda necessita delle autorizzazioni per gestire la capacità di elaborazione nel tuo account. Il ruolo operatore fornisce queste autorizzazioni tramite policy IAM che consentono a Lambda di EC2 gestire le istanze nel provider di capacità.

Lambda assume il ruolo di operatore durante l'esecuzione di queste operazioni di gestione, in modo simile a come Lambda assume un ruolo di esecuzione durante l'esecuzione della funzione.

Creazione di un ruolo di operatore

Puoi creare un ruolo operatore nella console IAM o con la AWS CLI. Il ruolo deve includere:

- Politica delle autorizzazioni: concede le autorizzazioni per gestire i provider di capacità e le risorse associate
- Politica di fiducia: consente al servizio Lambda (`lambda.amazonaws.com`) di assumere il ruolo

Policy delle autorizzazioni

Il ruolo di operatore richiede le autorizzazioni per gestire i provider di capacità e le risorse di elaborazione sottostanti. Come minimo, il ruolo richiede le autorizzazioni previste nella policy AWSLambdaEC2ResourceOperatorgestita, attualmente:

```
{
  "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"
      ],
      "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"
        }
      }
    }
  ]
}
```

Policy di attendibilità

La policy di fiducia consente a Lambda di assumere il ruolo di operatore:

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

Ruolo collegato ai servizi per le istanze gestite Lambda

Per gestire in modo responsabile il ciclo di vita delle istanze gestite Lambda, Lambda richiede un accesso persistente per terminare le istanze gestite nel tuo account. Lambda utilizza un ruolo collegato al servizio (SLR) AWS Identity and Access Management (IAM) per eseguire queste operazioni.

Creazione automatica: il ruolo collegato al servizio viene creato automaticamente la prima volta che si crea un provider di capacità. L'utente che crea il primo fornitore di capacità deve disporre dell'`iam:CreateServiceLinkedRole`autorizzazione per il `lambda.amazonaws.com` principale.

Autorizzazioni: il ruolo collegato al servizio concede a Lambda le seguenti autorizzazioni sulle istanze gestite:

- `ec2:TerminateInstances`— Per terminare le istanze alla fine del loro ciclo di vita
- `ec2:DescribeInstances`— Per enumerare le istanze gestite

Eliminazione: puoi eliminare questo ruolo collegato al servizio solo dopo aver eliminato tutti i provider di capacità di Lambda Managed Instances nel tuo account.

Per ulteriori informazioni sui ruoli collegati al servizio, consulta the section called “Uso di ruoli collegati ai servizi”.

Comprendere l'ambiente di esecuzione di Lambda Managed Instances

Le istanze gestite Lambda forniscono un modello di distribuzione alternativo che esegue il codice funzionale su istanze Amazon di proprietà del cliente EC2 mentre Lambda gestisce gli aspetti operativi. L'ambiente di esecuzione per le istanze gestite presenta diverse differenze importanti rispetto alle funzioni Lambda (predefinite), in particolare nel modo in cui gestisce le chiamate simultanee e gestisce i cicli di vita dei container.

Nota: per informazioni sull'ambiente di esecuzione Lambda (predefinito), vedere Comprendere il ciclo di vita dell'ambiente di esecuzione Lambda.

Ciclo di vita dell'ambiente di esecuzione

Il ciclo di vita di un ambiente di esecuzione delle funzioni Lambda Managed Instances differisce da Lambda (impostazione predefinita) in diversi modi chiave:

Fase di init

Durante la fase di inizializzazione, Lambda esegue le seguenti operazioni:

- Inizializza e registra tutte le estensioni
- Avvia il punto di ingresso del runtime. Il runtime genera il numero configurato di runtime worker (l'implementazione dipende dal runtime)

- Codice di inizializzazione della funzione Run (codice esterno al gestore)
- Attendi che almeno un runtime worker segnali la disponibilità chiamando `/runtime/invocation/next`

La fase di inizializzazione è considerata completa quando le estensioni sono state inizializzate e almeno un runtime worker ha chiamato. `/runtime/invocation/next` La funzione è quindi pronta per elaborare le chiamate.

> **Note**
>
> Per le funzioni Lambda Managed Instances, l'inizializzazione può richiedere fino a 15 minuti. Il timeout è il massimo di 130 secondi o il timeout della funzione configurata (fino a 900 secondi).

Invoca fase

La fase Invoke per le funzioni Lambda Managed Instances presenta diverse caratteristiche uniche:

Funzionamento continuo. A differenza di Lambda (impostazione predefinita), l'ambiente di esecuzione rimane continuamente attivo ed elabora le chiamate non appena arrivano senza bloccarsi tra una chiamata e l'altra.

Elaborazione parallela. È possibile eseguire più chiamate contemporaneamente all'interno dello stesso ambiente di esecuzione, ciascuna gestita da un runtime worker diverso.

Timeout indipendenti. Il timeout configurato della funzione si applica a ogni singola chiamata. Quando scade una chiamata, Lambda contrassegna quella chiamata specifica come fallita ma non interrompe le altre chiamate in esecuzione né termina l'ambiente di esecuzione.

Gestione della contropressione. Se tutti i runtime worker sono impegnati nell'elaborazione delle chiamate, le nuove richieste di chiamata vengono rifiutate fino a quando un worker non diventa disponibile.

Gestione e ripristino degli errori

La gestione degli errori negli ambienti di esecuzione delle funzioni Lambda Managed Instances è diversa da Lambda (impostazione predefinita):

Richiama i timeout. Quando scade una singola chiamata, Lambda restituisce un errore di timeout per quella chiamata. Tuttavia, Lambda Managed Instances non impone il timeout: il codice continuerà a funzionare. In qualità di sviluppatore di funzioni, sei responsabile del rilevamento e della gestione del timeout. L'oggetto context espone il tempo rimanente per l'invocazione, con un valore zero o negativo che indica un timeout. Le altre chiamate simultanee nell'ambiente di esecuzione continuano l'elaborazione normalmente.

Errori dei Runtime Worker. Se un processo di runtime worker si blocca, l'ambiente di esecuzione continua a funzionare con i restanti lavoratori sani.

L'estensione si blocca. Se un processo di estensione si blocca durante l'inizializzazione o l'operazione, l'intero ambiente di esecuzione viene contrassegnato come non integro e viene terminato. Lambda crea un nuovo ambiente di esecuzione per sostituirlo.

Nessun ripristino o riparazione. A differenza di Lambda (impostazione predefinita), le istanze gestite non tentano di reimpostare e reinizializzare l'ambiente di esecuzione dopo errori. Invece, i contenitori non integri vengono chiusi e sostituiti con altri nuovi.

Versione $LATEST.PUBLISHED in Lambda Managed Instances

Le funzioni Lambda Managed Instances supportano lo stesso flusso di lavoro di controllo delle versioni numerate di Lambda (impostazione predefinita). Se preferisci non mantenere versioni numerate, Lambda Managed Instances introduce un nuovo tipo di versione:. `$LATEST.PUBLISHED` Questa versione consente di creare o ripubblicare l'ultima versione pubblicata secondo necessità con codice o configurazione aggiornati, senza gestire versioni numerate.

Differenza fondamentale rispetto a $LATEST: quando richiami una funzione Lambda Managed Instances utilizzando un ARN non qualificato, Lambda richiama implicitamente la versione anziché la versione $LATEST non pubblicata. `$LATEST.PUBLISHED`

Il seguente comando AWS CLI crea o ripubblica la versione. `$LATEST.PUBLISHED`

```
aws lambda publish-version --function-name my-function --publish-to LATEST_PUBLISHED
```

Verrà visualizzato l'output seguente:

```
{
  "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**
>
> Se utilizzi AWS CloudFormation o la console Lambda per creare una funzione Lambda Managed Instances, Lambda crea automaticamente la versione. `$LATEST.PUBLISHED`

Runtime di esecuzione delle istanze gestite Lambda

Lambda elabora le richieste in modo diverso quando utilizza istanze gestite Lambda. Invece di gestire le richieste in sequenza in ogni ambiente di esecuzione, le istanze gestite Lambda elaborano più richieste contemporaneamente all'interno di ciascun ambiente di esecuzione. Questa modifica nel modello di esecuzione significa che le funzioni che utilizzano le istanze gestite Lambda devono considerare la sicurezza dei thread, la gestione dello stato e l'isolamento del contesto, problemi che non sorgono nel modello a concorrenza singola Lambda (predefinito). Inoltre, l'implementazione multi-concorrenza varia a seconda dei runtime.

Runtime supportati

I seguenti runtime supportano le istanze gestite Lambda:

- Java: Java 21 e versioni successive.
- Python: Python 3.13 e versioni successive.
- Node.js: Node.js 22 e versioni successive.
- .NET: .NET 8 e versioni successive.

Considerazioni specifiche sul runtime

Ogni runtime implementa la concorrenza multipla in modo diverso. È necessario comprendere come viene implementata la concorrenza multipla nel linguaggio di programmazione prescelto per applicare le migliori pratiche di concorrenza appropriate.

Java

Utilizza un unico processo con thread del sistema operativo per la concorrenza. Più thread eseguono il metodo handler contemporaneamente, richiedendo una gestione thread-safe delle risorse statali e condivise.

Python

Utilizza più processi Python in cui ogni richiesta simultanea viene eseguita in un processo separato. Ciò protegge dalla maggior parte dei problemi di concorrenza, sebbene sia necessaria attenzione per le risorse condivise come la directory. `/tmp`

Node.js

Utilizza thread di lavoro con esecuzione asincrona. Le richieste concorrenti sono distribuite tra i thread di lavoro e ogni thread di lavoro può anche gestire le richieste simultanee in modo asincrono, il che richiede una gestione sicura delle risorse statali e condivise.

.NET

Utilizza .NET Tasks con l'elaborazione asincrona di più richieste simultanee. Richiede una gestione sicura delle risorse statali e condivise.

Fasi successive

Per informazioni dettagliate su ogni runtime, consultate i seguenti argomenti:

- Runtime Java per istanze gestite Lambda
- Runtime Node.js per istanze gestite Lambda
- Runtime Python per istanze gestite Lambda
- .NET runtime per istanze gestite Lambda

Runtime Java per istanze gestite Lambda

Per i runtime Java, le istanze gestite Lambda utilizzano i thread del sistema operativo per la concorrenza. Lambda carica l'oggetto gestore una volta per ambiente di esecuzione durante l'inizializzazione e quindi crea più thread. Questi thread vengono eseguiti in parallelo e richiedono una gestione thread-safe delle risorse statali e condivise. Ogni thread condivide lo stesso oggetto gestore e tutti i campi statici.

Configurazione della concorrenza

Il numero massimo di richieste simultanee che Lambda invia a ciascun ambiente di esecuzione è controllato dall'impostazione nella configurazione `PerExecutionEnvironmentMaxConcurrency` della funzione. Questa è un'impostazione opzionale e il valore predefinito varia a seconda del runtime. Per i runtime Java, l'impostazione predefinita è 32 richieste simultanee per vCPU, oppure è possibile configurare un valore personalizzato. Questo valore determina anche il numero di thread utilizzati dal runtime Java. Lambda regola automaticamente il numero di richieste simultanee fino al massimo configurato in base alla capacità di ciascun ambiente di esecuzione di assorbire tali richieste.

Creazione di funzioni per la multiconcorrenza

È necessario applicare le stesse pratiche di sicurezza dei thread quando si utilizzano le istanze gestite Lambda come si farebbe in qualsiasi altro ambiente multithread. Poiché l'oggetto handler è condiviso tra tutti i thread di runtime worker, qualsiasi stato mutabile deve essere thread-safe. Ciò include raccolte, connessioni al database e qualsiasi oggetto statico modificato durante l'elaborazione della richiesta.

AWS I client SDK sono thread-safe e non richiedono una gestione speciale.

Esempio: pool di connessioni al database

Il codice seguente utilizza un oggetto di connessione al database statico condiviso tra i thread. A seconda della libreria di connessioni utilizzata, questa potrebbe non essere thread-safe.

```
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();
    }
```

```
}
```

Un approccio thread-safe consiste nell'utilizzare un pool di connessioni. Nell'esempio seguente, il gestore di funzioni recupera una connessione dal pool. La connessione viene utilizzata solo nel contesto di una singola richiesta.

```
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();
    }
}
```

Esempio: collezioni

Le raccolte Java standard non sono thread-safe:

```
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";
    }
}
```

Utilizza invece raccolte 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";
    }
}
```

Directory /tmp condivisa

La `/tmp` directory è condivisa tra tutte le richieste simultanee nell'ambiente di esecuzione. Le scritture simultanee sullo stesso file possono causare il danneggiamento dei dati, ad esempio se un altro processo sovrascrive il file. Per risolvere questo problema, implementate il blocco dei file per i file condivisi o utilizzate nomi di file univoci per thread o per richiesta per evitare conflitti. Ricordati di ripulire i file non necessari per evitare di esaurire lo spazio disponibile.

Registrazione dei log

L'interlacciamento dei log (le voci di registro di diverse richieste vengono interlacciate nei log) è normale nei sistemi multi-concorrenti.

Le funzioni che utilizzano Lambda Managed Instances utilizzano sempre il formato di registro JSON strutturato introdotto con controlli di registrazione avanzati. Questo formato include `requestId`, che consente di correlare le voci di registro a una singola richiesta. Quando si utilizza l'`LambdaLogger`oggetto, `requestId` viene incluso automaticamente in ogni voce di registro. `context.getLogger()` Per ulteriori informazioni, vederethe section called “Utilizzo dei controlli di registrazione avanzati di Lambda con Java”.

Contesto della richiesta

L'`context`oggetto è associato al thread di richiesta. Using `context.getAwsRequestId()` fornisce un accesso thread-safe all'ID della richiesta corrente.

Utilizzare `context.getXrayTraceId()` per accedere all'ID di traccia X-Ray. Ciò fornisce un accesso thread-safe all'ID di traccia per la richiesta corrente. Lambda non supporta la variabile di `_X_AMZN_TRACE_ID` ambiente con Lambda Managed Instances. L'ID di traccia X-Ray viene propagato automaticamente quando si utilizza l'SDK. AWS

Utilizzato per rilevare i `com.amazonaws.services.lambda.runtime.Context.getRemainingTimeInMillis()` timeout. Per ulteriori informazioni, consulta the section called “Gestione e ripristino degli errori”.

Se utilizzi thread virtuali nel tuo programma o crei thread durante l'inizializzazione, dovrai passare qualsiasi contesto di richiesta richiesto a questi thread.

Inizializzazione e spegnimento

L'inizializzazione della funzione avviene una volta per ambiente di esecuzione. Gli oggetti creati durante l'inizializzazione vengono condivisi tra i thread.

Per le funzioni Lambda con estensioni, l'ambiente di esecuzione emette un segnale SIGTERM durante lo spegnimento. Questo segnale viene utilizzato dalle estensioni per attivare attività di pulizia, come lo svuotamento dei buffer. È possibile sottoscrivere eventi SIGTERM per attivare attività di pulizia delle funzioni, come la chiusura delle connessioni al database. Per ulteriori informazioni sul ciclo di vita dell'ambiente di esecuzione, consulta the section called “Ambiente di esecuzione”.

Versioni di dipendenza

Lambda Managed Instances richiede le seguenti versioni minime del pacchetto:

- AWS SDK for Java 2.0: versione 2.34.0 o successiva

- AWS X-Ray SDK per Java: versione 2.20.0 o successiva
- AWS Distro for OpenTelemetry - Instrumentation for Java: versione 2.20.0 o successiva
- Powertools for AWS Lambda (Java): versione 2.8.0 o successiva

Utensili elettrici per AWS Lambda (Java)

Powertools for AWS Lambda (Java) è compatibile con Lambda Managed Instances e fornisce utilità per la registrazione, il tracciamento, le metriche e altro ancora. Per ulteriori informazioni, vedere Powertools for AWS Lambda (Java).

Fasi successive

- Esamina il runtime di Node.js per le istanze gestite Lambda
- Esamina il runtime Python per le istanze gestite Lambda
- Esamina il runtime di.NET per le istanze gestite da Lambda
- Scopri come scalare le istanze gestite Lambda

Runtime Node.js per istanze gestite Lambda

Per i runtime di Node.js, Lambda Managed Instances utilizza thread di lavoro `async` con esecuzione basata su`await`/per gestire le richieste simultanee. L'inizializzazione della funzione avviene una volta per thread di lavoro. Le chiamate simultanee vengono gestite in due dimensioni: i thread di lavoro forniscono il parallelismo su v e l'esecuzione asincrona fornisce la concorrenza all'interno di CPUs ciascun thread. Ogni richiesta concorrente gestita dallo stesso thread di lavoro condivide lo stesso oggetto del gestore e lo stesso stato globale, il che richiede una gestione sicura in caso di più richieste simultanee.

Simultaneità massima

Il numero massimo di richieste simultanee che Lambda invia a ciascun ambiente di esecuzione è controllato dall'impostazione nella configurazione `PerExecutionEnvironmentMaxConcurrency` della funzione. Questa è un'impostazione opzionale e il valore predefinito varia a seconda del runtime. Per i runtime di Node.js, l'impostazione predefinita è 64 richieste simultanee per vCPU, oppure è possibile configurare un valore personalizzato. Lambda regola automaticamente il numero di richieste simultanee fino al massimo configurato in base alla capacità di ciascun ambiente di esecuzione di assorbire tali richieste.

Per Node.js, il numero di richieste simultanee che ogni ambiente di esecuzione può elaborare è determinato dal numero di thread di lavoro e dalla capacità di ciascun thread di lavoro di elaborare le richieste concorrenti in modo asincrono. Il numero predefinito di thread di lavoro è determinato dal numero di v CPUs disponibili oppure è possibile configurare il numero di thread di lavoro impostando la variabile di ambiente. `AWS_LAMBDA_NODEJS_WORKER_COUNT` Si consiglia di utilizzare gestori di funzioni asincroni poiché ciò consente l'elaborazione di più richieste per thread di lavoro. Se il gestore di funzioni è sincrono, ogni thread di lavoro può elaborare solo una singola richiesta alla volta.

Creazione di funzioni per la concorrenza multipla

Con un gestore di funzioni asincrone, ogni runtime worker elabora più richieste contemporaneamente. Gli oggetti globali verranno condivisi tra più richieste simultanee. Per gli oggetti mutabili, evita di usare global state or use. `AsyncLocalStorage`

AWS I client SDK sono asincroni e non richiedono una gestione speciale.

Esempio: stato globale

Il codice seguente utilizza un oggetto globale che viene modificato all'interno del gestore della funzione. Questo non è asincrono.

```
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 };
};
```

L'inizializzazione dell'`state`oggetto all'interno del gestore della funzione evita lo stato globale condiviso.

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

```
        currentUser: event.userId,
        requestData: event.data
    };

    await processData(state.requestData);

    return { user: state.currentUser };
};
```

Esempio: connessioni al database

Il codice seguente utilizza un oggetto client condiviso che viene condiviso tra più chiamate. A seconda della libreria di connessioni utilizzata, questo potrebbe non essere sicuro in termini di concorrenza.

```
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])
  };
};
```

Un approccio sicuro per la concorrenza consiste nell'utilizzare un pool di connessioni. Il pool utilizza una connessione separata per ogni interrogazione simultanea del database.

```
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])
  };
};
```

Gestori basati su callback Node.js 22

Quando si utilizza Node.js 22, non è possibile utilizzare un gestore di funzioni basato su callback con Lambda Managed Instances. I gestori basati su callback sono supportati solo per le funzioni Lambda (predefinite). Per i runtime di Node.js 24 e versioni successive, i gestori di funzioni basati su callback sono obsoleti sia per le istanze gestite Lambda (impostazione predefinita) che per quelle gestite Lambda.

Utilizza invece un gestore di `async` funzioni quando usi Lambda Managed Instances. Per ulteriori informazioni, consulta Definire il gestore di funzioni Lambda in Node.js.

Directory /tmp condivisa

La `/tmp` directory è condivisa tra tutte le richieste simultanee nell'ambiente di esecuzione. Le scritture simultanee sullo stesso file possono causare il danneggiamento dei dati, ad esempio se un altro processo sovrascrive il file. Per risolvere questo problema, implementate il blocco dei file per i file condivisi o utilizzate nomi di file univoci per richiesta per evitare conflitti. Ricordati di ripulire i file non necessari per evitare di esaurire lo spazio disponibile.

Registrazione dei log

L'interlacciamento dei log (le voci di registro di diverse richieste vengono interlacciate nei log) è normale nei sistemi multi-concorrenti. Le funzioni che utilizzano Lambda Managed Instances utilizzano sempre il formato di registro JSON strutturato introdotto con controlli di registrazione avanzati. Questo formato include`requestId`, che consente di correlare le voci di registro a una singola richiesta. Quando si utilizza il `console` logger, `requestId` viene automaticamente incluso in ogni voce di registro. Per ulteriori informazioni, vederethe section called “Utilizzo dei controlli di registrazione avanzati di Lambda con Node.js”.

Le librerie di registrazione di terze parti più diffuse, come Winston, includono in genere il supporto per l'utilizzo della console per l'output dei log.

Contesto della richiesta

Using `context.awsRequestId` fornisce un accesso sicuro e asincrono all'ID della richiesta corrente.

Utilizzare `context.xRayTraceId` per accedere all'ID di traccia X-Ray. Ciò fornisce un accesso sicuro in termini di concorrenza all'ID di traccia per la richiesta corrente. Lambda non supporta la variabile di `_X_AMZN_TRACE_ID` ambiente con Lambda Managed Instances. L'ID di traccia X-Ray viene propagato automaticamente quando si utilizza l'SDK. AWS

Utilizzato per rilevare i `context.getRemainingTimeInMillis()` timeout. Per ulteriori informazioni, consulta the section called “Gestione e ripristino degli errori”.

Inizializzazione e spegnimento

L'inizializzazione della funzione avviene una volta per thread di lavoro. È possibile che vengano visualizzate voci di registro ripetute se la funzione emette log durante l'inizializzazione.

Per le funzioni Lambda con estensioni, l'ambiente di esecuzione emette un segnale SIGTERM durante lo spegnimento. Questo segnale viene utilizzato dalle estensioni per attivare attività di pulizia, come lo svuotamento dei buffer. Le funzioni Lambda (impostazione predefinita) con estensioni possono anche sottoscrivere il segnale SIGTERM utilizzando. `process.on()` Questa funzionalità non è supportata per le funzioni che utilizzano istanze gestite Lambda poiché `process.on()` non può essere utilizzata con i thread di lavoro. Per ulteriori informazioni sul ciclo di vita dell'ambiente di esecuzione, consulta the section called “Ambiente di esecuzione”.

Versioni di dipendenza

Lambda Managed Instances richiede le seguenti versioni minime del pacchetto:

- AWS SDK per JavaScript v3: versione 3.933.0 o successiva
- AWS X-Ray SDK per Node.js: versione 3.12.0 o successiva
- AWS Distro for OpenTelemetry - Strumentazione per: versione 0.8.0 o successiva JavaScript
- Powertools for AWS Lambda TypeScript (): versione 2.29.0 o successiva

Elettroutensili per AWS TypeScript Lambda ()

Powertools for AWS Lambda TypeScript () è compatibile con Lambda Managed Instances e fornisce utilità per la registrazione, il tracciamento, le metriche e altro ancora. Per ulteriori informazioni, vedete Powertools for AWS Lambda TypeScript ().

Fasi successive

- Esamina il runtime Java per le istanze gestite Lambda
- Esamina il runtime Python per le istanze gestite Lambda
- Esamina il runtime di.NET per le istanze gestite da Lambda
- Scopri come scalare le istanze gestite Lambda

Runtime Python per istanze gestite Lambda

Il runtime Lambda utilizza più processi Python per gestire le richieste simultanee. Ogni richiesta simultanea viene eseguita in un processo separato con spazio di memoria e inizializzazione propri. Ogni processo gestisce una richiesta alla volta, in modo sincrono. I processi non condividono direttamente la memoria, quindi le variabili globali, le cache a livello di modulo e gli oggetti singleton sono isolati tra le richieste simultanee.

Configurazione della concorrenza

Il numero massimo di richieste simultanee che Lambda invia a ciascun ambiente di esecuzione è controllato dall'impostazione nella configurazione `PerExecutionEnvironmentMaxConcurrency` della funzione. Questa è un'impostazione opzionale e il valore predefinito varia a seconda del runtime. Per i runtime Python, l'impostazione predefinita è 16 richieste simultanee per vCPU, oppure puoi configurare il tuo valore. Questo valore determina anche il numero di processi utilizzati dal

runtime di Python. Lambda regola automaticamente il numero di richieste simultanee fino al massimo configurato in base alla capacità di ciascun ambiente di esecuzione di assorbire tali richieste.

> **Important**
>
> L'utilizzo della concorrenza basata sui processi significa che ogni processo di runtime worker esegue la propria inizializzazione. L'utilizzo totale della memoria è uguale alla memoria per processo moltiplicata per il numero di processi simultanei. Se state caricando librerie o set di dati di grandi dimensioni e la concorrenza è elevata, avrete un notevole ingombro di memoria. In base al carico di lavoro, potrebbe essere necessario regolare il CPU-to-memory rapporto o utilizzare un'impostazione di concorrenza inferiore per evitare di superare la memoria disponibile. Puoi utilizzare la `MemoryUtilization` metrica in per tenere traccia del consumo CloudWatch di memoria.

Creazione di funzioni per la multiconcorrenza

Grazie al modello multi-concorrenza basato sui processi, le funzioni Lambda Managed Instances che utilizzano i runtime Python non accedono contemporaneamente alle risorse in memoria da più richiami. Non è necessario applicare pratiche di codifica per la sicurezza della concorrenza in memoria.

Directory /tmp condivisa

La `/tmp` directory è condivisa tra tutte le richieste simultanee nell'ambiente di esecuzione. Le scritture simultanee sullo stesso file possono causare il danneggiamento dei dati, ad esempio se un altro processo sovrascrive il file. Per risolvere questo problema, implementate il blocco dei file per i file condivisi o utilizzate nomi di file univoci per processo o per richiesta per evitare conflitti. Ricorda di ripulire i file non necessari per evitare di esaurire lo spazio disponibile.

Registrazione dei log

L'interlacciamento dei log (le voci di registro di diverse richieste vengono interlacciate nei log) è normale nei sistemi multi-concorrenti.

Le funzioni che utilizzano Lambda Managed Instances utilizzano sempre il formato di registro JSON strutturato introdotto con controlli di registrazione avanzati. Questo formato include`requestId`, che consente di correlare le voci di registro a una singola richiesta. Quando usi il `logging` modulo della libreria standard Python in Lambda, `requestId` viene incluso automaticamente in ogni voce

di registro. Per ulteriori informazioni, consulta Usare i controlli di registrazione avanzati Lambda con Python.

Contesto della richiesta

Utilizzare `context.aws_request_id` per accedere all'ID della richiesta corrente.

Con i runtime Python, puoi usare la variabile di `_X_AMZN_TRACE_ID` ambiente per accedere all'ID di traccia X-Ray con Lambda Managed Instances. L'ID di traccia X-Ray viene propagato automaticamente quando si utilizza l'SDK. AWS

Utilizzato per rilevare i `context.get_remaining_time_in_millis()` timeout. Per ulteriori informazioni, consulta the section called “Gestione e ripristino degli errori”.

Inizializzazione e spegnimento

L'inizializzazione della funzione avviene una volta per processo. È possibile che vengano visualizzate voci di registro ripetute se la funzione emette registri durante l'inizializzazione.

Per le funzioni Lambda con estensioni, l'ambiente di esecuzione emette un segnale SIGTERM durante lo spegnimento. Questo segnale viene utilizzato dalle estensioni per attivare attività di pulizia, come lo svuotamento dei buffer. È possibile sottoscrivere eventi SIGTERM per attivare attività di pulizia delle funzioni, come la chiusura delle connessioni al database. Per ulteriori informazioni sul ciclo di vita dell'ambiente di esecuzione, consulta the section called “Ambiente di esecuzione”.

Versioni di dipendenza

Lambda Managed Instances richiede le seguenti versioni minime del pacchetto:

- Powertools for AWS Lambda (Python): versione 3.23.0 o successiva

Powertools per AWS Lambda (Python)

Powertools for AWS Lambda (Python) è compatibile con Lambda Managed Instances e fornisce utilità per la registrazione, il tracciamento, le metriche e altro ancora. Per ulteriori informazioni, vedere Powertools for AWS Lambda (Python).

Fasi successive

- Esamina il runtime Java per le istanze gestite Lambda

- Esamina il runtime di Node.js per le istanze gestite Lambda
- Esamina il runtime di.NET per le istanze gestite Lambda
- Scopri come scalare le istanze gestite Lambda

.NET runtime per istanze gestite Lambda

Per i runtime.NET, le istanze gestite Lambda utilizzano un singolo processo.NET per ambiente di esecuzione. Più richieste simultanee vengono elaborate utilizzando .NET Tasks.

Configurazione della concorrenza

Il numero massimo di richieste simultanee che Lambda invia a ciascun ambiente di esecuzione è controllato dall'impostazione nella configurazione `PerExecutionEnvironmentMaxConcurrency` della funzione. Questa è un'impostazione opzionale e il valore predefinito varia a seconda del runtime. Per i runtime.NET, l'impostazione predefinita è 32 richieste simultanee per vCPU, oppure è possibile configurare un valore personalizzato. Lambda regola automaticamente il numero di richieste simultanee fino al massimo configurato in base alla capacità di ciascun ambiente di esecuzione di assorbire tali richieste.

Creazione di funzioni per la multiconcorrenza

È necessario applicare le stesse pratiche di sicurezza della concorrenza quando si utilizzano le istanze gestite Lambda come si farebbe in qualsiasi altro ambiente multi-concorrente. Poiché l'oggetto handler è condiviso tra tutte le attività, qualsiasi stato mutabile deve essere thread-safe. Ciò include raccolte, connessioni al database e qualsiasi oggetto statico modificato durante l'elaborazione della richiesta.

AWS I client SDK sono thread-safe e non richiedono una gestione speciale.

Esempio: pool di connessioni al database

Il codice seguente utilizza un oggetto di connessione al database statico che viene condiviso tra richieste simultanee. L'`SqlConnection`oggetto non è thread-safe.

```
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();

        ...
    }
}
```

Per risolvere questo problema, utilizzate una connessione separata per ogni richiesta, ricavata da un pool di connessioni. I provider ADO.NET supportano `Microsoft.Data.SqlClient` automaticamente il pool di connessioni quando l'oggetto di connessione viene aperto.

```
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();

        ...
    }
}
```

Esempio: raccolte

Le raccolte .NET standard non sono thread-safe:

```
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";
    }
}
```

Usa le raccolte dallo spazio dei `System.Collections.Concurrent` nomi per la sicurezza della concorrenza:

```
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";
    }
}
```

Directory /tmp condivisa

La `/tmp` directory è condivisa tra tutte le richieste simultanee nell'ambiente di esecuzione. Le scritture simultanee sullo stesso file possono causare il danneggiamento dei dati, ad esempio se un'altra richiesta sovrascrive il file. Per risolvere questo problema, implementate il blocco dei file per i file condivisi o utilizzate nomi di file univoci per richiesta per evitare conflitti. Ricordati di ripulire i file non necessari per evitare di esaurire lo spazio disponibile.

Registrazione dei log

L'interlacciamento dei log (le voci di registro di diverse richieste vengono interlacciate nei log) è normale nei sistemi multi-concorrenti. Le funzioni che utilizzano Lambda Managed Instances utilizzano sempre il formato di registro JSON strutturato introdotto con controlli di registrazione avanzati. Questo formato include`requestId`, che consente di correlare le voci di registro a una singola richiesta. Quando si utilizza l'`context.Logger`oggetto per generare registri, `requestId` viene automaticamente incluso in ogni voce di registro. Per ulteriori informazioni, vederethe section called “Utilizzo dei controlli di registrazione avanzati di Lambda con .NET”.

Contesto della richiesta

Utilizza la `context.AwsRequestId` proprietà per accedere all'ID della richiesta corrente.

Utilizzate la `context.TraceId` proprietà per accedere all'ID della traccia X-Ray. Ciò fornisce un accesso sicuro per la concorrenza all'ID di traccia per la richiesta corrente. Lambda non supporta la variabile di `_X_AMZN_TRACE_ID` ambiente con Lambda Managed Instances. L'ID di traccia X-Ray viene propagato automaticamente quando si utilizza l'SDK. AWS

Utilizzato per rilevare i `ILambdaContext.RemainingTime` timeout. Per ulteriori informazioni, consulta the section called “Gestione e ripristino degli errori”.

Inizializzazione e spegnimento

L'inizializzazione della funzione avviene una volta per ambiente di esecuzione. Gli oggetti creati durante l'inizializzazione vengono condivisi tra le richieste.

Per le funzioni Lambda con estensioni, l'ambiente di esecuzione emette un segnale SIGTERM durante lo spegnimento. Questo segnale viene utilizzato dalle estensioni per attivare attività di pulizia, come lo svuotamento dei buffer. È possibile sottoscrivere eventi SIGTERM per attivare attività di pulizia delle funzioni, come la chiusura delle connessioni al database. Per ulteriori informazioni sul ciclo di vita dell'ambiente di esecuzione, consulta the section called “Ambiente di esecuzione”.

Versioni di dipendenza

Lambda Managed Instances richiede le seguenti versioni minime del pacchetto:

- Amazon.Lambda.Core: versione 2.7.1 o successiva
- Amazon.Lambda. RuntimeSupport: versione 1.14.1 o successiva
- OpenTelemetry.Strumentazione. AWSLambda: versione 1.14.0 o successiva

- AWSXRayRecorder.Core: versione 2.16.0 o successiva
- AWSSDK.Core: versione 4.0.0.32 o successiva

Utensili elettrici per AWS Lambda (.NET)

Powertools for AWS Lambda (.NET) AWS e Distro OpenTelemetry for - Instrumentation for DotNet attualmente non supportano le istanze gestite Lambda.

Fasi successive

- Esamina il runtime Java per le istanze gestite Lambda
- Esamina il runtime di Node.js per le istanze gestite Lambda
- Esamina il runtime Python per le istanze gestite Lambda
- Scopri come scalare le istanze gestite Lambda

Rete per istanze gestite Lambda

Quando si eseguono le funzioni Lambda Managed Instances, è necessario configurare la connettività di rete per consentire alle funzioni di accedere alle risorse esterne al VPC. Ciò include AWS servizi come Amazon S3 e DynamoDB. La connettività è necessaria anche per trasmettere dati di telemetria a CloudWatch Logs e X-Ray.

Opzioni di connettività

Esistono tre approcci principali per configurare la connettività VPC, ciascuno con diversi compromessi in termini di costi, sicurezza e complessità.

Sottorete pubblica con un gateway Internet

Questa opzione utilizza una sottorete pubblica con accesso diretto a Internet tramite un gateway Internet. È possibile scegliere tra IPv4 e IPv6 configurazioni.

IPv4 con gateway internet

Per configurare la IPv4 connettività con un gateway Internet

1. Crea o usa una sottorete pubblica esistente con un blocco IPv4 CIDR.

2. Collegare un Internet Gateway al VPC.
3. Aggiorna la tabella delle rotte per indirizzare il `0.0.0.0/0` traffico verso il gateway Internet.
4. Assicurati che alle risorse siano assegnati IPv4 indirizzi pubblici o indirizzi IP elastici.
5. Configura i gruppi di sicurezza per consentire il traffico in uscita sulle porte richieste.

Questa configurazione fornisce una connettività bidirezionale, che consente sia le connessioni in uscita dalle tue funzioni sia le connessioni in entrata da Internet.

IPv6 con gateway internet

Per configurare la IPv6 connettività con un gateway Internet

1. Abilita IPv6 sul tuo VPC.
2. Crea o usa una sottorete pubblica esistente con un blocco IPv6 CIDR assegnato.
3. Collega un gateway Internet al tuo VPC (lo stesso gateway Internet può gestire entrambi IPv4 e IPv6).
4. Aggiorna la tabella delle rotte per indirizzare il `::/0` traffico verso il gateway Internet.
5. Verifica che i AWS servizi di cui hai bisogno per accedere all'assistenza IPv6 nella tua regione.
6. Configura i gruppi di sicurezza per consentire il traffico in uscita sulle porte richieste.

Questa configurazione fornisce una connettività bidirezionale tramite l'indirizzamento. IPv6

IPv6 con gateway Internet solo in uscita

Per configurare la IPv6 connettività con un gateway Internet solo in uscita

1. Abilita IPv6 sul tuo VPC.
2. Crea o usa una sottorete pubblica esistente con un blocco IPv6 CIDR assegnato.
3. Collega un gateway Internet solo in uscita al tuo VPC.
4. Aggiorna la tabella delle rotte per indirizzare il `::/0` traffico verso il gateway Internet solo in uscita.
5. Verifica che i AWS servizi di cui hai bisogno per accedere all'assistenza IPv6 nella tua regione.
6. Configura i gruppi di sicurezza per consentire il traffico in uscita sulle porte richieste.

Questa configurazione fornisce connettività solo in uscita, impedendo le connessioni in entrata da Internet e consentendo alle funzioni di avviare connessioni in uscita.

Endpoint VPC

Gli endpoint VPC ti consentono di connettere privatamente il tuo VPC ai AWS servizi supportati senza richiedere un gateway Internet, un dispositivo NAT, una connessione VPN o una connessione Direct Connect. AWS Il traffico tra il tuo VPC e il AWS servizio non esce dalla rete Amazon.

Per configurare gli endpoint VPC

1. Apri la console Amazon VPC all'indirizzo console.aws.amazon.com/vpc/.
2. Nel pannello di navigazione, seleziona Endpoints (Endpoint).
3. Seleziona Crea endpoint.
4. Per Service category (Categoria servizio), scegli AWS services.
5. Per Nome del servizio, seleziona l'endpoint del servizio di cui hai bisogno (ad esempio, `com.amazonaws.region.s3` per Amazon S3).
6. In VPC, seleziona il tuo VPC.
7. Per Sottoreti, seleziona le sottoreti in cui desideri creare interfacce di rete per endpoint. Per un'elevata disponibilità, seleziona le sottoreti in più zone di disponibilità.
8. Per Security groups (Gruppi di sicurezza), seleziona i gruppi di sicurezza da associare alle interfacce di rete dell'endpoint. I gruppi di sicurezza devono consentire il traffico in entrata dal gruppo di sicurezza della funzione sulle porte richieste.
9. Seleziona Crea endpoint.

Ripeti questi passaggi per ogni AWS servizio a cui le tue funzioni devono accedere.

Sottorete privata con gateway NAT

Questa opzione utilizza un gateway NAT per fornire l'accesso a Internet alle risorse nelle sottoreti private, mantenendo le risorse private.

Per configurare una sottorete privata con gateway NAT

1. Crea una sottorete pubblica (se non ne esiste già una) con un blocco CIDR.
2. Collegare un Internet Gateway al VPC.
3. Crea un gateway NAT nella sottorete pubblica e assegna un indirizzo IP elastico.
4. Aggiorna la tabella delle rotte della sottorete pubblica per aggiungere una route: `0.0.0.0/0` → internet gateway.

5. Crea o usa una sottorete privata esistente con un blocco CIDR.
6. Aggiorna la tabella di routing della sottorete privata per aggiungere una rotta: `0.0.0.0/0` → gateway NAT.
7. Configura i gruppi di sicurezza per consentire il traffico in uscita sulle porte richieste.

Per un'elevata disponibilità, implementa un gateway NAT in ogni zona di disponibilità e configura le tabelle di routing per zona di disponibilità per utilizzare il gateway NAT locale. In questo modo si evitano costi di trasferimento dati Cross-AZ e si migliora la resilienza.

Scelta di un'opzione di connettività

Quando scegli un'opzione di connettività, considera i seguenti fattori:

Sottorete pubblica con gateway Internet

- Configurazione più semplice con il costo più basso
- Adatto per ambienti di sviluppo e test
- Le risorse possono ricevere connessioni in entrata da Internet (considerazioni relative alla sicurezza)
- Supporta sia IPv4 e IPv6

Endpoint VPC

- Massima sicurezza, il traffico rimane all'interno della AWS rete
- Latenza inferiore rispetto al routing Internet
- Consigliato per ambienti di produzione con requisiti di sicurezza rigorosi
- Costo più elevato per endpoint, per zona di disponibilità e per GB di elaborazione
- Richiede un endpoint in ogni zona di disponibilità per un'elevata disponibilità

Sottorete privata con gateway NAT

- Le risorse rimangono private senza accesso a Internet in entrata
- Modello di architettura aziendale standard
- Supporta tutto il traffico IPv4 Internet

- Costo moderato con tariffe orarie per il gateway NAT e per l'elaborazione dei dati
- Supporta solo IPv4

Fasi successive

- Scopri i provider di capacità per le istanze gestite Lambda
- Comprendi la scalabilità per le istanze gestite Lambda
- Consulta le guide specifiche di runtime per Java, Node.js e Python
- Comprendi la sicurezza e le autorizzazioni per le istanze gestite Lambda

Monitoraggio delle istanze gestite da Lambda

Puoi monitorare le istanze gestite Lambda utilizzando le metriche. CloudWatch Lambda pubblica automaticamente le metriche per aiutarti CloudWatch a monitorare l'utilizzo delle risorse, tenere traccia dei costi e ottimizzare le prestazioni.

Metriche disponibili

Lambda Managed Instances fornisce metriche a due livelli: a livello di provider di capacità e a livello di ambiente di esecuzione.

Metriche a livello di provider di capacità

Le metriche a livello di Capacity Provider forniscono visibilità sull'utilizzo complessivo delle risorse in tutte le istanze. Queste metriche utilizzano le seguenti dimensioni:

- CapacityProviderName- Il nome del fornitore di capacità
- InstanceType- Il tipo di EC2 istanza

Metriche di utilizzo delle risorse:

- CPUUtilization- La percentuale di utilizzo della CPU tra le istanze del provider di capacità
- MemoryUtilization- La percentuale di utilizzo della memoria tra le istanze del provider di capacità
- NetworkOut- Traffico di rete inviato tramite il cliente ENI (in byte)
- NetworkIn- Traffico di rete ricevuto tramite il cliente ENI (in byte)

- DiskReadBytes- Leggi il traffico dallo storage locale tra le istanze (in byte)
- DiskWriteBytes- Scrivi il traffico verso l'archiviazione locale tra le istanze (in byte)

Metriche sulla capacità:

- v CPUAvailable - La quantità di vCPU disponibile nelle istanze da allocare (in numero)
- MemoryAvailable- La quantità di memoria disponibile nelle istanze da allocare (in byte)
- v CPUAllocated - La quantità di vCPU allocata sulle istanze per gli ambienti di esecuzione (in numero)
- MemoryAllocated- La quantità di memoria allocata sulle istanze per gli ambienti di esecuzione (in byte)

Metriche a livello di ambiente di esecuzione

Le metriche a livello di ambiente di esecuzione forniscono visibilità sull'utilizzo delle risorse e sulla concorrenza per le singole funzioni. Queste metriche utilizzano le seguenti dimensioni:

- CapacityProviderName- Il nome del fornitore di capacità
- FunctionName- Il nome della tua funzione Lambda

Metriche dell'ambiente di esecuzione disponibili:

- ExecutionEnvironmentConcurrency- La massima concorrenza in un periodo di campionamento di 5 minuti
- ExecutionEnvironmentConcurrencyLimit- Il limite massimo di concorrenza per ambiente di esecuzione
- ExecutionEnvironmentCPUUtilization- La percentuale di utilizzo della CPU per gli ambienti di esecuzione della funzione
- ExecutionEnvironmentMemoryUtilization- La percentuale di utilizzo della memoria per gli ambienti di esecuzione della funzione

Frequenza e ritenzione metriche

Le metriche di Lambda Managed Instances vengono pubblicate a intervalli di 5 minuti e conservate per 15 mesi.

Visualizzazione delle metriche in CloudWatch

Per visualizzare i parametri delle istanze gestite da Lambda nella console CloudWatch

1. Apri la CloudWatch console all'indirizzo console.aws.amazon.com/cloudwatch/.
2. Nel riquadro di navigazione, seleziona Parametri.
3. Nella scheda Tutte le metriche, scegli AWS/Lambda.
4. Scegli la dimensione metrica che desideri visualizzare:
   - Per le metriche a livello di provider di capacità, filtra per e CapacityProviderNameInstanceType
   - Per le metriche a livello di ambiente di esecuzione, filtra per e CapacityProviderNameFunctionName
5. Seleziona le metriche che desideri monitorare.

Utilizzo delle metriche per ottimizzare le prestazioni

Monitora l'utilizzo della CPU e della memoria per capire se le tue funzioni sono dimensionate correttamente. Un utilizzo elevato può indicare la necessità di tipi di istanze più grandi o di una maggiore allocazione della memoria funzionale. Tieni traccia delle metriche di concorrenza per comprendere il comportamento di scalabilità e identificare potenziali limitazioni.

Monitora i parametri di capacità per garantire che siano disponibili risorse sufficienti per i tuoi carichi di lavoro. I parametri v CPUAvailable e le MemoryAvailablemetriche ti aiutano a comprendere la capacità residua delle tue istanze.

Fasi successive

- Scopri come scalare le istanze gestite Lambda
- Consulta le guide specifiche di runtime per Java, Node.js e Python
- Configura la connettività VPC per i tuoi provider di capacità
- Comprendi la sicurezza e le autorizzazioni per le istanze gestite Lambda

Quote delle istanze gestite Lambda

Questa pagina descrive le quote di servizio per le istanze gestite AWS Lambda. Queste quote sono separate dalle quote AWS Lambda (predefinite). Alcune quote possono essere aumentate su richiesta.

Quote di richieste API Lambda

Queste quote controllano la velocità con cui è possibile effettuare chiamate API per gestire i provider di capacità di Lambda Managed Instances. I limiti di velocità delle API di lettura e scrittura si applicano a tutte le operazioni dei provider di capacità combinate, tra cui la creazione, l'aggiornamento, la descrizione e l'eliminazione dei provider di capacità.

Risorsa	Quota
La velocità combinata massima (richieste al secondo) per tutti i provider di capacità di lettura APIs	15 richieste al secondo. Non può essere aumentato.
La velocità combinata massima (richieste al secondo) per tutti i provider di capacità di scrittura APIs	1 richiesta al secondo. Non può essere aumentato.

Quote di risorse per Lambda Managed Instances

Queste quote definiscono i limiti per le risorse principali di Lambda Managed Instances all'interno del tuo account. AWS Esse regolano il numero di provider di capacità che è possibile creare e il numero di versioni di funzioni che possono essere associate a ciascun provider di capacità.

Risorsa	Quota
fornitori di capacità	1.000. Il numero massimo di fornitori di capacità creati in un account.
Versioni delle funzioni per provider di capacità	100. Il numero massimo di versioni di funzioni per provider di capacità. Non può essere aumentato.

Quote di mappatura delle sorgenti degli eventi

Queste quote controllano il throughput e i limiti di configurazione per l'elaborazione degli eventi da vari AWS servizi sulle istanze gestite Lambda. I limiti di throughput garantiscono prestazioni

prevedibili, mentre i limiti del numero di mappature aiutano a mantenere la stabilità del servizio. Le mappature delle sorgenti degli eventi su Lambda Managed Instances supportano Amazon SQS, DynamoDB Streams, Amazon Kinesis Data Streams, Amazon MSK e Apache Kafka autogestito come sorgenti di eventi.

Risorsa	Quota
Throughput di mappatura delle sorgenti di eventi SQS standard su istanze gestite Lambda	5 MB al secondo. Non può essere aumentato.
Velocità effettiva di mappatura delle sorgenti di eventi Kafka standard su istanze gestite Lambda	1 MB al secondo. Non può essere aumentato.
Mappature standard delle sorgenti degli eventi Kafka su istanze gestite Lambda	100 mappature delle sorgenti degli eventi. Non può essere aumentato.
Throughput di mappatura delle sorgenti di eventi Kinesis su istanze gestite Lambda	25 MB al secondo. Può essere aumentato.
Throughput di mappatura delle sorgenti di eventi DynamoDB su istanze gestite Lambda	10 MB al secondo. Può essere aumentato.
Richiama la velocità effettiva delle richieste per le chiamate asincrone sulle istanze gestite Lambda	5 MB al secondo. Può essere aumentato.

Richiesta di aumento delle quote

Per le quote che possono essere aumentate, puoi richiedere un aumento tramite la console Service Quotas.

Richiesta di un aumento delle quote

1. Apri la console Service Quotas all'indirizzo console.aws.amazon.com/servicequotas/.
2. Nel pannello di navigazione, scegliere servizi AWS .
3. Selezionare AWS Lambda.
4. Seleziona la quota che desideri aumentare.
5. Scegliere Request quota increase (Richiedi aumento di quota).

6. Inserisci il nuovo valore della quota e fornisci una giustificazione per l'aumento.
7. Scegli Richiedi.

Fasi successive

- Scopri i provider di capacità per le istanze gestite Lambda
- Comprendi la scalabilità per le istanze gestite Lambda
- Consulta le guide specifiche di runtime per Java, Node.js e Python
- Configura la connettività VPC per i tuoi provider di capacità

Le migliori pratiche per le istanze gestite Lambda

Configurazione del provider di capacità

Separa i fornitori di capacità in base al livello di fiducia. Crea diversi provider di capacità per carichi di lavoro con requisiti di sicurezza diversi. Tutte le funzioni assegnate allo stesso fornitore di capacità devono essere considerate affidabili da entrambe le parti, in quanto i fornitori di capacità fungono da limite di sicurezza.

Usa nomi descrittivi. Denominate i fornitori di capacità per indicare chiaramente l'uso previsto e il livello di affidabilità (ad esempio`production-trusted`,`dev-sandbox`). Questo aiuta i team a comprendere lo scopo e il livello di sicurezza di ciascun fornitore di capacità.

Utilizza più zone di disponibilità. Specificate le sottoreti su più zone di disponibilità durante la creazione di provider di capacità. Lambda lancia tre istanze per impostazione predefinita per la resilienza AZ, garantendo un'elevata disponibilità delle funzioni.

Selezione del tipo di istanza

Consenti a Lambda di scegliere i tipi di istanza. Per impostazione predefinita, Lambda sceglie i tipi di istanza migliori per il tuo carico di lavoro. Ti consigliamo di lasciare che le istanze gestite da Lambda scelgano i tipi di istanza per te, poiché limitare il numero di tipi di istanze possibili può comportare una minore disponibilità.

Specificate i tipi di istanza per requisiti specifici. Se hai requisiti hardware specifici, imposta i tipi di istanze consentiti su un elenco di istanze compatibili. Esempio:

- Per le applicazioni che richiedono un'elevata larghezza di banda di rete, seleziona diversi tipi di istanze n
- Per ambienti di test o sviluppo con vincoli di costo, scegli tipi di istanze più piccoli come m7a.large

Configurazione delle funzioni

Scegli le impostazioni di memoria e vCPU appropriate. Seleziona configurazioni di memoria e vCPU che supportano le esecuzioni simultanee della tua funzione. La dimensione minima della funzione supportata è 2 GB e 1 vCPU.

- Per le applicazioni Python, scegli un rapporto più elevato tra memoria e v CPUs (ad esempio 4 a 1 o 8 a 1) a causa del modo in cui Python gestisce la multi-concorrenza
- Per operazioni che richiedono un uso intensivo della CPU o funzioni che eseguono poco I/O, scegli più di una vCPU
- Per le applicazioni che richiedono un elevato livello di I/O, come i servizi Web o i lavori in batch, la multi-concorrenza offre i maggiori vantaggi

Configura la massima concorrenza in modo appropriato. Lambda sceglie impostazioni predefinite ragionevoli per la massima concorrenza che bilanciano il consumo di risorse e la velocità effettiva. Modifica questa impostazione in base all'utilizzo delle risorse della tua funzione:

- Aumenta la concorrenza massima (fino a 64 per vCPU) se le chiamate alle funzioni utilizzano pochissima CPU
- Riduci la concorrenza massima se l'applicazione consuma una grande quantità di memoria e pochissima CPU

Tieni presente che gli ambienti di esecuzione con una concorrenza molto bassa possono presentare rallentamenti e difficoltà di scalabilità.

Scaling configuration (Configurazione dimensionamento)

Imposta l'utilizzo appropriato delle risorse target. Per impostazione predefinita, Lambda offre un margine di manovra sufficiente per raddoppiare il traffico entro 5 minuti senza acceleratori. Modifica questa impostazione in base alle caratteristiche del carico di lavoro:

- Per carichi di lavoro molto stabili o applicazioni non sensibili alle limitazioni di velocità, stabilisci l'obiettivo su un livello elevato per ottenere un maggiore utilizzo e ridurre i costi
- Per i carichi di lavoro con potenziali aumenti di traffico, stabilisci obiettivi di risorse a un livello basso per mantenere un margine di crescita aggiuntivo

Pianifica la crescita del traffico. Se il traffico raddoppia più che nel giro di 5 minuti, potresti riscontrare rallentamenti man mano che Lambda aumenta la scalabilità delle istanze e degli ambienti di esecuzione. Progetta la tua applicazione per gestire le potenziali limitazioni durante i periodi di rapida scalabilità.

Sicurezza

Applica il privilegio minimo per le autorizzazioni. PassCapacityProvider Concedi `lambda:PassCapacityProvider` le autorizzazioni solo ai fornitori di capacità necessari. Utilizza le autorizzazioni a livello di risorsa per limitare i fornitori di capacità che gli utenti possono assegnare alle funzioni.

Monitora l'utilizzo dei provider di capacità. Utilizzato AWS CloudTrail per monitorare le assegnazioni dei provider di capacità e i modelli di accesso. Questo aiuta a identificare i tentativi di accesso non autorizzati e garantisce la conformità alle politiche di sicurezza.

Separa i carichi di lavoro non affidabili. Non fare affidamento sui contenitori per l'isolamento di sicurezza tra carichi di lavoro non affidabili. Utilizza diversi provider di capacità per separare i carichi di lavoro che non sono reciprocamente affidabili.

Ottimizzazione dei costi

Sfrutta le opzioni EC2 di prezzo. Sfrutta i vantaggi di EC2 Savings Plans e Reserved Instances per ridurre i costi. Queste opzioni di prezzo si applicano all' EC2 elaborazione sottostante (la commissione di gestione del 15% non è scontata).

Ottimizza per carichi di lavoro stazionari. Le istanze gestite Lambda sono più adatte per funzioni stazionarie con traffico prevedibile ad alto volume. Per modelli di traffico a raffica, Lambda (impostazione predefinita) può essere più conveniente.

Monitora l'utilizzo delle risorse. Tieni traccia delle CloudWatch metriche per comprendere l'utilizzo della CPU e della memoria. Modifica l'allocazione della memoria delle funzioni e la selezione del tipo di istanza in base ai modelli di utilizzo effettivi per ottimizzare i costi.

Monitoraggio e osservabilità

Monitora le metriche dei fornitori di capacità. Tieni traccia delle metriche a livello di provider di capacità CPUUtilization MemoryUtilization, tra cuiCPUAvailable, v e verifica MemoryAvailable che siano disponibili risorse sufficienti per i tuoi carichi di lavoro.

Monitora le metriche dell'ambiente di esecuzione. Tieni traccia delle metriche a livello di ambiente di esecuzione, tra cui ExecutionEnvironmentConcurrency e ExecutionEnvironmentConcurrencyLimit per comprendere il comportamento di scalabilità e identificare potenziali limitazioni.

Imposta allarmi. CloudWatch Crea CloudWatch allarmi per le metriche chiave per identificare in modo proattivo i problemi:

- Utilizzo elevato della CPU o della memoria
- Bassa capacità disponibile
- Ci avviciniamo ai limiti di concorrenza

Considerazioni specifiche sul runtime

Segui le best practice specifiche per il runtime. Ogni runtime gestisce la concorrenza multipla in modo diverso. Consulta le guide specifiche per il runtime per consigli dettagliati:

- Java: utilizza raccolte thread-safe e per lo stato specifico della richiesta `AtomicInteger` `ThreadLocal`
- Node.js: da utilizzare InvokeStore per tutti gli stati specifici della richiesta ed evitare le variabili globali
- Python: usa nomi di file univoci nella richiesta IDs e considera `/tmp` l'isolamento della memoria basato sul processo

Verifica la sicurezza dei thread e i problemi di concorrenza. Prima di passare alla produzione, testate accuratamente le vostre funzioni per verificare eventuali problemi di sicurezza dei thread, le condizioni di gara e il corretto isolamento dello stato in presenza di carichi simultanei.

Fasi successive

- Scopri i provider di capacità per le istanze gestite Lambda
- Comprendi la scalabilità per le istanze gestite Lambda

- Consulta le guide specifiche di runtime per Java, Node.js e Python
- Configura la connettività VPC per i tuoi provider di capacità
- Monitora le istanze gestite Lambda con metriche CloudWatch

Risoluzione dei problemi relativi alle istanze gestite da Lambda

Problemi di limitazione e scalabilità

Tassi di errore elevati durante lo scale-up

Problema: si verificano errori di limitazione (HTTP 429) quando il traffico aumenta rapidamente.

Causa: le istanze gestite Lambda scalano in modo asincrono in base all'utilizzo delle risorse della CPU e alla saturazione multi-concorrenza. Se il traffico raddoppia più che nel giro di 5 minuti, potresti riscontrare rallentamenti man mano che Lambda ridimensiona le istanze e gli ambienti di esecuzione per soddisfare la domanda.

Soluzione::

- Modifica l'utilizzo delle risorse target: se il carico di lavoro presenta modelli di traffico prevedibili, imposta un utilizzo delle risorse target inferiore per mantenere un margine di manovra aggiuntivo in caso di picchi di traffico.
- Capacità di preriscaldamento: per gli aumenti di traffico pianificati, aumenta gradualmente il traffico su un periodo più lungo per consentire alla scalabilità di tenere il passo.
- Monitora le metriche di scalabilità: tieni traccia delle metriche degli errori di accelerazione per comprendere il motivo delle limitazioni e dei problemi di scalabilità della capacità.
- Rivedi la configurazione delle funzioni: assicurati che le impostazioni della memoria delle funzioni e della vCPU supportino le esecuzioni simultanee multiple. Se necessario, aumentare la memoria funzionale o l'allocazione della vCPU.

Ridimensionamento lento

Problema: una volta diminuito il traffico, le istanze impiegano molto tempo a ridursi.

Causa: le istanze gestite Lambda si ridimensionano gradualmente per mantenere la disponibilità ed evitare rapidi cambiamenti di capacità che potrebbero influire sulle prestazioni.

Soluzione::

Questo è il comportamento previsto. Lambda riduce le istanze in modo conservativo per garantire la stabilità. Monitora le tue CloudWatch metriche per tenere traccia del numero di istanze in esecuzione.

Problemi di concorrenza

Ambienti di esecuzione con bassa esperienza di concorrenza, limiti

Problema: nonostante la capacità disponibile, le funzioni subiscono delle limitazioni.

Causa: gli ambienti di esecuzione con una concorrenza massima molto bassa possono avere difficoltà a scalare in modo efficace. Le istanze gestite Lambda sono progettate per applicazioni multi-concorrenti.

Soluzione::

- Aumenta la concorrenza massima: se le chiamate alle funzioni utilizzano pochissima CPU, aumenta l'impostazione della concorrenza massima fino a 64 per vCPU.
- Ottimizza il codice della funzione: rivedi il codice della funzione per ridurre il consumo di CPU per chiamata e consentire una maggiore concorrenza.
- Regola la memoria delle funzioni e la vCPU: assicurati che la tua funzione disponga di risorse sufficienti per gestire più chiamate simultanee.

Problemi di sicurezza dei thread (runtime Java)

Problema: la funzione Java produce risultati errati o presenta condizioni di gara sotto carico.

Causa: più thread eseguono il metodo handler contemporaneamente e lo stato condiviso non è thread-safe.

Soluzione::

- Usa `AtomicInteger` o per i contatori anziché `AtomicLong` i tipi primitivi
- Sostituisci con `HashMap` `ConcurrentHashMap`
- Utilizzare `Collections.synchronizedList()` per avvolgere `ArrayList`
- Usa `ThreadLocal` per lo stato specifico della richiesta
- Accedi alla traccia IDs dall'oggetto Lambda Context, non dalle variabili di ambiente

Per una guida dettagliata, consulta la documentazione di Java runtime for Lambda Managed Instances.

Problemi di isolamento dello stato (runtime Node.js)

Problema: la funzione Node.js restituisce dati da richieste diverse o presenta un danneggiamento dei dati.

Causa: le variabili globali sono condivise tra chiamate simultanee sullo stesso thread di lavoro. Quando le operazioni asincrone danno il controllo, altre invocazioni possono modificare lo stato condiviso.

Soluzione::

- Installa e usa `@aws/lambda-invoke-store` per tutti gli stati specifici della richiesta
- Sostituisci le variabili globali con e `InvokeStore.set()` `InvokeStore.get()`
- Usa nomi di file univoci nella `/tmp` richiesta IDs
- Accedi alla traccia IDs utilizzando `InvokeStore.getXRayTraceId()` invece delle variabili di ambiente

Per una guida dettagliata, consulta la documentazione del runtime Node.js per le istanze gestite Lambda.

Conflitti tra file (runtime Python)

Problema: la tua funzione Python legge dati errati dai file in. `/tmp`

Causa: più processi condividono la `/tmp` directory. Le scritture simultanee sullo stesso file possono causare il danneggiamento dei dati.

Soluzione::

- Usa nomi di file univoci con richiesta IDs: `/tmp/request_{context.request_id}.txt`
- Usa il blocco dei file con `fcntl.flock()` per i file condivisi
- Pulisci i file temporanei con `os.remove()` dopo l'uso

Per una guida dettagliata, consulta la documentazione del runtime Python per le istanze gestite Lambda.

Problemi di prestazioni

Elevato utilizzo della memoria

Problema: le funzioni presentano un elevato utilizzo della memoria o out-of-memory errori.

Causa: ogni richiesta simultanea in Python viene eseguita in un processo separato con il proprio spazio di memoria. L'utilizzo totale della memoria è uguale alla memoria per processo moltiplicata per i processi simultanei.

Soluzione::

- `MemoryUtilization`Monitora la metrica in CloudWatch
- Riduci l'`MaxConcurrency`impostazione se l'utilizzo della memoria si avvicina al limite di memoria della funzione
- Aumenta l'allocazione della memoria funzionale per supportare una maggiore concorrenza
- Ottimizza l'utilizzo della memoria caricando i dati su richiesta anziché durante l'inizializzazione

Prestazioni incoerenti

Problema: le prestazioni delle funzioni variano in modo significativo tra le chiamate.

Causa: Lambda può selezionare diversi tipi di istanze in base alla disponibilità oppure le funzioni possono essere eseguite su istanze con disponibilità di risorse variabile.

Soluzione::

- Specificare i tipi di istanze consentiti: se hai requisiti prestazionali specifici, configura i tipi di istanza consentiti nel tuo provider di capacità per limitare i tipi di istanze che Lambda può selezionare.
- Monitora le metriche a livello di istanza: monitora `CPUUtilization` e `MemoryUtilization` a livello di provider di capacità per identificare i vincoli di risorse.
- Esamina le metriche sulla capacità: verifica `vCPUAvailable` e `MemoryAvailable` assicurati che siano disponibili risorse sufficienti sulle tue istanze.

Problemi con i fornitori di capacità

La versione della funzione non riesce a diventare ATTIVA

Problema: la versione della funzione rimane in sospeso dopo la pubblicazione.

Causa: Lambda sta avviando istanze gestite e avviando ambienti di esecuzione. Questo processo richiede tempo, soprattutto per la prima versione della funzione su un nuovo provider di capacità.

Soluzione::

Attendi che Lambda completi il processo di inizializzazione. Lambda avvia tre istanze per impostazione predefinita per la resilienza AZ e avvia tre ambienti di esecuzione prima di contrassegnare la versione della funzione come ATTIVA. Questo richiede in genere diversi minuti.

Impossibile eliminare il provider di capacità

Problema: viene visualizzato un errore quando si tenta di eliminare un provider di capacità.

Causa: non è possibile eliminare un provider di capacità a cui sono associate versioni di funzioni.

Soluzione::

1. Identifica tutte le versioni delle funzioni utilizzando il provider di capacità con l'`ListFunctionVersionsByCapacityProvider`API.
2. Elimina o aggiorna le versioni di tali funzioni per rimuovere l'associazione del fornitore di capacità.
3. Riprova a eliminare il provider di capacità.

Messaggi di errore generici durante la pubblicazione delle funzioni

Problema: durante la pubblicazione delle funzioni vengono visualizzati messaggi di errore generici come «Si è verificato un errore interno durante la pubblicazione».

Soluzione::

- Verifica le autorizzazioni IAM: assicurati di disporre dell'`lambda:PassCapacityProvider`autorizzazione per il provider di capacità che stai cercando di utilizzare.
- Verifica la configurazione del provider di capacità: conferma che il tuo fornitore di capacità sia nello stato ATTIVO utilizzando l'`GetCapacityProvider`API.

- Rivedi la configurazione del VPC: assicurati che le sottoreti e i gruppi di sicurezza specificati nel tuo provider di capacità siano configurati e accessibili correttamente.
- Controlla AWS CloudTrail i log: consulta i CloudTrail log per informazioni dettagliate sugli errori relativi all'operazione non riuscita.

Problemi di monitoraggio e osservabilità

Metriche mancanti CloudWatch

Problema: non vedi le metriche previste CloudWatch per il tuo capacity provider o per le tue funzioni.

Causa: le metriche vengono pubblicate a intervalli di 5 minuti. I nuovi fornitori di capacità o le nuove funzioni potrebbero non disporre di metriche immediatamente disponibili.

Soluzione::

Attendi almeno 5-10 minuti dopo la pubblicazione della versione di una funzione prima di aspettarti che le metriche vengano visualizzate in. CloudWatch Verifica di utilizzare lo spazio dei nomi (`AWS/Lambda`) e le dimensioni (`CapacityProviderName`, o) corretti. `FunctionName` `InstanceType`

Impossibile trovare i log CloudWatch

Problema: la funzione viene eseguita correttamente, ma non è possibile trovare i log in Logs. CloudWatch

Causa: le istanze gestite Lambda vengono eseguite nel tuo VPC e richiedono una connettività di rete per inviare i log ai log. CloudWatch Senza una corretta configurazione della connettività VPC, le funzioni non possono raggiungere l'endpoint del servizio CloudWatch Logs.

Soluzione::

Configura la connettività VPC per consentire alle tue funzioni di inviare log a Logs. CloudWatch Sono disponibili tre opzioni:

Opzione 1: endpoint VPC per CloudWatch registri (consigliato per la produzione)

1. Apri la console Amazon VPC all'indirizzo console.aws.amazon.com/vpc/.
2. Nel pannello di navigazione, seleziona Endpoints (Endpoint).
3. Seleziona Crea endpoint.

4. Per Service category (Categoria servizio), scegli AWS services.
5. Per Nome del servizio, seleziona (sostituisci con la tua regione). `com.amazonaws.region.logs` `region` AWS
6. Per VPC, seleziona il VPC utilizzato dal tuo provider di capacità.
7. Per le sottoreti, seleziona le sottoreti in cui desideri creare interfacce di rete endpoint. Per un'elevata disponibilità, seleziona le sottoreti in più zone di disponibilità.
8. Per i gruppi di sicurezza, seleziona i gruppi di sicurezza che consentono il traffico HTTPS in entrata (porta 443) dal gruppo di sicurezza della funzione.
9. Abilita il DNS privato per l'endpoint.
10. Seleziona Crea endpoint.

Opzione 2: sottorete pubblica con gateway Internet

Se il tuo fornitore di capacità utilizza sottoreti pubbliche, assicurati che:

1. Un gateway Internet è collegato al tuo VPC
2. La tabella delle rotte indirizza `0.0.0.0/0` il traffico verso il gateway Internet
3. I gruppi di sicurezza consentono il traffico HTTPS in uscita sulla porta 443

Opzione 3: sottorete privata con gateway NAT

Se il tuo fornitore di capacità utilizza sottoreti private, assicurati di:

1. Un gateway NAT esiste in una sottorete pubblica
2. La tabella di routing della sottorete privata indirizza il `0.0.0.0/0` traffico verso il gateway NAT
3. La tabella di routing della sottorete pubblica indirizza il `0.0.0.0/0` traffico verso un gateway Internet
4. I gruppi di sicurezza consentono il traffico HTTPS in uscita sulla porta 443

Per una guida dettagliata sulle opzioni di connettività VPC, consulta Connettività VPC per istanze gestite Lambda.

Difficoltà a correlare i log delle richieste simultanee

Problema: i log di richieste diverse sono interlacciati, il che rende difficile tracciare le singole richieste.

Causa: l'interlacciamento dei log è un comportamento previsto e standard nei sistemi multi-concorrenti.

Soluzione::

- Usa la registrazione strutturata con formato JSON: includi l'ID della richiesta in tutte le istruzioni di registro
- Java: usa Log4j con `ThreadContext` per includere automaticamente l'ID della richiesta
- Node.js: da utilizzare `console.log()` con la formattazione JSON e includere `InvokeStore.getRequestId()`
- Python: usa il modulo di registrazione standard con formattazione JSON e includi `context.request_id`

Per una guida dettagliata, consulta le pagine di documentazione specifiche del runtime.

Ottenere ulteriore assistenza

Se continui a riscontrare problemi dopo aver provato queste soluzioni:

1. Esamina CloudWatch le metriche: verifica le metriche del provider di capacità e dell'ambiente di esecuzione per identificare i vincoli di risorse o i problemi di scalabilità.
2. Controlla AWS CloudTrail i log: consulta i CloudTrail log per informazioni dettagliate su chiamate ed errori delle API.
3. Contatta l' AWS assistenza: se non riesci a risolvere il problema, contatta l' AWS assistenza per fornire dettagli sulla configurazione del tuo provider di capacità, sulla configurazione delle funzioni e sui messaggi di errore specifici che stai riscontrando.

Fasi successive

- Scopri i provider di capacità per le istanze gestite Lambda
- Comprendi la scalabilità per le istanze gestite Lambda
- Consulta le guide specifiche di runtime per Java, Node.js e Python
- Monitora le istanze gestite Lambda con metriche CloudWatch
- Consulta le best practice per le istanze gestite Lambda

Runtime Lambda

Lambda supporta più lingue attraverso l'uso di runtime. Un runtime fornisce un ambiente specifico del linguaggio che inoltra gli eventi di chiamata, le informazioni di contesto e le risposte tra Lambda e la funzione. Puoi utilizzare i runtime forniti da Lambda o puoi crearne uno personalizzato.

Lambda è indipendente dalla tua scelta di runtime. Per funzioni semplici, i linguaggi interpretati come Python e Node.js offrono le prestazioni più veloci. Per le funzioni con calcoli più complessi, i linguaggi compilati come Java sono spesso più lenti da inizializzare ma vengono eseguiti rapidamente nell'handler Lambda. La scelta del runtime è influenzata anche dalle preferenze degli sviluppatori e dalla familiarità del linguaggio.

Ogni versione del linguaggio di programmazione principale ha un runtime separato, con un identificatore di runtime univoco, ad esempio `nodejs24.x` o `python3.14`. Per configurare una funzione in modo da utilizzare una nuova versione principale del linguaggio, è necessario modificare l'identificatore di runtime. Poiché AWS Lambda non è possibile garantire la retrocompatibilità tra le versioni principali, si tratta di un'operazione gestita dal cliente.

Per una funzione definita come immagine di container, scegli un runtime e la distribuzione Linux quando crei l'immagine di container. Per modificare il runtime, è necessario creare una nuova immagine di container.

Quando si utilizza un archivio di file con estensione .zip per il pacchetto di implementazione, è necessario scegliere un runtime quando si crea la funzione. Per modificare il runtime, è possibile aggiornare la configurazione della funzione. Il runtime è associato a una delle distribuzioni Amazon Linux. L'ambiente di esecuzione sottostante fornisce ulteriori librerie e variabili di ambiente alle quali è possibile accedere dal codice della funzione.

Lambda richiama la funzione in un ambiente di esecuzione. Un ambiente di esecuzione fornisce un ambiente di runtime sicuro e isolato che gestisce le risorse necessarie per eseguire la funzione. Lambda riutilizza l'ambiente di esecuzione da una chiamata precedente, se disponibile, oppure può creare un nuovo ambiente di esecuzione.

Per utilizzare altri linguaggi in Lambda, come Go o Rust, utilizza un runtime solo per il sistema operativo. L'ambiente di esecuzione Lambda offre un'interfaccia di runtime per ricevere gli eventi di chiamata e inviare le risposte. Puoi implementare altri linguaggi implementando un runtime personalizzato insieme al codice della funzione o in un livello.

Runtime supportati

Nella tabella seguente sono elencati i runtime Lambda supportati e le date di ritiro previste. Una volta ritirato un runtime, è ancora possibile creare e aggiornare le funzioni per un periodo limitato. Per ulteriori informazioni, consulta the section called “Utilizzo del runtime dopo la deprecazione”. La tabella fornisce le date attualmente previste per l'obsolescenza del runtime, in base alle nostre. the section called “Policy di deprecazione del runtime” Tali date vengono indicate a scopo di pianificazione e sono soggette a modifiche.

> **Important**
>
> La fine del ciclo di vita di Amazon Linux 2 è prevista per il 30 giugno 2026. I runtime Lambda e le immagini base dei container per Java 8 (AL2), Java 11, Java 17, Python 3.10, Python 3.11 e provided.al2 continueranno a ricevere patch per importanti problemi di sicurezza di Amazon Linux 2 critici e selezionati, oltre alle patch di runtime del linguaggio, fino alle date di deprecazione indicate nella tabella seguente.
> Consigliamo ai clienti di eseguire l'aggiornamento a un runtime basato su Amazon Linux 2023 il prima possibile. Per i clienti che effettuano l'aggiornamento a Java 21 o Java 25, è possibile utilizzare AWS Transform custom per fornire assistenza con questi aggiornamenti. Per i clienti che non possono aggiornare la loro versione Java, prevediamo di rilasciare i runtime basati su Amazon Linux 2023 per Java 8, Java 11 e Java 17 prima della fine del secondo trimestre 2026.

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Node.js 24	`nodejs24.x`	Amazon Linux 2023	30 aprile 2028	1 giugno 2028	1 luglio 2028
Node.js 22	`nodejs22.x`	Amazon Linux 2023	30 aprile 2027	1 giugno 2027	1 luglio 2027
Node.js 20	`nodejs20.x`	Amazon Linux 2023	30 aprile 2026	31 agosto 2026	30 settembre 2026

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Python 3.14	`python3.14`	Amazon Linux 2023	30 giugno 2029	31 luglio 2029	31 agosto 2029
Python 3.13	`python3.13`	Amazon Linux 2023	30 giugno 2029	31 luglio 2029	31 agosto 2029
Python 3.12	`python3.12`	Amazon Linux 2023	31 ottobre 2028	30 novembre 2028	10 gennaio 2029
Python 3.11	`python3.11`	Amazon Linux 2	30 giugno 2027	31 luglio 2027	31 agosto 2027
Python 3.10	`python3.10`	Amazon Linux 2	31 ottobre 2026	30 novembre 2026	15 gennaio 2027
Giava 25	`java25`	Amazon Linux 2023	30 giugno 2029	31 luglio 2029	31 agosto 2029
Java 21	`java21`	Amazon Linux 2023	30 giugno 2029	31 luglio 2029	31 agosto 2029
Java 17	`java17`	Amazon Linux 2	30 giugno 2027	31 luglio 2027	31 agosto 2027
Java 11	`java11`	Amazon Linux 2	30 giugno 2027	31 luglio 2027	31 agosto 2027
Java 8	`java8.al2`	Amazon Linux 2	30 giugno 2027	31 luglio 2027	31 agosto 2027
.NET 10	`dotnet10`	Amazon Linux 2023	14 novembre 2028	14 dicembre 2028	15 gennaio 2029
.NET 9 (solo contenitore)	`dotnet9`	Amazon Linux 2023	10 novembre 2026	Non programmato	Non programmato

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
.NET 8	`dotnet8`	Amazon Linux 2023	10 novembre 2026	10 dicembre 2026	11 gennaio 2027
Rubino 3.4	`ruby3.4`	Amazon Linux 2023	31 marzo 2028	30 aprile 2028	31 maggio 2028
Ruby 3.3	`ruby3.3`	Amazon Linux 2023	31 marzo 2027	30 aprile 2027	31 maggio 2027
Ruby 3.2	`ruby3.2`	Amazon Linux 2	31 marzo 2026	31 agosto 2026	30 settembre 2026
Runtime solo per il sistema operativo	`provided.al2023`	Amazon Linux 2023	30 giugno 2029	31 luglio 2029	31 agosto 2029
Runtime solo per il sistema operativo	`provided.al2`	Amazon Linux 2	31 luglio 2026	31 agosto 2026	30 settembre 2026

> **Note**
>
> Per le nuove regioni, Lambda non supporterà i runtime per cui è previsto il ritiro entro i prossimi 6 mesi.

Lambda mantiene aggiornati i runtime gestiti e le immagini di base dei container corrispondenti con patch e supporto per le versioni minori. Per ulteriori informazioni, consulta Aggiornamenti di runtime Lambda.

Per interagire a livello di codice con altre Servizi AWS risorse della funzione Lambda, puoi usare una delle. AWS SDKs I runtime Node.js, Python e Ruby includono una versione dell'SDK. AWS Tuttavia, per mantenere il pieno controllo delle dipendenze e massimizzare la compatibilità con le versioni precedenti durante gli aggiornamenti automatici del runtime, ti consigliamo di includere sempre i

moduli SDK utilizzati dal codice (insieme a qualsiasi dipendenza) nel pacchetto di distribuzione della funzione o in un livello Lambda.

Ti consigliamo di utilizzare la versione SDK inclusa nel runtime solo quando non puoi includere pacchetti aggiuntivi nella distribuzione. Ad esempio, quando crei la tua funzione utilizzando l'editor di codice della console Lambda o utilizzando il codice di funzione in linea in un modello. CloudFormation

Lambda aggiorna periodicamente le versioni AWS SDKs incluse nei runtime Node.js, Python e Ruby. Per determinare la versione dell' AWS SDK inclusa nel runtime che stai utilizzando, consulta le seguenti sezioni:

- Versioni SDK incluse nel runtime (Node.js)
- Versioni SDK incluse nel runtime (Python)
- Versioni SDK incluse nel runtime (Ruby)

Lambda continua a supportare il linguaggio di programmazione Go dopo il ritiro del runtime Go 1.x. Per ulteriori informazioni, consulta la sezione Migrazione AWS Lambda delle funzioni dal runtime Go1.x al runtime personalizzato su Amazon Linux 2 sul blog di Compute AWS .

Tutti i runtime Lambda supportati supportano sia le architetture x86_64 che le architetture arm64.

Nuovi rilasci di runtime

Lambda fornisce runtime gestiti per le nuove versioni di linguaggio solo quando il rilascio raggiunge la fase di supporto a lungo termine (LTS) del ciclo di rilascio del linguaggio. Ad esempio, per il ciclo di rilascio di Node.js, quando il rilascio raggiunge la fase Active LTS.

Prima che raggiunga la fase di supporto a lungo termine, il rilascio rimane in fase di sviluppo e può ancora essere soggetto a modifiche sostanziali. Lambda applica in automatico gli aggiornamenti di runtime per impostazione predefinita, quindi le modifiche sostanziali a una versione di runtime potrebbe impedire alle funzioni di funzionare come previsto.

Lambda non fornisce runtime gestiti per le versioni di linguaggio che non sono pianificate per il rilascio LTS.

L'elenco seguente mostra il mese di lancio previsto per i runtime Lambda futuri. Queste date sono solo indicative e soggette a modifica.

- Ruby 3.5 - marzo 2026
- Java 8, 11 e 17 nel 2023 - 2° AL2 trimestre 2026
- Node.js 26 - novembre 2026
- Python 3.15 - novembre 2026

Policy di deprecazione del runtime

I runtime Lambda per gli archivi di file.zip si basano su una combinazione di sistema operativo, linguaggio di programmazione e librerie software soggette a manutenzione e aggiornamenti di sicurezza. In base alla policy di deprecazione standard di Lambda, un runtime diventa obsoleto quando uno dei suoi componenti principali raggiunge la fine del supporto a lungo termine (LTS) della community e gli aggiornamenti di sicurezza non sono più disponibili. In genere si tratta del runtime del linguaggio, anche se in alcuni casi un runtime può diventare obsoleto perché il sistema operativo (OS) raggiunge la fine dell'LTS.

Dopo che un runtime è diventato obsoleto, non AWS possono più applicare patch o aggiornamenti di sicurezza a quel runtime e le funzioni che utilizzano quel runtime non sono più idonee per il supporto tecnico. Tali runtime obsoleti vengono forniti "così come sono", senza alcuna garanzia e possono contenere bug, errori, difetti o altre vulnerabilità.

Per ulteriori informazioni sulla gestione degli aggiornamenti di runtime e sulla deprecazione, consulta le seguenti sezioni e la gestione degli aggiornamenti di runtime sul blog di Compute. AWS Lambda AWS

> **Important**
>
> Occasionalmente, Lambda ritarda la deprecazione di un runtime Lambda per un periodo limitato oltre la data di fine del supporto della versione di linguaggio supportata dal runtime. Durante questo periodo, Lambda applica le patch di sicurezza soltanto al sistema operativo runtime. Una volta raggiunta la data di fine del supporto, Lambda non applica le patch di sicurezza ai runtime dei linguaggi di programmazione.

Modello di responsabilità condivisa

Lambda è responsabile della cura e della pubblicazione degli aggiornamenti di sicurezza per tutti i runtime gestiti e le immagini dei container supportati. Per impostazione predefinita, Lambda applica

automaticamente questi aggiornamenti alle funzioni che utilizzano runtime gestiti. Se l'impostazione predefinita per l'aggiornamento automatico del runtime è stata modificata, consulta il modello di responsabilità condivisa dei controlli di gestione del runtime. Per le funzioni implementate utilizzando immagini di container, l'utente sarà responsabile della ricostruzione dell'immagine di container della funzione dall'immagine di base più recente e della nuova implementazione dell'immagine di container.

Quando un runtime diventa obsoleto, la responsabilità di Lambda per l'aggiornamento del runtime gestito e delle immagini di base del container cessa. L'utente è responsabile dell'aggiornamento delle funzioni per utilizzare un runtime o un'immagine di base supportata.

In tutti i casi, l'utente è responsabile dell'applicazione degli aggiornamenti al codice della funzione, comprese le sue dipendenze. Le tue responsabilità nell'ambito del modello di responsabilità condivisa sono riassunte nella tabella seguente.

Fase del ciclo di vita di runtime	Responsabilità di Lambda	Le tue responsabilità
Runtime gestito supportato	Fornisci aggiornamenti di runtime regolari con patch di sicurezza e altri aggiornamenti. Applica automaticamente gli aggiornamenti di runtime per impostazione predefinita (vedi the section called “Modalità di aggiornamento del runtime” per i comportamenti non predefiniti).	Aggiorna il codice della funzione, comprese le dipendenze, per risolvere eventuali vulnerabilità di sicurezza.
Immagine di container supportata	Fornisci aggiornamenti di runtime regolari all'immagine di base del container con patch di sicurezza e altri aggiornamenti.	Aggiorna il codice della funzione, comprese le dipendenze, per risolvere eventuali vulnerabilità di sicurezza. Ricostruisci e reimplementa regolarmente l'immagine di

Fase del ciclo di vita di runtime	Responsabilità di Lambda	Le tue responsabilità
		container utilizzando l'immagin e di base più recente.
Runtime gestito prossimo alla deprecazione	Avvisate i clienti prima della deprecazione del runtime tramite documentazione, e-mail e. Health Dashboard Trusted Advisor La responsabilità per gli aggiornamenti del runtime termina quando il runtime diventa obsoleto.	Monitora la documentazione di Lambda Health Dashboard, la posta elettronica o le informazi oni sull'obsolescenza in fase Trusted Advisor di esecuzione. Aggiorna le funzioni a un runtime supportato prima che il runtime precedente diventi obsoleto.
Immagine di container che si avvicina alla deprecazione	Le notifiche di deprecazi one non sono disponibili per le funzioni che utilizzano immagini di container. La responsabilità per gli aggiornamenti dell'immagine di base del container termina quando il runtime diventa obsoleto.	Fai attenzione alle pianifica zioni di deprecazione e alle funzioni di aggiornamento a un'immagine di base supportat a prima che l'immagine precedente diventi obsoleta.

Utilizzo del runtime dopo la deprecazione

Dopo che un runtime è diventato obsoleto, non AWS possono più applicare patch di sicurezza o aggiornamenti a quel runtime e le funzioni che utilizzano quel runtime non sono più idonee al supporto tecnico. Sebbene sia possibile continuare a richiamare le funzioni a tempo indeterminato, si consiglia AWS vivamente di migrare a un runtime supportato. I runtime obsoleti vengono forniti «così come sono», senza alcuna garanzia e possono contenere bug, errori, difetti o altre vulnerabilità. Le funzioni che utilizzano un runtime obsoleto possono inoltre presentare un peggioramento delle prestazioni o altri problemi, come la scadenza del certificato, che possono impedirne il corretto funzionamento.

È possibile aggiornare una funzione per utilizzare un runtime supportato più recente in qualsiasi momento dopo che un runtime è diventato obsoleto. Consigliamo di testare la funzione con il nuovo runtime prima di applicare le modifiche negli ambienti di produzione. Non sarà possibile tornare al runtime obsoleto dopo il blocco degli aggiornamenti delle funzioni. Consigliamo di utilizzare le versioni e gli alias delle funzioni per consentire una implementazione sicura con rollback.

La seguente sequenza temporale descrive cosa succede quando un runtime è obsoleto:

Fase del ciclo di vita di runtime	Quando	What
Periodo di avviso di obsolescenza	Almeno 180 giorni prima della deprecazione	• AWS invia notifiche tramite e-mail e Health Dashboard agli account che dispongono di funzioni che utilizzano questo runtime nella loro versione. `$LATEST` • Le funzioni interessate sono elencate anche nella scheda Modifiche Health Dashboard pianificate e nel controllo dei AWS Trusted Advisor runtime obsoleti.
Raggiunta obsolescenza	Data di ritiro	• AWS potrebbe non applicare più aggiornamenti di sicurezza o altri aggiornamenti. • Le funzioni non sono più idonee al supporto tecnico. • Non è più possibile creare o aggiornare funzioni utilizzando il runtime obsoleto nella console Lambda. Puoi continuare a creare e aggiornare funzioni tramite, o. AWS CLI AWS SAM CloudFormation
Blocco creazione funzioni	Almeno 30 giorni dopo la deprecazione	• Lambda inizia a bloccare la creazione di nuove funzioni. • Puoi continuare ad aggiornare il codice e la configurazione per le funzioni esistenti tramite AWS CLI AWS SAM, o.. CloudFormation

Fase del ciclo di vita di runtime	Quando	What
Blocco aggiornamento funzioni	Almeno 60 giorni dopo la deprecazione	• Lambda inizia a bloccare l'aggiornamento del codice e della configurazione per le funzioni esistenti. • Puoi comunque aggiornare la configura zione della funzione a un runtime supportat o. Tuttavia, il ripristino del runtime obsoleto potrebbe essere bloccato.

> **Note**
>
> Per alcuni runtime, AWS posticipa block-function-update le date di fine oltre i normali 30 block-function-create e 60 giorni dopo la deprecazione. AWS ha apportato questa modifica in risposta al feedback dei clienti per darti più tempo per aggiornare le tue funzioni. Fai riferimento alle tabelle riportate in the section called “Runtime supportati” e the section called “Runtime obsoleti” per vedere le date dei runtime. Lambda non inizierà a bloccare la creazione o l'aggiornamento delle funzioni prima delle date indicate in queste tabelle.

Ricezione di notifiche di ritiro del runtime

Quando un runtime si avvicina alla data di deprecazione, Lambda invia un avviso e-mail se alcune funzioni del runtime utilizzano quel runtime. Account AWS Le notifiche vengono visualizzate anche in e in. Health Dashboard AWS Trusted Advisor

- Ricezione di notifiche e-mail:

  Lambda ti invia un avviso e-mail almeno 180 giorni prima che un runtime venga ritirato. Questa e-mail elenca le versioni $LATEST di tutte le funzioni che utilizzano il runtime. Per visualizzare un elenco completo delle versioni delle funzioni interessate, usa Trusted Advisor o seethe section called “Ottenere dati sulle funzioni in base al runtime”.

  Lambda invia una notifica via e-mail al contatto principale Account AWS del tuo account. Per informazioni sulla visualizzazione o l'aggiornamento degli indirizzi e-mail nel tuo account, consulta la pagina Aggiornamento delle informazioni di contatto della Guida generale di AWS .

- Ricezione di notifiche tramite: Health Dashboard

  Health Dashboard Visualizza una notifica almeno 180 giorni prima che un runtime diventi obsoleto. Le notifiche compaiono nella pagina Stato del tuo account, in Altre notifiche. La scheda Risorse interessate della notifica elenca le versioni $LATEST di tutte le funzioni che utilizzano il runtime.

  > **Note**
  >
  > Per visualizzare un up-to-date elenco completo delle versioni delle funzioni interessate, usa Trusted Advisor or see. the section called “Ottenere dati sulle funzioni in base al runtime”

  Health Dashboard le notifiche scadono 90 giorni dopo che il runtime interessato è diventato obsoleto.

- Usando AWS Trusted Advisor

  Trusted Advisor visualizza una notifica almeno 180 giorni prima che un runtime diventi obsoleto. Le notifiche compaiono nella pagina Sicurezza. Un elenco delle funzioni interessate viene visualizzato in Funzioni AWS Lambda che utilizzano runtime ritirati. Questo elenco di funzioni mostra sia le versioni $LATEST sia le versioni pubblicate e si aggiorna in automatico per riflettere lo stato attuale delle funzioni.

  Puoi attivare le notifiche e-mail settimanali dalla Trusted Advisor pagina Preferenze della console. Trusted Advisor

Runtime obsoleti

Per i seguenti runtime è stata raggiunta la fine del supporto:

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Python 3.9	`python3.9`	Amazon Linux 2	15 dicembre 2025	31 agosto 2026	30 settembre 2026

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Node.js 18	`nodejs18.x`	Amazon Linux 2	1 settembre 2025	31 agosto 2026	30 settembre 2026
.NET 6	`dotnet6`	Amazon Linux 2	20 dicembre 2024	31 agosto 2026	30 settembre 2026
Python 3.8	`python3.8`	Amazon Linux 2	14 ottobre 2024	31 agosto 2026	30 settembre 2026
Node.js 16	`nodejs16.x`	Amazon Linux 2	12 giugno 2024	31 agosto 2026	30 settembre 2026
.NET 7 (solo container)	`dotnet7`	Amazon Linux 2	14 maggio 2024	N/D	N/D
Java 8	`java8`	Amazon Linux	8 gennaio 2024	8 febbraio 2024	30 settembre 2026
Go 1.x	`go1.x`	Amazon Linux	8 gennaio 2024	8 febbraio 2024	30 settembre 2026
Runtime solo per il sistema operativo	`provided`	Amazon Linux	8 gennaio 2024	8 febbraio 2024	30 settembre 2026
Ruby 2.7	`ruby2.7`	Amazon Linux 2	7 dicembre 2023	9 gennaio 2024	30 settembre 2026
Node.js 14	`nodejs14.x`	Amazon Linux 2	4 dicembre 2023	9 gennaio 2024	30 settembre 2026
Python 3.7	`python3.7`	Amazon Linux	4 dicembre 2023	9 gennaio 2024	30 settembre 2026

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
.NET Core 3.1	`dotnetcore3.1`	Amazon Linux 2	3 aprile 2023	3 aprile 2023	3 maggio 2023
Node.js 12	`nodejs12.x`	Amazon Linux 2	31 marzo 2023	31 marzo 2023	30 aprile 2023
Python 3.6	`python3.6`	Amazon Linux	18 luglio 2022	18 luglio 2022	29 agosto 2022
.NET 5 (solo container)	`dotnet5.0`	Amazon Linux 2	10 maggio 2022	N/D	N/D
.NET Core 2.1	`dotnetcore2.1`	Amazon Linux	5 gennaio 2022	5 gennaio 2022	13 aprile 2022
Node.js 10	`nodejs10.x`	Amazon Linux 2	30 luglio 2021	30 luglio 2021	14 febbraio 2022
Ruby 2.5	`ruby2.5`	Amazon Linux	30 luglio 2021	30 luglio 2021	31 marzo 2022
Python 2.7	`python2.7`	Amazon Linux	15 luglio 2021	15 luglio 2021	30 maggio 2022
Node.js 8.10	`nodejs8.10`	Amazon Linux	6 marzo 2020	4 febbraio 2020	6 marzo 2020
Node.js 4.3	`nodejs4.3`	Amazon Linux	5 marzo 2020	3 febbraio 2020	5 marzo 2020
Node.js 4.3 edge	`nodejs4.3-edge`	Amazon Linux	5 marzo 2020	31 marzo 2019	30 aprile 2019
Node.js 6.10	`nodejs6.10`	Amazon Linux	12 agosto 2019	12 luglio 2019	12 agosto 2019

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
.NET Core 1.0	`dotnetcore1.0`	Amazon Linux	27 giugno 2019	30 giugno 2019	30 luglio 2019
.NET Core 2.0	`dotnetcore2.0`	Amazon Linux	30 maggio 2019	30 aprile 2019	30 maggio 2019
Node.js 0.10	`nodejs`	Amazon Linux	30 agosto 2016	30 settembre 2016	31 ottobre 2016

Nella quasi totalità dei casi, la end-of-life data della versione linguistica o del sistema operativo è nota con largo anticipo. I seguenti collegamenti forniscono le end-of-life pianificazioni per ogni lingua supportata da Lambda come runtime gestito.

Policy di supporto su linguaggio e framework

- Node.js – github.com
- Python – devguide.python.org
- Ruby – www.ruby-lang.org
- Java — www.oracle.com e Corretto FAQs
- Go – golang.org
- .NET – dotnet.microsoft.com

Informazioni su come Lambda gestisce gli aggiornamenti delle versioni di runtime

Lambda mantiene aggiornato ogni runtime gestito con aggiornamenti di sicurezza, correzioni di bug, nuove funzionalità, miglioramenti delle prestazioni e supporto per versioni minori. Questi aggiornamenti di runtime vengono pubblicati come versioni di runtime. Lambda applica gli aggiornamenti di runtime alle funzioni migrando la funzione da una versione di runtime precedente a una nuova versione di runtime.

Per impostazione predefinita, Lambda applica automaticamente gli aggiornamenti di runtime alle funzioni che utilizzano runtime gestiti. Con gli aggiornamenti automatici del runtime, Lambda si assume l'onere operativo dell'applicazione di patch alle versioni di runtime. Per la maggior parte dei clienti, gli aggiornamenti automatici sono la scelta ideale. È possibile modificare questo comportamento predefinito configurando le impostazioni di gestione del runtime.

Lambda pubblica inoltre ogni nuova versione di runtime come immagine di container. Per aggiornare le versioni di runtime per le funzioni basate su container, è necessario creare una nuova immagine di container dall'immagine di base aggiornata e implementare nuovamente la funzione.

Ogni versione di runtime è associata a un numero di versione e a un nome della risorsa Amazon (ARN). I numeri di versione di runtime utilizzano uno schema numerico definito da Lambda indipendente dai numeri di versione utilizzati dal linguaggio di programmazione. I numeri di versione di runtime non sono sempre sequenziali. Ad esempio, la versione 42 potrebbe essere seguita dalla versione 45. L'ARN della versione di runtime è un identificatore univoco per ogni versione di runtime. È possibile visualizzare l'ARN della versione di runtime corrente della funzione nella console Lambda o nella riga `INIT_START` dei log della funzione.

Le versioni di runtime non devono essere confuse con gli identificatori di runtime. Ogni runtime ha un identificatore di runtime univoco, ad esempio `python3.14` o `nodejs24.x`. Questi corrispondono a ciascuna delle principali versioni del linguaggio di programmazione. Le versioni di runtime descrivono la versione patch di un singolo runtime.

> **Note**
>
> L'ARN per lo stesso numero di versione di runtime può variare tra le architetture Regioni AWS della CPU.

Argomenti

- Compatibilità con le versioni precedenti
- Modalità di aggiornamento del runtime
- Rollout della versione runtime in due fasi
- Configurazione delle impostazioni di gestione del runtime Lambda
- Rollback di una versione di runtime Lambda
- Identificazione delle modifiche alla versione di runtime di Lambda
- Informazioni sul modello di responsabilità condivisa per la gestione del runtime Lambda
- Controllo delle autorizzazioni di aggiornamento del runtime Lambda per applicazioni ad alta conformità

Compatibilità con le versioni precedenti

Lambda fornisce aggiornamenti di runtime compatibili con le versioni precedenti delle funzioni esistenti. Tuttavia, come nel caso delle patch software, ci sono rari casi in cui un aggiornamento del runtime può influire negativamente su una funzione esistente. Ad esempio, le patch di sicurezza possono evidenziare un problema di fondo di una funzione esistente che dipende dal comportamento precedente e non sicuro.

Quando si crea e si distribuisce una funzione, è importante capire come gestire le dipendenze per evitare potenziali incompatibilità con un futuro aggiornamento di runtime. Ad esempio, supponiamo che la funzione dipenda dal pacchetto A, che a sua volta dipende dal pacchetto B. Entrambi i pacchetti sono inclusi nel runtime Lambda (ad esempio, potrebbero far parte dell'SDK o delle sue dipendenze o parti delle librerie del sistema di runtime).

Considerare i seguenti scenari:

Implementazione	Compatibile con le patch	Motivo
• Package A: utilizzo dal runtime • Package B: utilizzo dal runtime	Sì	I futuri aggiornamenti di runtime ai pacchetti A e B sono compatibili con le versioni precedenti.
• Package A: nel pacchetto di distribuzione	Sì	La tua distribuzione ha la precedenza, quindi i futuri

Implementazione	Compatibile con le patch	Motivo
• Package B: nel pacchetto di distribuzione		aggiornamenti di runtime ai pacchetti A e B non hanno alcun effetto.
• Package A: nel pacchetto di distribuzione • Package B: utilizzo dal runtime	Sì*	I futuri aggiornamenti di runtime al pacchetto B sono compatibili con le versioni precedenti. *Se A e B sono strettamente accoppiati, possono verificar si problemi di compatibilità. Ad esempio, i `botocore` pacchetti `boto3` and nell' AWS SDK per Python devono essere distribuiti insieme.
• Package A: utilizzo dal runtime • Package B: nel pacchetto di distribuzione	No	I futuri aggiornamenti di runtime del pacchetto A potrebbero richiedere una versione aggiornata del pacchetto B. Tuttavia, la versione distribuita del pacchetto B ha la precedenz a e potrebbe non essere compatibile con la versione aggiornata del pacchetto A.

Per mantenere la compatibilità con i futuri aggiornamenti di runtime, segui queste best practice:

- Quando possibile, impacchetta tutte le dipendenze: includi tutte le librerie richieste, incluso l' AWS SDK e le sue dipendenze, nel pacchetto di distribuzione. Ciò garantisce un set di componenti stabile e compatibile.

- Usa SDKs con moderazione i pacchetti forniti in fase di esecuzione: affidati all'SDK fornito in fase di esecuzione solo quando non puoi includere pacchetti aggiuntivi (ad esempio, quando usi l'editor di codice della console Lambda o il codice in linea in un modello). AWS CloudFormation
- Evita di sovrascrivere le librerie di sistema: non distribuire librerie di sistemi operativi personalizzate che potrebbero entrare in conflitto con i futuri aggiornamenti di runtime.

Modalità di aggiornamento del runtime

Lambda fornisce aggiornamenti di runtime compatibili con le versioni precedenti delle funzioni esistenti. Tuttavia, come nel caso delle patch software, ci sono rari casi in cui un aggiornamento del runtime può influire negativamente su una funzione esistente. Ad esempio, le patch di sicurezza possono evidenziare un problema di fondo di una funzione esistente che dipende dal comportamento precedente e non sicuro. I controlli di gestione del runtime Lambda aiutano a ridurre il rischio di impatto sui carichi di lavoro nel raro caso di incompatibilità delle versioni di runtime. Per ogni versione della funzione ($LATESTo versione pubblicata), è possibile scegliere una delle seguenti modalità di aggiornamento del runtime:

- Auto (default) (Automatico [impostazione predefinita]): esegue automaticamente l'aggiornamento alla versione di runtime più recente e sicura tramite una Rollout della versione runtime in due fasi. Consigliamo questa modalità alla maggior parte dei clienti in modo da beneficiare sempre degli aggiornamenti di runtime.
- Aggiornamento delle funzioni: aggiorna alla versione di runtime più recente e sicura quando aggiorni la funzione. Quando aggiorni la funzione, Lambda aggiorna il runtime della funzione alla versione più recente e sicura. Questo approccio sincronizza gli aggiornamenti del runtime con le implementazioni delle funzioni, dando all'utente il controllo su quando Lambda applica gli aggiornamenti del runtime. Con questa modalità, è possibile rilevare e mitigare tempestivamente le rare incompatibilità degli aggiornamenti del runtime. Quando si utilizza questa modalità, è necessario aggiornare regolarmente le funzioni in modo da mantenerne aggiornato il runtime.
- Manuale: aggiorna manualmente la versione del runtime. Specifica una versione di runtime nella configurazione della funzione. La funzione utilizzerà questa versione di runtime a tempo indeterminato. Nel raro caso in cui una nuova versione di runtime non sia compatibile con una funzione esistente, ciò consente di ripristinare la funzione a una versione di runtime precedente. Si consiglia di non utilizzare la modalità Manual (Manuale) per cercare di ottenere la coerenza del runtime tra le varie implementazioni. Per ulteriori informazioni, consulta Rollback di una versione di runtime Lambda.

La responsabilità dell'applicazione degli aggiornamenti di runtime alle funzioni varia in base alla modalità di aggiornamento del runtime scelta. Per ulteriori informazioni, consulta Informazioni sul modello di responsabilità condivisa per la gestione del runtime Lambda.

Rollout della versione runtime in due fasi

Lambda introduce nuove versioni di runtime nel seguente ordine:

1. Nella prima fase Lambda utilizza la nuova versione di runtime ogni volta che crei o aggiorni una funzione. Una funzione viene aggiornata quando si chiamano le operazioni UpdateFunctionCodeo l'UpdateFunctionConfigurationAPI.
2. Nella seconda fase, Lambda aggiorna qualsiasi funzione che utilizza la modalità di aggiornamento del runtime Auto e che non è già stata aggiornata alla nuova versione di runtime.

La durata complessiva del processo di rollout varia in base a diversi fattori, tra cui la gravità di eventuali patch di sicurezza incluse nell'aggiornamento del runtime.

Se si stanno sviluppando e implementando attivamente le funzioni, è molto probabile che vengano acquistate nuove versioni di runtime durante la prima fase. Ciò sincronizza gli aggiornamenti del runtime con gli aggiornamenti delle funzioni. Nel raro caso in cui l'ultima versione di runtime abbia un impatto negativo sull'applicazione, questo approccio ti permette di intraprendere azioni correttive in maniera tempestiva. Le funzioni che non sono in fase di sviluppo attivo ricevono comunque i vantaggi operativi degli aggiornamenti automatici del runtime durante la seconda fase.

Questo approccio non influisce sulle funzioni impostate sulla modalità Function update (Aggiornamento della funzione) o Manual (Manuale). Le funzioni che utilizzano la modalità Function update (Aggiornamento della funzione) ricevono gli aggiornamenti di runtime più recenti solo quando vengono create o aggiornate. Le funzioni che utilizzano la modalità Manual (Manuale) non ricevono aggiornamenti di runtime.

Lambda pubblica nuove versioni di runtime in modo graduale e continuo nelle Regioni AWS. Se le funzioni sono impostate sulle modalità Auto (Aggiornamento automatico) o Function update (Aggiornamento della funzione), è possibile che le funzioni implementate nello stesso momento in regioni diverse o in momenti diversi nella stessa regione abbiano versioni di runtime diverse. I clienti che richiedono una coerenza delle versioni di runtime garantita nei propri ambienti devono utilizzare le immagini di container per implementare le proprie funzioni Lambda. La modalità Manuale è concepita come mitigazione temporanea per consentire il rollback della versione di runtime nel raro caso in cui una versione di runtime non sia compatibile con la tua funzione.

Configurazione delle impostazioni di gestione del runtime Lambda

È possibile configurare le impostazioni di gestione del runtime utilizzando la console Lambda o AWS Command Line Interface (AWS CLI).

> **Note**
>
> È possibile configurare le impostazioni di gestione del runtime separatamente per ciascuna versione della funzione.

Configurazione del modo in cui Lambda aggiorna la versione di runtime (console)

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Nella scheda Code (Codice), in Runtime settings (Impostazioni di runtime), scegli Edit runtime management configuration (Modifica configurazione di gestione del runtime).
4. In Runtime management configuration (Configurazione di gestione del runtime), scegli una delle seguenti opzioni:

   - Per aggiornare automaticamente la funzione all'ultima versione di runtime, scegli Auto (Automatico).
   - Per fare in modo che la funzione venga aggiornata all'ultima versione di runtime quando si modifica la funzione, scegli Function update (Aggiornamento della funzione).
   - Perché la funzione venga aggiornata all'ultima versione di runtime solo quando si modifica l'ARN della versione di runtime, scegli Manual (Manuale). L'ARN della versione di runtime è disponibile in Configurazione della gestione del runtime. L'ARN è disponibile anche nella riga `INIT_START` dei log della funzione.

   Per ulteriori informazioni su queste opzioni, consulta Modalità di aggiornamento del runtime.
5. Selezionare Salva.

Configurazione del modo in cui Lambda aggiorna la versione di runtime (AWS CLI)

Per configurare la gestione del runtime per una funzione, esegui il comando put-runtime-management-config della AWS CLI. Quando si utilizza la modalità `Manual`, è necessario fornire anche l'ARN della versione di 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
```

Verrà visualizzato un output simile al seguente:

```
{
  "UpdateRuntimeOn": "Manual",
  "FunctionArn": "arn:aws:lambda:us-east-2:111122223333:function:my-function",
  "RuntimeVersionArn": "arn:aws:lambda:us-
east-2::runtime:8eeff65f6809a3ce81507fe733fe09b835899b99481ba22fd75b5a7338290ec1"
}
```

Rollback di una versione di runtime Lambda

Nel raro caso in cui una nuova versione di runtime non sia compatibile con una funzione esistente, è possibile ripristinare la versione di runtime a una versione precedente. Ciò mantiene l'applicazione funzionante e riduce al minimo le interruzioni, dando il tempo necessario per porre rimedio all'incompatibilità prima di tornare alla versione di runtime più recente.

Lambda non impone un limite per il tempo che è possibile utilizzare una particolare versione di runtime. Tuttavia, consigliamo vivamente di eseguire sempre l'aggiornamento all'ultima versione di runtime il prima possibile per beneficiare delle patch di sicurezza, dei miglioramenti delle prestazioni e delle funzionalità più recenti. Lambda offre la possibilità di ripristinare una versione di runtime precedente solo come mitigazione temporanea nel raro caso in cui si verifichi un problema di compatibilità con gli aggiornamenti del runtime. Le funzioni che utilizzano una versione di runtime precedente per un periodo prolungato possono alla fine presentare un peggioramento delle prestazioni o problemi, come la scadenza del certificato, che possono impedirne il corretto funzionamento.

Puoi eseguire il rollback di una versione di runtime nei seguenti modi:

- Utilizzo della modalità manuale per l'aggiornamento del runtime
- Utilizzo delle versioni pubblicate della funzione

Per ulteriori informazioni, consulta Introduzione ai controlli di gestione del runtime AWS Lambda sul blog di AWS Compute.

Rollback di una versione di runtime tramite la modalità di aggiornamento del runtime Manual (Manuale)

Se utilizzi la modalità di aggiornamento della versione di runtime Auto (Automatico) o stai utilizzando la versione di runtime `$LATEST`, puoi ripristinare la versione di runtime precedente utilizzando la modalità Manual (Manuale). Per la versione della funzione che desideri ripristinare, modifica la modalità di aggiornamento della versione di runtime in Manual (Manuale) e specifica l'ARN della versione di runtime precedente. Per ulteriori informazioni su come trovare l'ARN della versione di runtime precedente, consulta Identificazione delle modifiche alla versione di runtime di Lambda.

> **Note**
>
> Se la versione `$LATEST` della funzione è configurata per utilizzare la modalità Manual (Manuale), non è possibile modificare l'architettura della CPU o la versione di runtime utilizzata dalla funzione. Per apportare queste modifiche, è necessario passare alla modalità di aggiornamento Auto (Automatico) o Function update (Aggiornamento della funzione).

Rollback di una versione di runtime utilizzando le versioni pubblicate della funzione

Le versioni delle funzioni pubblicate sono un'istantanea immutabile del codice e della configurazione della funzione `$LATEST` al momento della loro creazione. In modalità Auto (Automatico), Lambda aggiorna automaticamente la versione di runtime delle versioni delle funzioni pubblicate durante la seconda fase del rollout della versione di runtime. In modalità Function update (Aggiornamento della funzione), Lambda non aggiorna la versione di runtime delle versioni delle funzioni pubblicate.

Le versioni delle funzioni pubblicate che utilizzano la modalità Function update (Aggiornamento della funzione) creano quindi un'istantanea statica del codice della funzione, della configurazione e della versione di runtime. Utilizzando la modalità Function update (Aggiornamento della funzione) con le versioni delle funzioni, è possibile sincronizzare gli aggiornamenti di runtime con le varie implementazioni. È possibile coordinare il rollback del codice, della configurazione e delle versioni di runtime anche reindirizzando il traffico verso una versione della funzione pubblicata in precedenza. Questo approccio può essere integrato nell'integrazione continua e distribuzione continua (CI/CD) per un rollback completamente automatico nel raro caso di incompatibilità degli aggiornamenti di runtime. Quando si utilizza questo approccio, è necessario aggiornare regolarmente la funzione e pubblicare

nuove versioni delle funzioni per avere gli ultimi aggiornamenti di runtime. Per ulteriori informazioni, consulta Informazioni sul modello di responsabilità condivisa per la gestione del runtime Lambda.

Identificazione delle modifiche alla versione di runtime di Lambda

Il numero di versione di runtime e l'ARN vengono registrati nella riga del log `INIT_START`, che Lambda invia a CloudWatch Logs ogni volta che crea un nuovo ambiente di esecuzione. Poiché l'ambiente di esecuzione utilizza la stessa versione di runtime per tutte le chiamate di funzione, Lambda emette la riga di log `INIT_START` solo quando esegue la fase di inizializzazione. Non emette questa riga di log per ogni chiamata di funzione. Lambda invia la riga di log a CloudWatch Logs, ma non è visibile nella console.

> **Note**
>
> I numeri di versione di runtime non sono sempre sequenziali. Ad esempio, la versione 42 potrebbe essere seguita dalla versione 45.

Example Riga di log INIT_START di esempio

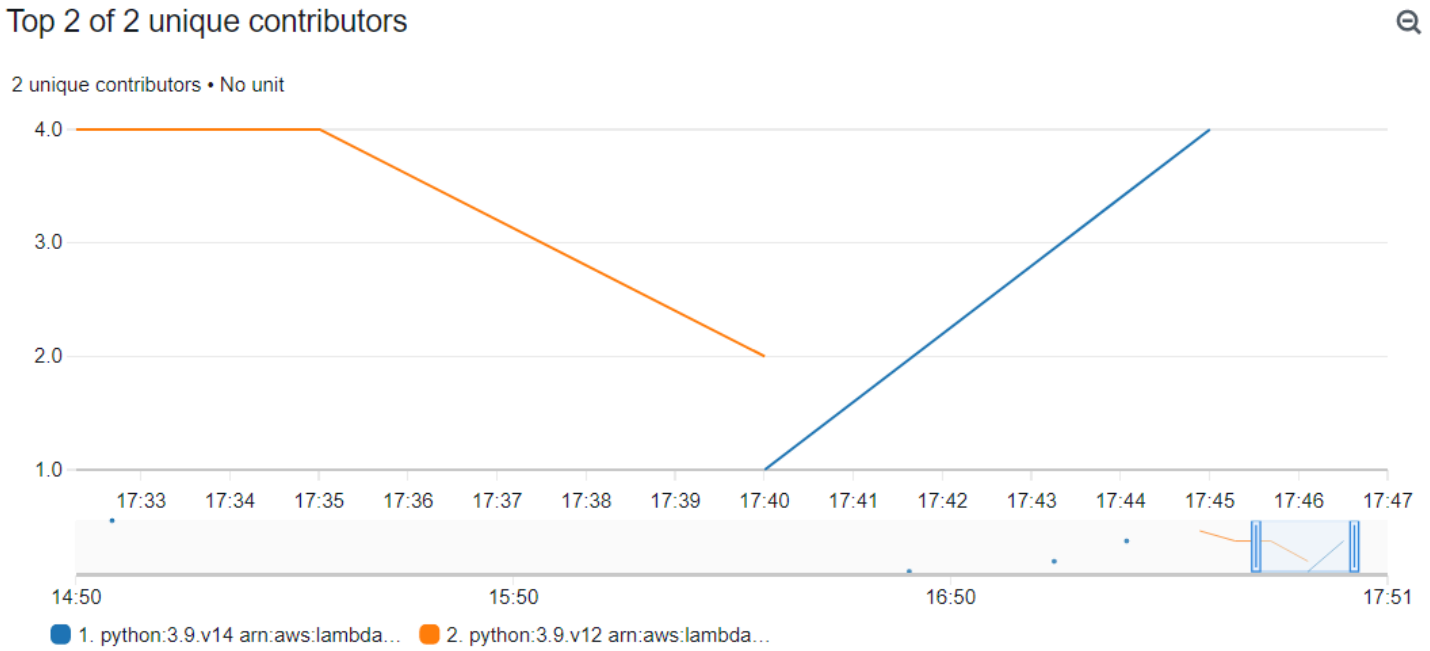
```
INIT_START Runtime Version: python:3.13.v14    Runtime Version ARN: arn:aws:lambda:eu-south-1::runtime:7b620fc2e66107a1046b140b9d320295811af3ad5d4c6a011fad1fa65127e9e6I
```

Anziché lavorare direttamente con i log, è possibile utilizzare Approfondimenti sulle contribuzioni Amazon CloudWatch per identificare le transizioni tra le versioni di runtime. La regola seguente conta le versioni di runtime distinte di ciascuna riga di log `INIT_START`. Per utilizzare la regola, sostituisci il nome del gruppo di log di esempio `/aws/lambda/*` con il prefisso appropriato per la funzione o il gruppo di funzioni.

```
{
  "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"
  }
}
```

Il seguente report di Approfondimenti sulle contribuzioni CloudWatch mostra un esempio di transizione della versione di runtime acquisita dalla regola precedente. La riga arancione mostra l'inizializzazione dell'ambiente di esecuzione per la versione di runtime precedente (python:3.13.v12) mentre la linea blu mostra l'inizializzazione dell'ambiente di esecuzione per la nuova versione di runtime (python:3.13.v14).



Informazioni sul modello di responsabilità condivisa per la gestione del runtime Lambda

Lambda è responsabile della cura e della pubblicazione degli aggiornamenti di sicurezza per tutti i runtime gestiti e le immagini dei container supportati. La responsabilità di aggiornare le funzioni esistenti per utilizzare la versione di runtime più recente varia a seconda della modalità di aggiornamento del runtime utilizzata.

Lambda è responsabile dell'applicazione degli aggiornamenti di runtime a tutte le funzioni configurate per l'uso la modalità di aggiornamento del runtime Auto (Automatico).

Per le funzioni configurate con la modalità di aggiornamento del runtime Function update (Aggiornamento della funzione), l'utente è responsabile dell'aggiornamento regolare della funzione. Lambda è responsabile dell'applicazione degli aggiornamenti di runtime quando si effettuano tali aggiornamenti. Se non aggiorni la funzione, Lambda non aggiorna il runtime. Se non aggiorni regolarmente la tua funzione, ti consigliamo vivamente di configurarla per gli aggiornamenti automatici del runtime in modo da continuare a ricevere aggiornamenti di sicurezza.

Per le funzioni configurate per utilizzare la modalità di aggiornamento del runtime Manual (Manuale), l'utente è responsabile dell'aggiornamento della funzione per l'uso della versione di runtime più recente. Si consiglia vivamente di utilizzare questa modalità solo per ripristinare la versione di runtime come mitigazione temporanea nel raro caso di incompatibilità degli aggiornamenti del runtime. Ti consigliamo inoltre di passare alla modalità Auto (Automatico) il più rapidamente possibile per ridurre al minimo il tempo in cui le tue funzioni non vengono aggiornate.

Se per implementare le funzioni, vengono utilizzate le immagini di container, Lambda è responsabile della pubblicazione delle immagini di base aggiornate. In questo caso, l'utente sarà responsabile della ricostruzione dell'immagine di container della funzione dall'immagine di base più recente e della nuova implementazione dell'immagine di container.

Ciò è riassunto nella seguente tabella:

Deployment mode (Modalità distribuzione)	Responsabilità di Lambda	Responsabilità del cliente
Runtime gestito, modalità Auto (Automatico)	Pubblica nuove versioni di runtime contenenti le patch più recenti. Applica le patch di runtime alle funzioni esistenti.	Torna a una versione di runtime precedente nel raro caso in cui si verifichi un problema di compatibilità con l'aggiornamento del runtime. Segui le migliori pratiche per la compatibilità con le versioni precedenti.
Runtime gestito, modalità Function update (Aggiornamento della funzione)	Pubblica nuove versioni di runtime contenenti le patch più recenti.	Aggiorna regolarmente le funzioni per utilizzare la versione di runtime più recente. Passa una funzione alla modalità Auto (Automatico) quando questa non viene aggiornata regolarmente. Torna a una versione di runtime precedente nel raro caso in cui si verifichi un problema di compatibilità con l'aggiornamento del runtime. Segui le migliori pratiche per la compatibilità con le versioni precedenti.
Runtime gestito, modalità Manual (Manuale)	Pubblica nuove versioni di runtime contenenti le patch più recenti.	Utilizza questa modalità solo per il rollback temporaneo del runtime nel raro caso in cui si verifichi un problema di compatibilità con gli aggiornamenti del runtime. Passa le funzioni alla modalità di aggiornamento Auto (Automatico) o Function update (Aggiornamento della funzione) e alla versione di runtime più recente il più presto possibile.
Immagine di container	Pubblica nuove immagini di container contenenti le patch più recenti.	Implementa nuovamente le funzioni in maniera regolare utilizzando l'immagine di base del

Deployment mode (Modalità distribuzione)	Responsabilità di Lambda	Responsabilità del cliente
		container più recente per utilizzare le patch più recenti.

Per ulteriori informazioni sulla responsabilità condivisa con AWS, consulta Modello di responsabilità condivisa.

Controllo delle autorizzazioni di aggiornamento del runtime Lambda per applicazioni ad alta conformità

Per soddisfare i requisiti di patch, i clienti Lambda in genere si affidano agli aggiornamenti automatici del runtime. Se la tua applicazione è soggetta a severi requisiti di aggiornamento delle patch, potresti voler limitare l'uso delle versioni di runtime precedenti. È possibile limitare i controlli di gestione del runtime di Lambda utilizzando AWS Identity and Access Management (IAM) per negare agli utenti del proprio account AWS l'accesso all'operazione API PutRuntimeManagementConfig. Questa operazione viene utilizzata per scegliere la modalità di aggiornamento del runtime per una funzione. Se si nega l'accesso a questa operazione, tutte le funzioni passeranno automaticamente alla modalità Auto (Automatico). È possibile applicare questa limitazione in tutta l'organizzazione utilizzando una policy di controllo dei servizi. Nel caso in cui sia necessario ripristinare una funzione a una versione di runtime precedente, è possibile concedere un'eccezione alla policy caso per caso.

Recuperare dati sulle funzioni Lambda che utilizzano un runtime obsoleto

Quando un runtime Lambda sta per diventare obsoleto, Lambda ti avvisa tramite e-mail e fornisce notifiche al termine. Health Dashboard Trusted Advisor Queste e-mail e notifiche elencano le versioni $LATEST delle funzioni che utilizzano il runtime. Per elencare tutte le versioni delle funzioni che utilizzano un determinato runtime, puoi usare il () o uno dei AWS Command Line Interface .AWS CLI AWS SDKs

Se disponi di un gran numero di funzioni che utilizzano un runtime destinato a diventare obsoleto, puoi anche usare AWS CLI or AWS SDKs per aiutarti a dare priorità agli aggiornamenti delle funzioni richiamate più comunemente.

Fate riferimento alle seguenti sezioni per imparare a utilizzare AWS CLI e AWS SDKs raccogliere dati sulle funzioni che utilizzano un particolare runtime.

Elenco delle versioni delle funzioni che utilizzano un determinato runtime

Per utilizzare AWS CLI per elencare tutte le versioni delle funzioni che utilizzano un determinato runtime, esegui il comando seguente. Sostituiscilo `RUNTIME_IDENTIFIER` con il nome del runtime che è diventato obsoleto e scegli il tuo. Regione AWS Per elencare solo le versioni della funzione $LATEST, ometti `--function-version ALL` dal comando.

```
aws lambda list-functions --function-version ALL --region us-east-1 --output text --
query "Functions[?Runtime=='RUNTIME_IDENTIFIER'].FunctionArn"
```

> **Tip**
>
> Il comando di esempio elenca le funzioni nella `us-east-1` regione per una determinata regione. Account AWS Dovrai ripetere questo comando per ogni regione in cui il tuo account ha funzioni e per ognuna delle tue. Account AWS

Puoi anche elencare le funzioni che utilizzano un particolare runtime utilizzando uno dei AWS SDKs. Il codice di esempio seguente utilizza V3 AWS SDK per JavaScript e the AWS SDK per Python (Boto3) per restituire un elenco della funzione ARNs per le funzioni che utilizzano un determinato runtime. Il codice di esempio restituisce anche il gruppo di CloudWatch log per ciascuna delle funzioni elencate. È possibile utilizzare questo gruppo di log per trovare la data dell'ultima invocazione per

la funzione. Per ulteriori informazioni, consulta la sezione the section called “Identificazione delle funzioni richiamate più di frequente e più di recente” seguente.

Node.js

Example JavaScript codice per elencare le funzioni utilizzando un particolare runtime

```
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 Codice Python per elencare le funzioni utilizzando un particolare runtime

```
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')
```

Per ulteriori informazioni sull'utilizzo di un AWS SDK per elencare le funzioni utilizzando l'ListFunctionsazione, consulta la documentazione SDK relativa al linguaggio di programmazione preferito.

Puoi anche utilizzare la funzionalità Interrogazioni AWS Config avanzate per elencare tutte le funzioni che utilizzano un runtime interessato. Questa query restituisce solo le versioni della funzione $LATEST, ma è possibile aggregare le query per elencare le funzioni in tutte le regioni e più regioni Account AWS con un solo comando. Per ulteriori informazioni, consulta Interrogazione dello stato di configurazione corrente delle AWS Auto Scaling risorse nella Guida per gli sviluppatori.AWS Config

Identificazione delle funzioni richiamate più di frequente e più di recente

Se il tuo Account AWS file contiene funzioni che utilizzano un runtime che sta per essere obsoleto, potresti voler dare priorità all'aggiornamento delle funzioni che vengono richiamate di frequente o delle funzioni che sono state richiamate di recente.

Se disponi solo di poche funzioni, puoi utilizzare la console CloudWatch Logs per raccogliere queste informazioni esaminando i flussi di registro delle tue funzioni. Per ulteriori informazioni, consulta Visualizzare i dati di registro inviati ai CloudWatch registri.

Per vedere il numero di chiamate di funzioni recenti, puoi anche utilizzare le informazioni sulle CloudWatch metriche mostrate nella console Lambda. Per visualizzare queste informazioni, completa le seguenti operazioni:

1. Aprire la pagina Funzioni della console Lambda.
2. Seleziona la funzione per la quale desideri visualizzare le statistiche sull'invocazione.
3. Selezionare la scheda Monitor (Monitora).
4. Imposta il periodo di tempo per cui desideri visualizzare le statistiche utilizzando il selettore dell'intervallo di date. Le invocazioni recenti vengono visualizzate nel riquadro Invocazioni.

Per gli account con un numero maggiore di funzioni, può essere più efficiente raccogliere questi dati a livello di codice utilizzando AWS CLI o una delle azioni using the e API. AWS SDKs DescribeLogStreamsGetMetricStatistics

Gli esempi seguenti forniscono frammenti di codice che utilizzano V3 AWS SDK per JavaScript e AWS SDK per Python (Boto3) per identificare la data dell'ultima chiamata per una particolare funzione e per determinare il numero di chiamate per una particolare funzione negli ultimi 14 giorni.

Node.js

Example JavaScript codice per trovare l'ora dell'ultima chiamata per una funzione

```
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 Codice Python per trovare l'ora dell'ultima invocazione per una funzione

```
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**
>
> Puoi trovare il nome del gruppo di log della tua funzione utilizzando l'operazione ListFunctionsAPI. Per un esempio su come eseguire questa operazione, consulta the section called “Elenco delle versioni delle funzioni che utilizzano un determinato runtime”.

Node.js

Example JavaScript codice per trovare il numero di chiamate negli ultimi 14 giorni

```
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 Codice Python per trovare il numero di invocazioni negli ultimi 14 giorni

```
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}')
```

Modifica dell'ambiente di runtime

È possibile utilizzare le estensioni interne per modificare il processo di runtime. Le estensioni interne non sono processi separati, sono eseguiti nell'ambito del processo di runtime.

Lambda fornisce variabili di ambiente specifiche della lingua che è possibile impostare per aggiungere opzioni e strumenti al runtime. Lambda fornisce anche Script wrapper, che consentono a Lambda di delegare l'avvio del runtime allo script. È possibile creare uno script wrapper per personalizzare il comportamento di avvio del runtime.

Variabili di ambiente specifiche della lingua

Lambda supporta metodi di sola configurazione per abilitare il precaricamento del codice durante l'inizializzazione della funzione tramite le seguenti variabili di ambiente specifiche della lingua:

- `JAVA_TOOL_OPTIONS`: su Java, Lambda supporta questa variabile di ambiente per impostare ulteriori variabili della riga di comando in Lambda. Questa variabile di ambiente consente di specificare l'inizializzazione degli strumenti, in particolare l'avvio di agenti del linguaggio di programmazione nativo o Java utilizzando le opzioni `agentlib` o `javaagent`. Per ulteriori informazioni, consulta la sezione Variabili di ambiente `JAVA_TOOL_OPTIONS`.
- `NODE_OPTIONS`: disponibile nei runtime di Node.js.
- `DOTNET_STARTUP_HOOKS` – Su.NET Core 3.1 e versioni successive, questa variabile di ambiente specifica un percorso ad un assembly (dll) che Lambda può utilizzare.

L'utilizzo di variabili di ambiente specifiche della lingua è il modo preferito per impostare le proprietà di avvio.

Script wrapper

È possibile creare uno script wrapper per personalizzare il comportamento di avvio runtime della funzione Lambda. Uno script wrapper consente di impostare parametri di configurazione che non possono essere impostati tramite variabili di ambiente specifiche della lingua.

> **Note**
>
> Le chiamate potrebbero non riuscire se lo script wrapper non avvia correttamente il processo di runtime.

Gli script wrapper sono supportati su tutti i runtime Lambda nativi. Gli script wrapper non sono supportati su Runtime solo per il sistema operativo (la famiglia di runtime `provided`).

Quando si utilizza uno script wrapper per la funzione, Lambda avvia il runtime utilizzando lo script. Lambda invia allo script il percorso all'interprete e tutti gli argomenti originali per l'avvio del runtime standard. Lo script può estendere o trasformare il comportamento di avvio del programma. Ad esempio, lo script può iniettare e modificare argomenti, impostare variabili di ambiente o acquisire metriche, errori e altre informazioni diagnostiche.

È possibile specificare lo script impostando il valore della variabile di ambiente `AWS_LAMBDA_EXEC_WRAPPER` come percorso del file system di un file binario o di uno script eseguibile.

Esempio: creare e utilizzare uno script wrapper come layer Lambda

Nell'esempio seguente, si crea uno script wrapper per avviare l'interprete Python con l'opzione `-X importtime`. Quando si esegue la funzione, Lambda genera una voce di log per mostrare la durata del tempo di importazione per ogni importazione.

Per creare e utilizzare uno script wrapper come livello

1. Crea una directory per il livello:

   ```
   mkdir -p python-wrapper-layer/bin
   cd python-wrapper-layer/bin
   ```

2. Nella `bin` directory, incollate il codice seguente in un nuovo file denominato`importtime_wrapper`. Questo è lo script wrapper.

   ```
   #!/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. Assegnare autorizzazioni eseguibili a tutti gli script.

```
chmod +x importtime_wrapper
```

4. Crea un file .zip per il livello:

```
cd ..
zip -r ../python-wrapper-layer.zip .
```

5. Verifica che il file .zip abbia la seguente struttura di directory:

```
python-wrapper-layer.zip
# bin
    # importtime_wrapper
```

6. Create un livello utilizzando il pacchetto.zip.
7. Crea una funzione utilizzando la console Lambda.
   a. Aprire la console Lambda.
   b. Scegli Crea funzione.
   c. Immetti un nome per la funzione.
   d. Per Runtime, scegli l'ultimo runtime supportato di Python.
   e. Scegli Crea funzione.
8. Aggiungi il livello alla tua funzione.
   a. Scegliere la funzione, quindi scegliere Codice se non è già selezionato.
   b. Scorri verso il basso fino alla sezione Livelli, quindi scegli Aggiungi un livello.
   c. Per Layer source, seleziona Livelli personalizzati, quindi scegli il tuo livello dall'elenco a discesa Livelli personalizzati.
   d. In Version (Versione), selezionare 1.
   e. Scegli Add (Aggiungi).
9. Aggiungi la variabile di ambiente wrapper.
   a. Scegli Configurazione, quindi scegli Variabili di ambiente.
   b. In Environment variables (Variabili di ambiente), scegliere Edit (Modifica).

c. Scegli Add environment variable (Aggiungi variabile d'ambiente).

d. In Chiave, inserire `AWS_LAMBDA_EXEC_WRAPPER`.

e. Per Value, inserisci `/opt/bin/importtime_wrapper` (`/opt/` + la struttura delle cartelle del tuo livello.zip).

f. Selezionare Salva.

10. Provate lo script wrapper.

a. Seleziona la scheda Test.

b. In Evento di prova, scegli Test. Non è necessario creare un evento di test: l'evento predefinito funzionerà.

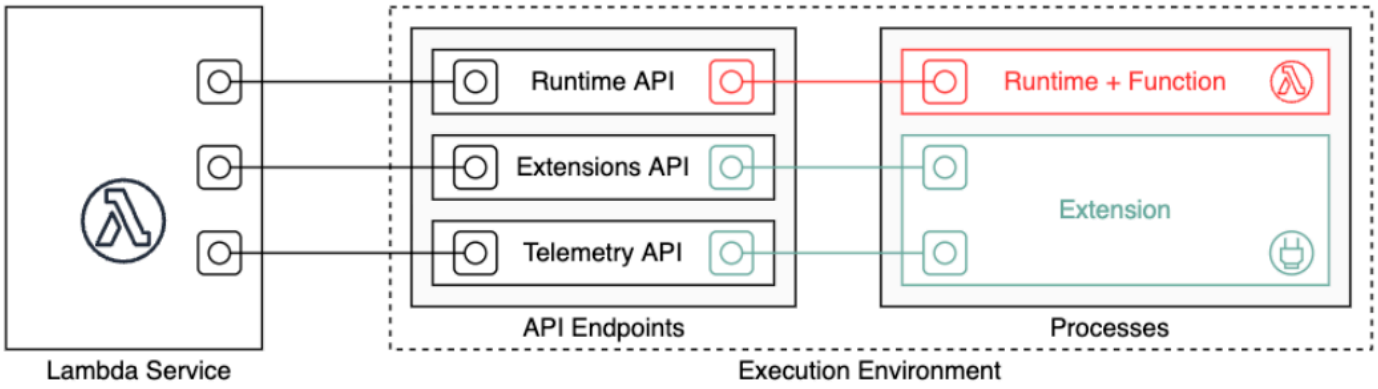
c. Scorri verso il basso fino a Log output. Poiché lo script wrapper ha avviato l'interprete Python con l'opzione `-X importtime`, i log mostrano il tempo richiesto per ogni importazione. Ad esempio:

```
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
```

Utilizzo dell'API di runtime Lambda per runtime personalizzati

AWS Lambda fornisce un'API HTTP per runtime personalizzati per ricevere eventi di chiamata da Lambda e inviare i dati di risposta all'interno dell'ambiente di esecuzione Lambda. In questa sezione è riportato il riferimento API per l'API di runtime Lambda.

> **Le istanze gestite Lambda supportano richieste simultanee**
>
> Le istanze gestite Lambda utilizzano la stessa API di runtime delle funzioni Lambda (predefinite). La differenza fondamentale è che le istanze gestite possono accettare `/response` richieste simultanee `/next` e fino al limite configurato. `AWS_LAMBDA_MAX_CONCURRENCY` Ciò consente l'elaborazione simultanea di più chiamate all'interno di un unico ambiente di esecuzione. Per ulteriori informazioni sulle istanze gestite, consulta. Comprendere l'ambiente di esecuzione di Lambda Managed Instances



La specifica OpenAPI per l'API runtime versione 2018-06-01 è disponibile qui: runtime-api.zip

Per creare un URL di richiesta API, i runtime ottengono l'endpoint API dalla variabile d'ambiente `AWS_LAMBDA_RUNTIME_API`, aggiungere la versione dell'API e aggiungere il percorso della risorsa desiderato.

Example Richiesta

```
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next"
```

Metodi API

- Chiamata successiva
- Risposta all'invocazione
- Errore di inizializzazione
- Errore della chiamata

Chiamata successiva

Percorso – `/runtime/invocation/next`

Metodo – GET

Il runtime invia questo messaggio a Lambda per richiedere un evento di chiamata. Il corpo della risposta contiene il payload della chiamata che è un documento JSON contenente i dati dell'evento dal trigger della funzione. Le intestazioni della risposta contengono dati aggiuntivi sulla chiamata.

Intestazioni di risposta

- `Lambda-Runtime-Aws-Request-Id` – L'ID della richiesta che identifica la richiesta che ha attivato la chiamata della funzione.

  Ad esempio, `8476a536-e9f4-11e8-9739-2dfe598c3fcd`.
- `Lambda-Runtime-Deadline-Ms` – La data del timeout della funzione in millisecondi Unix.

  Ad esempio, `1542409706888`.
- `Lambda-Runtime-Invoked-Function-Arn` – L'ARN della funzione Lambda, la versione o l'alias specificato nella chiamata.

  Ad esempio, `arn:aws:lambda:us-east-2:123456789012:function:custom-runtime`.
- `Lambda-Runtime-Trace-Id` – L'intestazione di tracciamento AWS X-Ray.

  Ad esempio, `Root=1-5bef4de7-ad49b0e87f6ef6c87fc2e700;Parent=9a9197af755a6419;Sampled=1`.
- `Lambda-Runtime-Client-Context`— Per le chiamate da AWS Mobile SDK, i dati sull'applicazione client e sul dispositivo.
- `Lambda-Runtime-Cognito-Identity`— Per le chiamate da AWS Mobile SDK, dati sul provider di identità Amazon Cognito.

Non impostare un timeout sulla richiesta GET in quanto la risposta potrebbe essere ritardata. Nell'intervallo di tempo che va dal bootstrap del runtime di Lambda al momento in cui il runtime dispone di un evento da restituire, il processo di runtime potrebbe rimanere bloccato per alcuni secondi.

L'ID della richiesta tiene traccia della chiamata in Lambda. Utilizzalo per specificare la chiamata quando invii la risposta.

L'intestazione di traccia contiene l'ID di traccia, l'ID dell'elemento padre e la selezione per il campionamento. Se la richiesta viene campionata, la richiesta è stata campionata da Lambda o da un servizio upstream. Il runtime deve impostare `_X_AMZN_TRACE_ID` sul valore dell'intestazione. L'X-Ray SDK lo legge per ottenere IDs e determinare se tracciare la richiesta.

Risposta all'invocazione

Percorso – `/runtime/invocation/AwsRequestId/response`

Metodo – POST

Una volta che la funzione è stata eseguita fino al completamento, il runtime invia una risposta di chiamata a Lambda. Per le chiamate sincrone, Lambda invia la risposta al client.

Example Richiesta con esito positivo

```
REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response"  -d "SUCCESS"
```

Errore di inizializzazione

Se la funzione restituisce un errore o il runtime rileva un errore durante l'inizializzazione, il runtime utilizzerà questo metodo per segnalare l'errore a Lambda.

Percorso – `/runtime/init/error`

Metodo – POST

Headers

`Lambda-Runtime-Function-Error-Type` – Tipo di errore rilevato dal runtime. Campo obbligatorio: no.

L'intestazione è costituita da un valore stringa. Lambda accetta qualsiasi stringa, ma si consiglia di utilizzare il formato <categoria.motivo>. Esempio:

- Runtime. NoSuchHandler
- Tempo di esecuzione. APIKeyNotFound
- Tempo di esecuzione. ConfigInvalid
- Tempo di esecuzione. UnknownReason

Parametri corpo

`ErrorRequest` – Informazioni sull'errore. Campo obbligatorio: no.

Questo campo è un oggetto JSON con la seguente struttura:

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

NB: Lambda accetta qualsiasi valore per `errorType`.

Nell'esempio seguente viene mostrato un messaggio di errore della funzione Lambda in cui la funzione non è stata in grado di analizzare i dati evento forniti nell'invocazione.

Example Errore di funzione

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Parametri del corpo della risposta

- `StatusResponse` – Stringa. Informazioni sullo stato, inviate con codici di risposta 202.
- `ErrorResponse`— Informazioni aggiuntive sull'errore, inviate con i codici di risposta all'errore. ErrorResponse contiene un tipo di errore e un messaggio di errore.

Codice di risposta

- 202 – Accettato
- 403 – Non consentito
- 500 – Errore del container. Stato non recuperabile. Il runtime dovrebbe uscire tempestivamente.

Example Richiesta con errore di inizializzazione

```
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"
```

Errore della chiamata

Se la funzione restituisce un errore o il runtime rileva un errore, il runtime utilizzerà questo metodo per segnalare l'errore a Lambda.

Percorso – `/runtime/invocation/`*`AwsRequestId`*`/error`

Metodo – POST

Headers

`Lambda-Runtime-Function-Error-Type` – Tipo di errore rilevato dal runtime. Campo obbligatorio: no.

L'intestazione è costituita da un valore stringa. Lambda accetta qualsiasi stringa, ma si consiglia di utilizzare il formato <categoria.motivo>. Esempio:

- Runtime. NoSuchHandler
- Tempo di esecuzione. APIKeyNotFound
- Tempo di esecuzione. ConfigInvalid
- Tempo di esecuzione. UnknownReason

Parametri corpo

`ErrorRequest` – Informazioni sull'errore. Campo obbligatorio: no.

Questo campo è un oggetto JSON con la seguente struttura:

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

NB: Lambda accetta qualsiasi valore per `errorType`.

Nell'esempio seguente viene mostrato un messaggio di errore della funzione Lambda in cui la funzione non è stata in grado di analizzare i dati evento forniti nell'invocazione.

Example Errore di funzione

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Parametri del corpo della risposta

- `StatusResponse` – Stringa. Informazioni sullo stato, inviate con codici di risposta 202.
- `ErrorResponse`— Informazioni aggiuntive sull'errore, inviate con i codici di risposta all'errore. ErrorResponse contiene un tipo di errore e un messaggio di errore.

Codice di risposta

- 202 – Accettato
- 400 – Richiesta non valida
- 403 – Non consentito
- 500 – Errore del container. Stato non recuperabile. Il tempo di esecuzione dovrebbe uscire tempestivamente.

Example Richiesta con esito errato

```
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"
```

Quando usare i runtime solo per il sistema operativo di Lambda

Lambda fornisce runtime gestiti per Java, Python, Node.js, .NET e Ruby. Per creare funzioni Lambda in un linguaggio di programmazione non disponibile come runtime gestito, utilizza un runtime solo per il sistema operativo (la famiglia di runtime `provided`). Esistono tre casi d'uso principali per i runtime solo per il sistema operativo:

- Compilazione nativa ahead-of-time (AOT): linguaggi come Go, Rust, Swift e C++ vengono compilati nativamente in un binario eseguibile, che non richiede un runtime linguistico dedicato. Questi linguaggi richiedono solo un ambiente di sistema operativo in cui sia possibile eseguire il file binario compilato. Puoi anche utilizzare i runtime solo per il sistema operativo Lambda per distribuire file binari compilati con.NET Native AOT e Java GraalVM Native Image.

  È necessario includere un client dell'interfaccia di runtime nel file binario. Il client dell'interfaccia di runtime chiama Utilizzo dell'API di runtime Lambda per runtime personalizzati per recuperare le invocazioni della funzione e quindi esegue la chiamata al gestore della funzione. Lambda fornisce client di interfaccia runtime per Rust, Go, .NET Native AOT, Swift (sperimentale) e C++ (sperimentale).

  Devi compilare il file binario per un ambiente Linux e per la stessa architettura di set di istruzioni che intendi utilizzare per la funzione (x86_64 o arm64).
- Runtime di terze parti: puoi eseguire funzioni Lambda off-the-shelf utilizzando runtime come Bref per PHP.
- Runtime personalizzati: puoi creare il tuo runtime personale per un linguaggio o una versione di linguaggio per cui Lambda non fornisce un runtime gestito, come Node.js 19. Per ulteriori informazioni, consulta Creazione di un runtime personalizzato per AWS Lambda. Questo è il caso d'uso meno comune per i runtime solo per il sistema operativo.

Lambda supporta i seguenti runtime solo per il sistema operativo:

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Runtime solo per il sistema operativo	`provided.al2023`	Amazon Linux 2023	30 giugno 2029	31 luglio 2029	31 agosto 2029

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Runtime solo per il sistema operativo	`provided.al2`	Amazon Linux 2	31 luglio 2026	31 agosto 2026	30 settembre 2026

Il runtime Amazon Linux 2023 (`provided.al2023`) offre diversi vantaggi rispetto ad Amazon Linux 2, tra cui un'impronta di implementazione ridotta e versioni aggiornate di librerie come `glibc`.

Il runtime `provided.al2023` utilizza `dnf` come gestore di pacchetti invece di `yum`, che è il gestore di pacchetti predefinito in Amazon Linux 2. Per ulteriori informazioni sulle differenze tra `provided.al2023` e`provided.al2`, consulta Introducing the Amazon Linux 2023 runtime for AWS Lambda sul AWS Compute Blog.

Creazione di un runtime personalizzato per AWS Lambda

È possibile implementare un AWS Lambda runtime in qualsiasi linguaggio di programmazione. Il runtime è un programma che esegue un metodo del gestore della funzione Lambda quando la funzione viene richiamata. Puoi includere il runtime nel pacchetto di implementazione della funzione o distribuirlo in un livello. Quando crei la funzione Lambda, scegli un runtime solo per il sistema operativo (la famiglia di runtime `provided`).

> **Note**
>
> La creazione di un runtime personalizzato è un caso d'uso avanzato. Se stai cercando informazioni sulla compilazione in un file binario nativo o sull'utilizzo di un off-the-shelf runtime di terze parti, consultaQuando usare i runtime solo per il sistema operativo di Lambda.

Per una procedura guidata sul processo di implementazione del runtime personalizzato, consulta Tutorial: Creazione di un runtime personalizzato.

Argomenti

- Requisiti

- Implementazione dello streaming delle risposte in un runtime personalizzato
- Creazione di runtime personalizzati per istanze gestite Lambda

Requisiti

I runtime personalizzati devono completare determinate attività di inizializzazione ed elaborazione. Il runtime è responsabile dell'esecuzione del codice di configurazione della funzione, della lettura del nome del gestore da una variabile di ambiente e della lettura degli eventi di richiamo dall'API del runtime Lambda. Il runtime passa i dati dell'evento al gestore della funzioni e invia la risposta dal gestore a Lambda.

Attività di inizializzazione

Le attività di inizializzazione vengono eseguite una volta per istanza della funzione per preparare l'ambiente a gestire le chiamate.

- Recupero delle impostazioni – Leggi le variabili d'ambiente per ottenere dettagli sulla funzione e sull'ambiente.
  - `_HANDLER` – Il percorso del gestore della configurazione della funzione. Il formato standard è *`file`*`.`*`method`*, dove `file` è il nome del file senza estensione e `method` è il nome di un metodo o una funzione definita nel file.
  - `LAMBDA_TASK_ROOT` – La directory che contiene il codice della funzione.
  - `AWS_LAMBDA_RUNTIME_API` – L'host e la porta dell'API di runtime.

  Per l'elenco completo delle variabili disponibili, consulta la pagina Variabili di ambiente di runtime definite.
- Inizializzazione della funzione – Carica il file del gestore ed esegui il codice globale o statico che contiene. Le funzioni devono creare le risorse statiche come i client SDK e le connessioni al database e quindi riutilizzarle per più chiamate.
- Gestione degli errori – Se si verifica un errore, viene chiamata l'API errore di inizializzazione e viene chiusa la funzione.

L'inizializzazione viene calcolata nella fatturazione del runtime e del timeout. Quando un'esecuzione attiva l'inizializzazione di una nuova istanza della funzione, puoi visualizzare il tempo di inizializzazione nei log e nella traccia 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
```

Attività di elaborazione

Durante l'esecuzione, un runtime usa l'interfaccia del runtime Lambda per gestire gli eventi in entrata e segnalare gli errori. Dopo aver completato le attività di inizializzazione, il runtime elabora gli eventi in entrata in un ciclo. Nel codice di runtime, eseguire le seguenti fasi in ordine.

- Recupero di un evento – Viene invocata l'API chiamata successiva per ottenere l'evento seguente. Il corpo della risposta contiene i dati dell'evento. Le intestazioni di risposta contengono l'ID della richiesta e altre informazioni.
- Propagazione dell'intestazione di traccia – Recupera l'intestazione di traccia X-Ray dall'intestazione `Lambda-Runtime-Trace-Id` nella risposta dell'API. Imposta la variabile di ambiente `_X_AMZN_TRACE_ID` locale allo stesso valore. L'SDK X-Ray utilizza questo valore per associare i dati di tracciamento tra i servizi.
- Creazione di un oggetto contesto – Crea un oggetto con informazioni di contesto dalle variabili di ambiente e dalle intestazioni nella risposta dell'API.
- Richiamo del gestore della funzione – Passa l'evento e l'oggetto contesto al gestore.
- Gestione della risposta – Viene invocata l'API risposta della chiamata per pubblicare la risposta del gestore.
- Gestione degli errori – Se si verifica un errore, viene chiamata l'API errore di chiamata.
- Pulizia – Rilascia le risorse inutilizzate, invia i dati ad altri servizi o esegui attività aggiuntive prima di ottenere il prossimo evento.

Entrypoint

Il punto di ingresso di un runtime personalizzato è un file eseguibile denominato `bootstrap`. Il file di bootstrap può essere il runtime oppure può richiamare un altro file che crea il runtime. Se la root del pacchetto di implementazione non contiene un file denominato `bootstrap`, Lambda cerca il file nei livelli della funzione. Se il file `bootstrap` non esiste o non è eseguibile, la funzione restituisce un errore `Runtime.InvalidEntrypoint` in caso di invocazione.

Ecco un `bootstrap` file di esempio che utilizza una versione in bundle di Node.js per eseguire un JavaScript runtime in un file separato denominato. `runtime.js`

Example bootstrap

```
#!/bin/sh
    cd $LAMBDA_TASK_ROOT
    ./node-v11.1.0-linux-x64/bin/node runtime.js
```

Implementazione dello streaming delle risposte in un runtime personalizzato

Per le funzioni di streaming delle risposte, gli endpoint `response` ed `error` manifestano un comportamento leggermente diverso che consente al runtime di trasmettere risposte parziali al client e restituire i payload in blocchi. Per ulteriori informazioni sul comportamento specifico, consulta le seguenti risorse:

- `/runtime/invocation/AwsRequestId/response`: propaga l'intestazione `Content-Type` dal runtime per inviarla al client. Lambda restituisce il payload di risposta in blocchi tramite codifica di trasferimento in blocchi HTTP/1.1. Il flusso della risposta può avere una dimensione massima di 20 MiB. Per trasmettere la risposta in streaming a Lambda, il runtime deve:
  - Impostare l'intestazione HTTP `Lambda-Runtime-Function-Response-Mode` su `streaming`.
  - Imposta l'intestazione `Transfer-Encoding` su `chunked`.
  - Scrivere la risposta in conformità alla specifica di codifica del trasferimento in blocchi HTTP/1.1.
  - Chiudere la connessione sottostante dopo la corretta scrittura della risposta.
- `/runtime/invocation/AwsRequestId/error`: il runtime può utilizzare questo endpoint per segnalare errori di funzione o di runtime a Lambda, che accetta anche l'intestazione `Transfer-Encoding`. Questo endpoint può essere chiamato solo prima che il runtime inizi a inviare una risposta alla chiamata.
- Segnala gli errori intermedi utilizzando i trailer degli errori in `/runtime/invocation/AwsRequestId/response`: per segnalare gli errori che si verificano dopo aver iniziato a scrivere la risposta alla chiamata, il runtime può facoltativamente collegare intestazioni HTTP finali denominate `Lambda-Runtime-Function-Error-Type` e `Lambda-Runtime-Function-Error-Body`. Lambda considera questa come una risposta riuscita e inoltra i metadati degli errori che il runtime fornisce al client.

> **Note**
>
> Per allegare le intestazioni finali, il runtime deve impostare il valore dell'intestazione `Trailer` all'inizio della richiesta HTTP. Si tratta di un requisito della specifica di codifica di trasferimento in blocchi HTTP/1.1.

- `Lambda-Runtime-Function-Error-Type`: il tipo di errore rilevato dal runtime. L'intestazione è costituita da un valore stringa. Lambda accetta qualsiasi stringa, ma consigliamo un formato di. *`<category.reason>`* Ad esempio, `Runtime.APIKeyNotFound`.
- `Lambda-Runtime-Function-Error-Body`: informazioni sull'errore nella codifica base64.

Creazione di runtime personalizzati per istanze gestite Lambda

Le istanze gestite Lambda utilizzano la stessa API di runtime delle funzioni Lambda (predefinite). Tuttavia, esistono differenze fondamentali nel modo in cui devono essere implementati i runtime personalizzati per supportare il modello di esecuzione simultanea delle istanze gestite.

Gestione simultanea delle richieste

La differenza principale nella creazione di runtime personalizzati per le istanze gestite è il supporto per le chiamate simultanee. A differenza delle funzioni Lambda (predefinite) in cui il runtime elabora una chiamata alla volta, le istanze gestite possono elaborare più chiamate contemporaneamente all'interno di un unico ambiente di esecuzione.

Il runtime personalizzato deve:

- Supporta **`/next`** richieste simultanee: il runtime può effettuare più chiamate simultanee all'API di chiamata successiva, fino al limite specificato dalla variabile di ambiente. `AWS_LAMBDA_MAX_CONCURRENCY`
- Gestisci **`/response`** le richieste simultanee: più chiamate possono chiamare contemporaneamente l'API di risposta alle chiamate.
- Implementa la gestione delle richieste thread-safe: assicurati che le chiamate simultanee non interferiscano tra loro gestendo correttamente le risorse e lo stato condivisi.
- Usa una richiesta univoca IDs: traccia ogni chiamata separatamente utilizzando l'`Lambda-Runtime-Aws-Request-Id`intestazione per abbinare le risposte alle richieste corrispondenti.

Modello di implementazione

Un tipico modello di implementazione per i runtime delle istanze gestite prevede la creazione di thread o processi di lavoro per gestire le chiamate simultanee:

1. Leggi il limite di concorrenza: al momento dell'inizializzazione, leggi la variabile di `AWS_LAMBDA_MAX_CONCURRENCY` ambiente per determinare quante chiamate simultanee supportare.
2. Crea un pool di lavoratori: inizializza un pool di lavoratori (thread, processi o attività asincrone) pari al limite di concorrenza.
3. Ciclo di elaborazione dei lavoratori: ogni lavoratore in modo indipendente:
   - Chiamate `/runtime/invocation/next` per ottenere un evento di invocazione
   - Richiama il gestore della funzione con i dati dell'evento
   - Invia la risposta a `/runtime/invocation/AwsRequestId/response`
   - Ripete il ciclo

Ulteriori considerazioni

- Formato di registrazione: le istanze gestite supportano solo il formato di registro JSON. Assicurati che il runtime rispetti la variabile di `AWS_LAMBDA_LOG_FORMAT` ambiente e utilizzi solo il formato JSON. Per ulteriori informazioni, consulta Configurazione dei formati di log JSON e testo normale.
- Risorse condivise: fai attenzione quando utilizzi risorse condivise come la `/tmp` directory con chiamate simultanee. Implementa meccanismi di blocco adeguati per prevenire condizioni di gara.

Per ulteriori informazioni sugli ambienti di esecuzione di Lambda Managed Instances, vedere. Comprendere l'ambiente di esecuzione di Lambda Managed Instances

Tutorial: Creazione di un runtime personalizzato

In questo tutorial crei una funzione Lambda con un runtime personalizzato. Si inizia includendo il runtime nel pacchetto di distribuzione della funzione. Quindi lo trasferisci in un livello che gestisci in modo indipendente dalla funzione. Infine, condividi il livello del runtime con tutti aggiornando la policy delle autorizzazioni basate sulle risorse.

Prerequisiti

Questo tutorial presuppone una certa conoscenza delle operazioni di base di Lambda e della console relativa. Se non lo si è già fatto, seguire le istruzioni riportate in Creare una funzione Lambda con la console per creare la prima funzione Lambda.

Per completare i passaggi seguenti, è necessaria la AWS CLI versione 2. I comandi e l'output previsto sono elencati in blocchi separati:

```
aws --version
```

Verrà visualizzato l'output seguente:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Per i comandi lunghi viene utilizzato un carattere di escape (\) per dividere un comando su più righe.

In Linux e macOS utilizzare la propria shell e il proprio programma di gestione dei pacchetti preferiti.

> **Note**
>
> Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere una versione integrata su Windows di Ubuntu e Bash, installa il sottosistema Windows per Linux. I comandi della CLI di esempio in questa guida utilizzano la formattazione Linux. Se si utilizza la CLI di Windows, i comandi che includono documenti JSON in linea dovranno essere riformattati.

È necessario un ruolo IAM per creare una funzione Lambda. Il ruolo richiede l'autorizzazione per inviare i log a CloudWatch Logs e accedere ai Servizi AWS utilizzati dalla tua funzione. Se non hai un ruolo per lo sviluppo di funzioni, creane uno ora.

Per creare un ruolo di esecuzione

1. Apri la pagina Ruoli nella console IAM.
2. Scegliere Crea ruolo.
3. Creare un ruolo con le seguenti proprietà.

- Trusted entity (Entità attendibile – Lambda
- Permissions (Autorizzazioni) – AWSLambdaBasicExecutionRole.
- Role name (Nome ruolo – **`lambda-role`**.

La policy AWSLambdaBasicExecutionRole dispone delle autorizzazioni delle quali la funzione necessita per scrivere i log in CloudWatch Logs.

Creazione di una funzione

Crea una funzione Lambda con un runtime personalizzato. Questo esempio include due file: un file `bootstrap` del runtime e un gestore della funzione. Entrambi sono implementati in Bash.

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir runtime-tutorial
cd runtime-tutorial
```

2. Crea un nuovo file denominato `bootstrap`. Questo è il runtime personalizzato.

   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)
```

```
  # Run the handler function from the script
  RESPONSE=$($(echo "$_HANDLER" | cut -d. -f2) "$EVENT_DATA")

  # Send the response
  curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response"  -d "$RESPONSE"
done
```

Il runtime carica uno script di funzione dal pacchetto di distribuzione. Utilizza due variabili per individuare lo script. `LAMBDA_TASK_ROOT` indica il percorso da cui il pacchetto è stato estratto e `_HANDLER` include il nome dello script.

Dopo che il runtime carica lo script della funzione, utilizza l'API del runtime per recuperare un evento di chiamata da Lambda, passa l'evento al gestore e trasmette la risposta a Lambda. Per ottenere l'ID della richiesta, il runtime salva le intestazioni dalla risposta dell'API in un file temporaneo e legge l'intestazione `Lambda-Runtime-Aws-Request-Id` dal file.

> **Note**
>
> I runtime hanno responsabilità aggiuntive inclusa la gestione degli errori e forniscono al gestore le informazioni sul contesto. Per informazioni dettagliate, vedi Requisiti.

3. Crea uno script per la funzione. Lo script di esempio seguente definisce una funzione del gestore che richiede i dati dell'evento, li registra in `stderr` e li restituisce.

   Example function.sh

   ```
   function handler () {
     EVENT_DATA=$1
     echo "$EVENT_DATA" 1>&2;
     RESPONSE="Echoing request: '$EVENT_DATA'"

     echo $RESPONSE
   }
   ```

   L'aspetto della directory `runtime-tutorial` dovrebbe essere simile al seguente:

   ```
   runtime-tutorial
   # bootstrap
   ```

```
# function.sh
```

4. Rendere i file eseguibili e aggiungerli ad un archive ZIP. Questo è il pacchetto di implementazione.

```
chmod 755 function.sh bootstrap
zip function.zip function.sh bootstrap
```

5. Crea una funzione denominata `bash-runtime`. Per `--role`, inserisci l'ARN del tuo ruolo di esecuzione Lambda.

```
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. Richiama la funzione.

```
aws lambda invoke --function-name bash-runtime --payload '{"text":"Hello"}'
 response.txt --cli-binary-format raw-in-base64-out
```

L'opzione cli-binary-format è necessaria se si utilizza la versione 2 della AWS CLI. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Dovresti ottenere una risposta simile a questa:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

7. Verifica la risposta.

```
cat response.txt
```

Dovresti ottenere una risposta simile a questa:

```
Echoing request: '{"text":"Hello"}'
```

Crea un livello

Per separare il codice del runtime dal codice della funzione, crea un livello che contenga solo il runtime. I livelli consentono di sviluppare le dipendenze della funzione in modo indipendente e possono ridurre l'utilizzo dello storage quando usi lo stesso livello con più funzioni. Per ulteriori informazioni, consulta Gestione delle dipendenze Lambda con i livelli.

1. Crea un file .zip contenente il file `bootstrap`.

   ```
   zip runtime.zip bootstrap
   ```

2. Crea un livello con il comando publish-layer-version.

   ```
   aws lambda publish-layer-version --layer-name bash-runtime --zip-file fileb://
   runtime.zip
   ```

   In tal modo viene creata la prima versione del livello.

Aggiorna la funzione

Per utilizzare il livello del runtime nella funzione, configura la funzione affinché utilizzi il livello e rimuovi il codice del runtime dalla funzione.

1. Aggiorna la configurazione della funzione da inserire nel livello.

   ```
   aws lambda update-function-configuration --function-name bash-runtime \
   --layers arn:aws:lambda:us-east-1:123456789012:layer:bash-runtime:1
   ```

   Questo aggiunge il runtime alla funzione nella directory `/opt`. Per garantire che Lambda utilizzi il runtime nel livello, è necessario rimuovere il `boostrap` dal pacchetto di implementazione della funzione, come illustrato nei due passaggi successivi.

2. Crea un file .zip contenente il codice della funzione.

   ```
   zip function-only.zip function.sh
   ```

3. Aggiorna il codice della funzione in modo da includere soltanto lo script del gestore.

```
aws lambda update-function-code --function-name bash-runtime --zip-file fileb://
function-only.zip
```

4. Chiama la funzione per verificare che funzioni con il livello del runtime.

```
aws lambda invoke --function-name bash-runtime --payload '{"text":"Hello"}'
 response.txt --cli-binary-format raw-in-base64-out
```

L'opzione cli-binary-format è necessaria se si utilizza la versione 2 della AWS CLI. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Dovresti ottenere una risposta simile a questa:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
```

5. Verifica la risposta.

```
cat response.txt
```

Dovresti ottenere una risposta simile a questa:

```
Echoing request: '{"text":"Hello"}'
```

Aggiorna il runtime

1. Per registrare le informazioni sull'ambiente di esecuzione, aggiorna lo script del runtime sulle variabili di ambiente di 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)

  # Run the handler function from the script
  RESPONSE=$($(echo "$_HANDLER" | cut -d. -f2) "$EVENT_DATA")

  # Send the response
  curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response"  -d "$RESPONSE"
done
```

2. Crea un file .zip contenente la nuova versione del file `bootstrap`.

```
zip runtime.zip bootstrap
```

3. Crea una nuova versione del livello `bash-runtime`.

```
aws lambda publish-layer-version --layer-name bash-runtime --zip-file fileb://
runtime.zip
```

4. Configura la funzione per utilizzare la nuova versione del livello.

```
aws lambda update-function-configuration --function-name bash-runtime \
--layers arn:aws:lambda:us-east-1:123456789012:layer:bash-runtime:2
```

Condividi il livello

Per condividere un livello con un altro Account AWS, aggiungi una istruzione di autorizzazione multi-account alla policy basata sulle risorse del livello. Esegui il comando add-layer-version-permission e specifica l'ID dell'account come `principal`. In ogni istruzione, puoi concedere l'autorizzazione a un singolo account, a tutti gli account o a un'organizzazione in AWS Organizations.

L'esempio seguente concede all'account 111122223333 l'accesso alla versione 2 del livello `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
```

Verrà visualizzato un output simile al seguente:

```
{"Sid":"xaccount","Effect":"Allow","Principal":
{"AWS":"arn:aws:iam::111122223333:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:
east-1:123456789012:layer:bash-runtime:2"}
```

Le autorizzazioni si applicano solo a un'unica versione di un livello. Ripeti la procedura ogni volta che crei la nuova versione di un livello.

Eliminazione

Eliminare ciascuna versione del livello.

```
aws lambda delete-layer-version --layer-name bash-runtime --version-number 1
aws lambda delete-layer-version --layer-name bash-runtime --version-number 2
```

Poiché contiene un riferimento alla versione 2 del livello, la funzione è ancora presente in Lambda. Continua a operare, ma le funzioni non possono più essere configurate per utilizzare la versione eliminata. Se modifichi l'elenco dei livelli sulla funzione, devi specificare una nuova versione oppure omettere il livello eliminato.

Elimina la funzione con il comando delete-function.

```
aws lambda delete-function --function-name bash-runtime
```

Archivi open source

AWS Lambda offre una varietà di strumenti, librerie e componenti open source per aiutarti a creare, personalizzare e ottimizzare le tue applicazioni serverless. Queste risorse includono client di interfaccia di runtime, librerie di eventi, immagini di base di container, strumenti di sviluppo e progetti di esempio gestiti AWS e disponibili su GitHub. Sfruttando questi repository open source, puoi estendere le funzionalità di Lambda, creare runtime personalizzati, elaborare eventi da vari AWS servizi e ottenere informazioni più approfondite sulle prestazioni della tua funzione. Questa pagina fornisce una panoramica dei principali progetti open source che supportano lo sviluppo di Lambda.

Client di interfaccia Runtime

I Lambda Runtime Interface Client (RICs) sono librerie open source che implementano l'API Runtime e gestiscono l'interazione tra il codice della funzione e il servizio Lambda. Questi client gestiscono la ricezione di eventi di chiamata, il passaggio di informazioni di contesto e la segnalazione degli errori.

I client dell'interfaccia di runtime utilizzati dai runtime gestiti e dalle immagini di base dei container di Lambda sono pubblicati come open source. Quando crei runtime personalizzati o estendi quelli esistenti, puoi utilizzare queste librerie open source per semplificare l'implementazione. I seguenti GitHub repository open source contengono il codice sorgente di Lambda: RICs

- Client di interfaccia Node.js Runtime
- Client di interfaccia Python Runtime
- Client di interfaccia Java Runtime
- Client di interfaccia Ruby Runtime
- Client di interfaccia di runtime .NET
- Client di interfaccia Rust Runtime
- Client di interfaccia Go Runtime
- Client di interfaccia Swift Runtime (sperimentale)
- Client di interfaccia Runtime C++ (sperimentale)
- Immagini Lambda Base

Per ulteriori informazioni sull'utilizzo di questi client per creare runtime personalizzati, consulta. the section called “Compilazione di un runtime personalizzato”

Librerie di eventi

Le librerie di eventi Lambda forniscono definizioni dei tipi e utilità di supporto per l'elaborazione di eventi da vari servizi. AWS Queste librerie consentono di analizzare e gestire i dati degli eventi in modo sicuro dai tipi, semplificando l'utilizzo di eventi provenienti da servizi come Amazon S3, Amazon DynamoDB e Amazon API Gateway.

Per i linguaggi compilati, AWS fornisce le seguenti librerie di eventi:

- Libreria di eventi Java
- Librerie di eventi.NET
- Libreria eventi Go
- Libreria di eventi Rust

Per linguaggi interpretati come Node.js, Python e Ruby, gli eventi possono essere analizzati direttamente come oggetti JSON senza richiedere una libreria separata. Tuttavia, gli sviluppatori che utilizzano Node.js e Python possono sfruttare powertools for AWS Lambda, che fornisce schemi integrati per AWS eventi che offrono suggerimenti sui tipi, convalida dei dati e funzionalità simili a quelle fornite dalle librerie di linguaggi compilati.

- Powertools per TypeScript
- Powertools per Python

Immagini di base del contenitore

AWS fornisce immagini di base di container open source che puoi usare come punto di partenza per creare immagini di container per le tue funzioni Lambda. Queste immagini di base includono il client dell'interfaccia di runtime e altri componenti necessari per eseguire le funzioni nell'ambiente di esecuzione Lambda.

Per ulteriori informazioni sulle immagini di base disponibili e su come utilizzarle, consulta l'archivio delle immagini di base AWS Lambda e. the section called “Immagini di container”

Strumenti di sviluppo

AWS fornisce strumenti di sviluppo open source aggiuntivi per aiutarti a creare e ottimizzare le tue funzioni Lambda:

Powertools per AWS Lambda

Powertools for AWS Lambda semplifica lo sviluppo senza server con utilità essenziali per prevenire l'elaborazione duplicata e l'elaborazione in batch per la gestione di più record e la libreria di consumo Kafka. Queste funzionalità consentono di ridurre al minimo la complessità del codice e il sovraccarico operativo.

Puoi anche sfruttare la convalida dello schema degli eventi, la registrazione e il tracciamento strutturati e l'integrazione dell'archivio di parametri, progettati per accelerare la creazione di funzioni Lambda pronte per la produzione seguendo al contempo best practice ben architettate. AWS

GitHub archivi:

- Python
- TypeScript
- Java
- .NET

strumenti di sviluppo Java

- Java Profiler (sperimentale): uno strumento per la profilazione delle funzioni Java Lambda.
- Librerie Java: un repository che contiene una raccolta completa di librerie e strumenti Java per lo sviluppo di Lambda, inclusi progetti chiave JUnit come utilità di test e strumenti di profilazione.
- Serverless Java Container: una libreria che consente di eseguire applicazioni Java esistenti su Lambda con modifiche minime.

Strumenti di sviluppo.NET

L'archivio AWS Lambda .NET fornisce librerie.NET e strumenti per lo sviluppo Lambda, inclusi progetti chiave come gli strumenti AWS Lambda per l'interfaccia della riga di comando DI.NET e il server .NET Core per l'hosting di applicazioni.NET Core.

Progetti di esempio

Esplora una raccolta completa di esempi di progetti e applicazioni Lambda nei repository Serverless Land. Questi esempi illustrano vari casi d'uso, modelli di integrazione e best practice di Lambda per aiutarti a iniziare a utilizzare le tue applicazioni serverless.

Configurazione delle funzioni AWS Lambda

Scopri come configurare le funzionalità e le opzioni principali per la funzione Lambda tramite l'API o la console Lambda.

archivi di file.zip

Crea un pacchetto di distribuzione delle funzioni Lambda quando desideri includere dipendenze, livelli di runtime personalizzati o qualsiasi file oltre al codice della funzione. Il pacchetto di distribuzione è un archivio di file.zip contenente il codice della funzione e le dipendenze.

Immagini di container

Usa le immagini dei contenitori per impacchettare il codice della funzione e le dipendenze quando hai bisogno di un maggiore controllo sul processo di compilazione o se la tua funzione richiede configurazioni di runtime personalizzate. Puoi creare, testare e distribuire funzioni Lambda come immagini di container utilizzando strumenti come Docker CLI.

Memoria

Scopri come e quando aumentare la memoria della funzione.

Archiviazione temporanea

Scopri come e quando aumentare la capacità di archiviazione temporanea della tua funzione.

Timeout

Scopri come e quando aumentare il valore di timeout della funzione.

Configurazione durevole

Consenti alle funzioni Lambda di funzionare per periodi prolungati, fino a un anno, mantenendo lo stato tra le chiamate. Configura i timeout di esecuzione, la conservazione dello stato e il comportamento di controllo delle versioni per flussi di lavoro a lunga durata e con stato. Le funzioni durevoli consentono di creare processi complessi e in più fasi in grado di sospendere, riprendere e gestire automaticamente le interruzioni.

Variabili d'ambiente

È possibile rendere il codice della funzione portabile e nascondere i segreti del codice archiviandoli nella configurazione della funzione utilizzando le variabili di ambiente.

Rete in uscita

Puoi usare la tua funzione Lambda con AWS risorse in un Amazon VPC. La connessione della funzione a un VPC consente di accedere alle risorse in una sottorete privata, ad esempio database relazionali e cache.

Rete in entrata

È possibile utilizzare un endpoint VPC dell'interfaccia per richiamare le funzioni Lambda senza attraversare l'Internet pubblico.

File system

È possibile utilizzare la funzione Lambda per montare un Amazon EFS in una directory locale. Un file system consente al codice della funzione di accedere e modificare le risorse condivise in modo sicuro e con una simultaneità elevata.

Alias

Quindi è possibile configurare i client per invocare una versione della funzione Lambda specifica utilizzando un alias anziché aggiornando il client.

Versioni

Pubblicando una versione della funzione, sarà possibile archiviare il codice e la configurazione come risorse separate che non possono essere modificate.

Tag

Utilizza i tag per abilitare il controllo degli accessi basato sugli attributi (ABAC), per organizzare le funzioni Lambda e per filtrare e generare report sulle tue funzioni utilizzando i nostri servizi Billing and AWS Cost Explorer Cost AWS Management.

Streaming delle risposte

Puoi configurare la tua funzione Lambda URLs per trasmettere i payload di risposta ai client. Lo streaming delle risposte può favorire le applicazioni sensibili alla latenza migliorando le prestazioni del time to first byte (TTFB). Questo perché consente di inviare risposte parziali al client non appena diventano disponibili. Inoltre, lo streaming delle risposte permette di creare funzioni che restituiscono payload più grandi.

Distribuzione di funzioni Lambda come archivi di file .zip

Quando si crea una funzione Lambda, viene utilizzato un pacchetto di implementazione per impacchettare il codice della funzione. Lambda supporta due tipi di pacchetti di implementazione: immagini di container e archivi di file .zip. Il flusso di lavoro per creare una funzione dipende dal tipo di pacchetto di implementazione. Per creare una funzione definita come immagine di container, consulta the section called “Immagini di container”.

Puoi utilizzare la console Lambda e l'API Lambda per creare una funzione definita con un archivio di file .zip. È inoltre possibile caricare un file .zip aggiornato per modificare il codice funzione.

> **Note**
>
> Non è possibile modificare il tipo di pacchetto di implementazione (.zip o immagine di container) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine di container esistente per utilizzare un archivio di file .zip. È necessario creare una nuova funzione.

Argomenti

- Creazione della funzione
- Utilizzo dell'editor di codice della console
- Aggiornamento del codice della funzione
- Modifica del runtime
- Modifica dell'architettura
- Utilizzo dell'API Lambda
- Download del codice della funzione
- CloudFormation
- Crittografia dei pacchetti di implementazione .zip per Lambda

Creazione della funzione

Quando si crea una funzione definita con un archivio di file .zip, si scelgono un modello di codice, la versione della lingua e il ruolo di esecuzione per la funzione. Si aggiunge il codice della funzione dopo che Lambda ha creato la funzione.

Creazione della funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli Crea funzione.
3. Scegliere Author from scratch (Crea da zero) o Use a blueprint (Usa un piano) per creare la funzione
4. In Basic information (Informazioni di base) eseguire queste operazioni:
   a. In Function name (Nome funzione), immettere il nome della funzione. I nomi delle funzioni hanno un limite di lunghezza di 64 caratteri.
   b. Per Runtime, scegliere la versione della lingua da utilizzare per la funzione.
   c. (Opzionale) Per Architecture (Architettura), scegli l'architettura del set di istruzioni da utilizzare per la funzione. L'architettura predefinita è x86_64. Quando crei il pacchetto di implementazione per la tua funzione, assicurati che sia compatibile con questa architettura del set di istruzioni.
5. (Opzionale) In Autorizzazioni espandere Modifica ruolo di esecuzione predefinito. Puoi creare un nuovo ruolo di esecuzione o utilizzare un ruolo esistente.
6. (Facoltativo) Espandi Advanced settings (Impostazioni avanzate). È possibile scegliere una Code signing configuration (Configurazione della firma del codice) per la funzione Puoi anche configurare un (Amazon VPC) per accedere alla funzione.
7. Scegli Crea funzione.

Lambda crea la nuova funzione. È ora possibile utilizzare la console per aggiungere il codice della funzione e configurare altri parametri di funzione e funzionalità. Per le istruzioni di distribuzione del codice, consulta la pagina del gestore del runtime utilizzato dalla funzione.

Node.js

Distribuisci funzioni Lambda in Node.js con archivi di file .zip

Python

Utilizzo di archivi di file .zip per le funzioni Lambda in Python

Ruby

Distribuire le funzioni Ruby Lambda con gli archivi di file .zip

Java

- Distribuisci funzioni Lambda per Java con archivi di file .zip o JAR

Go

- Distribuisci funzioni Lambda per Go con gli archivi di file .zip

C#

- Crea e implementa le funzioni Lambda C# con gli archivi di file .zip

PowerShell

- Distribuzione delle funzioni Lambda di PowerShell con gli archivi di file .zip

Utilizzo dell'editor di codice della console

La console crea una funzione Lambda con un unico file di origine. Per i linguaggi di scripting, è possibile modificare questo file e aggiungere altri file con l'editor di codice incorporato. Per salvare le modifiche, scegliere Save (Salva). Quindi, per eseguire il codice, scegliere Test (Testa).

Quando si salva il codice funzione, la console Lambda crea un pacchetto di implementazione dell'archivio di file .zip. Quando sviluppi il codice funzione al di fuori della console (utilizzando un IDE) devi creare un pacchetto di implementazione per caricare il codice nella funzione Lambda.

Aggiornamento del codice della funzione

Per i linguaggi di scripting (Node.js, Python e Ruby), puoi modificare il codice della funzione nell'editor di codice incorporato. Se il codice supera i 3 MB, o se è necessario aggiungere librerie, o per linguaggi che l'editor non supporta (come Java, Go e C#), è necessario caricare il codice funzione come archivio .zip. Se l'archivio di file .zip è inferiore a 50 MB, è possibile caricare l'archivio di file .zip direttamente dal computer locale. Se le dimensioni dell'archivio .zip sono maggiori di 50 MB, carica il file sulla funzione da un bucket Amazon S3.

Per caricare il codice funzione come archivio .zip

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere la funzione da aggiornare e scegliere la scheda Codice
3. In Code source (Origine codice), scegli Upload from (Carica da).

4. Scegli .zip file, quindi scegli Upload (Carica).
   - Nella finestra di selezione dei file, selezionare la nuova versione dell'immagine e scegliere Open (Apri), quindi scegliere Save (Salva).
5. (Alternativa al passaggio 4) Scegli Percorso Amazon S3.
   - Nella casella di testo, inserisci l'URL del link S3 dell'archivio di file .zip, quindi scegli Save (Salva).

Modifica del runtime

Se si aggiorna la configurazione della funzione in modo da utilizzare una nuova versione di runtime, potrebbe essere necessario aggiornare il codice della funzione per renderlo compatibile con la nuova versione. Se si aggiorna la configurazione della funzione per utilizzare un runtime diverso, è necessario fornire un nuovo codice funzione compatibile con il runtime e l'architettura. Per istruzioni su come creare un pacchetto di implementazione per il codice della funzione, consulta la pagina del gestore per il runtime utilizzato dalla funzione.

Le immagini di base Node.js 20, Python 3.12, Java 21, .NET 8, Ruby 3.3 e versioni successive si basano sull'immagine minima del container di Amazon Linux 2023. Le immagini di base precedenti utilizzavano Amazon Linux 2. AL2023 offre diversi vantaggi rispetto ad Amazon Linux 2, tra cui un ingombro di distribuzione ridotto e versioni aggiornate di librerie come. `glibc` Per ulteriori informazioni, consulta il post Introducing the Amazon Linux 2023 runtime for AWS Lambda del blog AWS Compute.

Modifica del runtime

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere la funzione da aggiornare e scegliere la scheda Codice
3. Scorrere fino alla sezione Runtime settings (Impostazioni runtime) sotto l'editor di codice.
4. Scegli Modifica.
   a. Per Runtime, seleziona l'identificatore di runtime.
   b. In Gestore, specifica il nome del file e il gestore per la funzione.
   c. Per Architecture (Architettura), scegli l'architettura del set di istruzioni da utilizzare per la funzione.
5. Scegli Save (Salva).

Modifica dell'architettura

Prima di poter modificare l'architettura del set di istruzioni, è necessario assicurarsi che il codice della funzione sia compatibile con l'architettura di destinazione.

Se usi Node.js, Python o Ruby e modifichi il codice della funzione nell'editor incorporato, il codice esistente può essere eseguito senza modifiche.

Tuttavia, se si fornisce il codice della funzione utilizzando un pacchetto di implementazione con un archivio di file. zip, è necessario preparare un nuovo archivio di file .zip compilato e sviluppato correttamente per il runtime e l'architettura del set di istruzioni di destinazione. Per le istruzioni, consulta la pagina del gestore del runtime della funzione.

Per modificare l'architettura del set di istruzioni

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere la funzione da aggiornare e scegliere la scheda Codice
3. In Impostazioni runtime, scegliere Modifica.
4. Per Architecture (Architettura), scegli l'architettura del set di istruzioni da utilizzare per la funzione.
5. Scegli Save (Salva).

Utilizzo dell'API Lambda

Per creare e configurare una funzione che utilizza un archivio di file .zip, utilizzare le seguenti operazioni API:

- CreateFunction
- UpdateFunctionCode
- UpdateFunctionConfiguration

Download del codice della funzione

Puoi scaricare la versione corrente non pubblicata (`$LATEST`) del tuo codice funzione .zip tramite la console Lambda. Per fare ciò, assicurati innanzitutto di disporre delle seguenti autorizzazioni IAM:

- `iam:GetPolicy`

- `iam:GetPolicyVersion`
- `iam:GetRole`
- `iam:GetRolePolicy`
- `iam:ListAttachedRolePolicies`
- `iam:ListRolePolicies`
- `iam:ListRoles`

Per scaricare il file .zip del codice della funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la funzione per la quale desideri scaricare il file .zip codice della funzione.
3. Nella panoramica delle funzioni, scegli il pulsante Scarica, quindi Scarica il file .zip del codice della funzione.

    - In alternativa, scegli Scarica AWS SAM file per generare e scaricare un modello SAM basato sulla configurazione della tua funzione. Puoi anche scegliere Scarica entrambi per scaricare sia il file .zip che il modello SAM.

CloudFormation

È possibile utilizzare CloudFormation per creare una funzione Lambda che utilizza un archivio di file.zip. Nel CloudFormation modello, la `AWS::Lambda::Function` risorsa specifica la funzione Lambda. Per le descrizioni delle proprietà della `AWS::Lambda::Function` risorsa, AWS::Lambda::Functionconsultate la Guida per l'AWS CloudFormation utente.

Nella risorsa `AWS::Lambda::Function`, impostare le seguenti proprietà per creare una funzione definita come archivio di file .zip:

- AWS::Lambda::Function
    - PackageType — Impostato su`Zip`.
    - Codice: inserisci il nome del bucket Amazon S3 e il nome del file .zip nei campi `S3Bucket` e `S3Key`. Per Node.js o Python, puoi fornire il codice sorgente inline della funzione Lambda.
    - Runtime: imposta il valore di runtime.
    - Architettura: imposta il valore dell'architettura su cui `arm64` utilizzare il processore AWS Graviton2. Per impostazione predefinita, il valore dell'architettura è `x86_64`.

Crittografia dei pacchetti di implementazione .zip per Lambda

Lambda fornisce sempre la crittografia lato server dei dati a riposo per i pacchetti di implementazione .zip e i dettagli di configurazione delle funzioni con una AWS KMS key. Per impostazione predefinita, Lambda utilizza un Chiave di proprietà di AWS. Se questo comportamento predefinito si adatta al tuo flusso di lavoro, non devi configurare nient'altro. AWS non ti addebita alcun costo per l'utilizzo di questa chiave.

Se preferisci, puoi invece fornire una chiave gestita AWS KMS dal cliente. È possibile eseguire questa operazione per avere il controllo sulla rotazione della chiave KMS o per soddisfare i requisiti dell'organizzazione per la gestione delle chiavi KMS. Quando si utilizza una chiave gestita dal cliente, solo gli utenti del tuo account con accesso alla chiave KMS possono visualizzare o gestire il codice o la configurazione della funzione.

Le chiavi gestite dal cliente sono soggette a costi standard AWS KMS . Per ulteriori informazioni, consultare Prezzi di AWS Key Management Service.

Creazione di una chiave gestita dal cliente

È possibile creare una chiave simmetrica gestita dal cliente utilizzando, o il Console di gestione AWS. AWS KMS APIs

Per creare una chiave simmetrica gestita dal cliente

Segui le fasi descritte in Creazione delle chiavi KMS per la crittografia simmetrica nella Guida per gli sviluppatori di AWS Key Management Service .

Permissions

Policy della chiave

Le policy della chiave controllano l'accesso alla chiave gestita dal cliente. Ogni chiave gestita dal cliente deve avere esattamente una policy della chiave, che contiene istruzioni che determinano chi può usare la chiave e come la possono usare. Per ulteriori informazioni, consulta Come modificare una policy delle chiavi nella Guida per gli sviluppatori di AWS Key Management Service .

Quando si utilizza una chiave gestita dal cliente per crittografare un pacchetto di implementazione .zip, Lambda non aggiunge alcuna concessione alla chiave. Invece, la tua policy AWS KMS chiave deve consentire a Lambda di chiamare le seguenti operazioni AWS KMS API per tuo conto:

- km: GenerateDataKey
- kms:Decrypt

Il seguente esempio di policy chiave consente a tutte le funzioni Lambda nell'account 111122223333 di richiamare le AWS KMS operazioni richieste per la chiave gestita dal cliente specificata:

Example AWS KMS politica chiave

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:*"
                }
            }
        }
    ]
}
```

Per informazioni sulla Risoluzione dei problemi delle chiavi di accesso consulta la Guida per gli sviluppatori di AWS Key Management Service .

Autorizzazioni del principale

Quando si utilizza una chiave gestita dal cliente per crittografare un pacchetto di implementazione .zip, solo i principali che hanno accesso a quella chiave possono accedere al

pacchetto di implementazione .zip. Ad esempio, i responsabili che non hanno accesso alla chiave gestita dal cliente non possono scaricare il pacchetto.zip utilizzando l'URL S3 predefinito incluso nella risposta. GetFunction `AccessDeniedException` viene restituito nella sessione `Code` della risposta.

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"
    },
 ...
```

Per ulteriori informazioni sulle autorizzazioni per le AWS KMS chiavi, consulta Autenticazione e controllo degli accessi per. AWS KMS

Utilizzo di una chiave gestita dal cliente per il pacchetto di implementazione .zip

Utilizza i seguenti parametri API per configurare le chiavi gestite dal cliente per i pacchetti di implementazione .zip:

- Source KMSKey Arn: crittografa il pacchetto di distribuzione.zip di origine (il file che carichi).
- KMSKeyArn: crittografa le variabili di ambiente e le istantanee Lambda. SnapStart

Quando `SourceKMSKeyArn` e `KMSKeyArn` sono specificati entrambi, Lambda utilizza la chiave `KMSKeyArn` per crittografare la versione decompressa del pacchetto utilizzata da Lambda per richiamare la funzione. Quando `SourceKMSKeyArn` è specificato ma `KMSKeyArn` non lo è, Lambda utilizza una Chiave gestita da AWS per crittografare la versione decompressa del pacchetto.

Lambda console

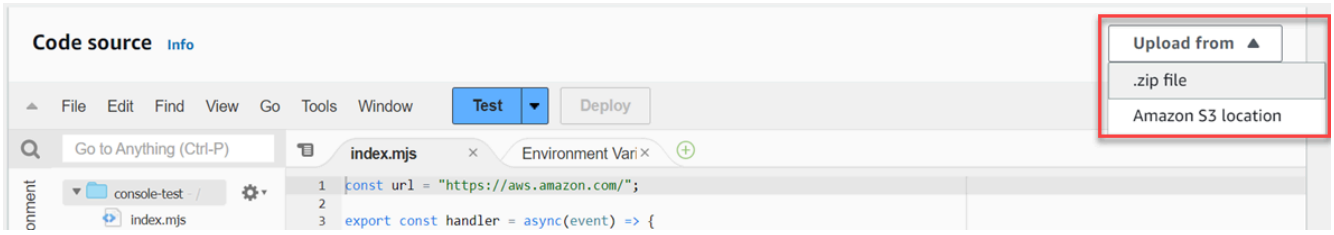
Per aggiungere la crittografia della chiave gestita dal cliente durante la creazione di una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli Crea funzione.
3. Scegliere Author from scratch (Crea da zero) o Container image (Immagine di container).
4. In Basic information (Informazioni di base) eseguire queste operazioni:
   a. In Function name (Nome funzione), immettere il nome della funzione.
   b. Per Runtime, scegliere la versione della lingua da utilizzare per la funzione.
5. Espandi Impostazioni avanzate, quindi seleziona Abilita la crittografia con una AWS KMS chiave gestita dal cliente.
6. Scegli una chiave gestita dal cliente.
7. Scegli Crea funzione.

Per rimuovere la crittografia con la chiave gestita dal cliente o per utilizzare una chiave diversa, è necessario caricare nuovamente il pacchetto di implementazione .zip.

Per aggiungere la crittografia con la chiave gestita dal cliente a una funzione esistente

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli il file .zip o la posizione di Amazon S3.



5. Carica il file o inserisci la posizione Amazon S3.
6. Scegli Abilita la crittografia con una chiave gestita dal AWS KMS cliente.
7. Scegli una chiave gestita dal cliente.
8. Scegli Save (Salva).

AWS CLI

Per aggiungere la crittografia della chiave gestita dal cliente durante la creazione di una funzione

Nel seguente esempio create-function:

- `--code`: specifica il percorso locale del pacchetto di distribuzione.zip (`ZipFile`) e la chiave gestita dal cliente per crittografarlo (). `SourceKMSKeyArn`
- `--kms-key-arn`: specifica la chiave gestita dal cliente per crittografare le variabili d'ambiente e la versione compressa del pacchetto di implementazione.

```
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
```

Nel seguente esempio create-function:

- `--code`: specifica la posizione del file.zip in un bucket Amazon S3 `S3Bucket` (`S3Key`,`S3ObjectVersion`,) e la chiave gestita dal cliente per crittografarlo (). `SourceKMSKeyArn`
- `--kms-key-arn`: specifica la chiave gestita dal cliente per crittografare le variabili d'ambiente e la versione compressa del pacchetto di implementazione.

```
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
```

Per aggiungere la crittografia con la chiave gestita dal cliente a una funzione esistente

Nell'esempio seguente: update-function-code

- `--zip-file`: specifica il percorso locale del pacchetto di implementazione .zip.
- `--source-kms-key-arn`: specifica la chiave gestita dal cliente per crittografare la versione compressa del pacchetto di implementazione. Lambda utilizza una chiave AWS proprietaria per crittografare il pacchetto decompresso per le chiamate di funzioni. Se desideri utilizzare una chiave gestita dal cliente per crittografare la versione decompressa del pacchetto, esegui il comando con l'opzione. update-function-configuration`--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
```

Nel seguente esempio: update-function-code

- `--s3-bucket`: specifica la posizione del file .zip in un bucket Amazon S3.
- `--s3-key`: specifica la chiave Amazon S3 del pacchetto di implementazione.
- `--s3-object-version`: per gli oggetti con versione, la versione dell'oggetto del pacchetto di implementazione da utilizzare.
- `--source-kms-key-arn`: specifica la chiave gestita dal cliente per crittografare la versione compressa del pacchetto di implementazione. Lambda utilizza una chiave AWS proprietaria per crittografare il pacchetto decompresso per le chiamate di funzioni. Se desideri utilizzare una chiave gestita dal cliente per crittografare la versione decompressa del pacchetto, esegui il comando con l'opzione. update-function-configuration`--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
```

Per rimuovere la crittografia con la chiave gestita dal cliente da una funzione esistente

Nell'update-function-codeesempio seguente, `--zip-file` specifica il percorso locale del pacchetto di distribuzione.zip. Quando esegui questo comando senza l'`--source-kms-key-`

`arn`opzione, Lambda utilizza una chiave AWS proprietaria per crittografare la versione compressa del pacchetto di distribuzione.

```
aws lambda update-function-code \
  --function-name myFunction \
  --zip-file fileb://myFunction.zip
```

Creare una funzione Lambda utilizzando un'immagine di container

Il codice della AWS Lambda funzione è costituito da script o programmi compilati e dalle relative dipendenze. Utilizza un pacchetto di implementazione per distribuire il codice della funzione a Lambda. Lambda supporta due tipi di pacchetti di implementazione: immagini di container e archivi di file .zip.

Esistono tre modi per creare un'immagine di container per una funzione Lambda:

- Usare un'immagine AWS di base per Lambda

  Le immagini di base AWS sono precaricate con un runtime in linguaggio, un client di interfaccia di runtime per gestire l'interazione tra Lambda e il codice della funzione e un emulatore di interfaccia di runtime per i test locali.

- Utilizzo di un'immagine di AWS base solo per il sistema operativo

  AWS Le immagini di base solo per il sistema operativo contengono una distribuzione Amazon Linux e l'emulatore di interfaccia di runtime. Queste immagini vengono comunemente utilizzate per creare immagini di container per linguaggi compilati, come Go e Rust, e per un linguaggio o una versione di linguaggio per cui Lambda non fornisce un'immagine di base, come Node.js 19. Puoi anche utilizzare immagini di base solo per il sistema operativo per implementare un runtime personalizzato. Per rendere l'immagine compatibile con Lambda, devi includere nell'immagine un client di interfaccia di runtime per il tuo linguaggio.

- Utilizzo di un'immagine non AWS di base

  È possibile utilizzare un'immagine di base alternativa da un altro registro del container, come ad esempio Alpine Linux o Debian. Puoi anche utilizzare un'immagine personalizzata creata dalla tua organizzazione. Per rendere l'immagine compatibile con Lambda, devi includere nell'immagine un client di interfaccia di runtime per il tuo linguaggio.

> **Tip**
>
> Per ridurre il tempo necessario all'attivazione delle funzioni del container Lambda, consulta Utilizzo di compilazioni a più fasi nella documentazione Docker. Per creare immagini di container efficienti, segui le best practice per scrivere file Docker.

Per creare una funzione Lambda da un'immagine di un container, crea l'immagine localmente e caricala in un repository Amazon Elastic Container Registry (Amazon ECR). Se si utilizza un'immagine container fornita da un venditore Marketplace AWS, devi prima clonare l'immagine nel tuo repository Amazon ECR privato. Quindi, specifica l'URI del repository al momento della creazione della funzione. Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS È possibile creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta Autorizzazioni multiaccount Amazon ECR.

> **Note**
>
> Lambda non supporta gli endpoint FIPS di Amazon ECR per le immagini dei container. Se l'URI del repository include `ecr-fips`, stai utilizzando un endpoint FIPS. Esempio: `111122223333.dkr.ecr-fips.us-east-1.amazonaws.com`.

Questa pagina spiega i tipi di immagini di base e i requisiti per la creazione di immagini di container compatibili con Lambda.

> **Note**
>
> Non è possibile modificare il tipo di pacchetto di implementazione (.zip o immagine di container) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine di container esistente per utilizzare un archivio di file .zip. È necessario creare una nuova funzione.

Argomenti

- Requisiti
- Usare un'immagine AWS di base per Lambda
- Utilizzo di un'immagine di AWS base solo per il sistema operativo
- Utilizzo di un'immagine non AWS di base
- Client di interfaccia runtime
- Autorizzazioni Amazon ECR
- Ciclo di vita delle funzioni

Requisiti

Installa la AWS CLI versione 2 e la CLI Docker. Sono inoltre necessari i seguenti requisiti:

- L'immagine di container deve implementare l'Utilizzo dell'API di runtime Lambda per runtime personalizzati. I client dell'interfaccia runtime AWS open-source implementano l'API. È possibile aggiungere un client di interfaccia di runtime all'immagine di base preferita per renderla compatibile con Lambda.
- L'immagine di container deve essere in grado di funzionare su un filesystem di sola lettura. Il codice della funzione può accedere a una directory `/tmp` scrivibile con spazio di storage compreso tra 512 MB e 10.240 MB con incrementi di 1 MB.
- L'utente Lambda predefinito deve essere in grado di leggere tutti i file necessari per eseguire il codice della funzione. Lambda segue le best practice di sicurezza tramite la definizione di un utente Linux predefinito con autorizzazioni meno privilegiate. Ciò significa che non è necessario specificare un USER nel Dockerfile. Verifica che il codice dell'applicazione non si basi su file che altri utenti Linux non possono eseguire.
- Lambda supporta solo le immagini di container basate su Linux.
- Lambda fornisce immagini di base multi-architettura. Tuttavia, l'immagine creata per la tua funzione deve essere destinata a una sola delle architetture. Lambda non supporta funzioni che utilizzano immagini container multi-architettura.

Usare un'immagine AWS di base per Lambda

È possibile utilizzare una delle immagini di base AWS per Lambda per creare l'immagine di container per il codice della funzione. Le immagini di base sono precaricate con un runtime di lingua e altri componenti necessari per eseguire un'immagine container su Lambda. Aggiungere il codice funzione e le dipendenze all'immagine di base e quindi impacchettarlo come immagine container.

AWS fornisce periodicamente aggiornamenti alle immagini di AWS base per Lambda. Se il Dockerfile include il nome dell'immagine nella proprietà FROM, il client Docker estrae l'ultima versione dell'immagine dal repository Amazon ECR. Per utilizzare l'immagine di base aggiornata, è necessario ricostruire l'immagine di container e aggiornare il codice della funzione.

Le immagini di base Node.js 20, Python 3.12, Java 21, .NET 8, Ruby 3.3 e versioni successive si basano sull'immagine minima del container di Amazon Linux 2023. Le immagini di base precedenti utilizzavano Amazon Linux 2. AL2023 offre diversi vantaggi rispetto ad Amazon Linux 2, tra cui un ingombro di distribuzione ridotto e versioni aggiornate di librerie come. `glibc`

AL2Le immagini basate su 023 utilizzano `microdnf` (symlinked as`dnf`) come gestore di pacchetti anzichéyum, che è il gestore di pacchetti predefinito in Amazon Linux 2. `microdnf`è un'implementazione autonoma di. `dnf` Per un elenco dei pacchetti inclusi nelle immagini AL2 basate su 023, consulta le colonne Minimal Container in Confronto dei pacchetti installati su Amazon Linux 2023 Container Images. Per ulteriori informazioni sulle differenze tra AL2 023 e Amazon Linux 2, consulta la sezione Introduzione al runtime di Amazon Linux 2023 AWS Lambda sul AWS Compute Blog.

> **Note**
>
> Per eseguire immagini AL2 basate su 023 localmente, incluso with AWS Serverless Application Model (AWS SAM), devi usare Docker versione 20.10.10 o successiva.

Per creare un'immagine del contenitore utilizzando un'immagine di AWS base, scegli le istruzioni per la tua lingua preferita:

- Node.js
- TypeScript(utilizza un'immagine di base Node.js)
- Python
- Java
- Go
- .NET
- Ruby

Utilizzo di un'immagine di AWS base solo per il sistema operativo

AWS Le immagini di base solo per il sistema operativo contengono una distribuzione Amazon Linux e l'emulatore di interfaccia di runtime. Queste immagini vengono comunemente utilizzate per creare immagini di container per linguaggi compilati, come Go e Rust, e per un linguaggio o una versione di linguaggio per cui Lambda non fornisce un'immagine di base, come Node.js 19. Puoi anche utilizzare immagini di base solo per il sistema operativo per implementare un runtime personalizzato. Per rendere l'immagine compatibile con Lambda, devi includere nell'immagine un client di interfaccia di runtime per il tuo linguaggio.

Tag	Runtime	Sistema operativo	Dockerfile	Raggiunta obsolescenza
al2023	Runtime solo per il sistema operativo	Amazon Linux 2023	Dockerfile per Runtime solo per sistema operativo su GitHub	30 giugno 2029
al2	Runtime solo per il sistema operativo	Amazon Linux 2	Dockerfile per Runtime solo per sistema operativo su GitHub	31 luglio 2026

Galleria pubblica di Amazon Elastic Container Registry: gallery.ecr. aws/lambda/provided

Utilizzo di un'immagine non AWS di base

Lambda supporta qualsiasi immagine conforme a uno dei seguenti formati manifest per le immagini:

- Docker Image Manifest V2 Schema 2 (utilizzato con Docker versione 1.10 e successive)
- Open Container Initiative (OCI) Specifications (v1.0.0 e successive)

Lambda supporta una dimensione massima dell'immagine non compressa pari a 10 GB, inclusi tutti i livelli.

> **Note**
>
> - Per rendere l'immagine compatibile con Lambda, devi includere nell'immagine un client di interfaccia di runtime per il tuo linguaggio.
> - Per prestazioni ottimali, mantenere la dimensione del manifesto dell'immagine al di sotto di 25.400 byte. Per ridurre le dimensioni del manifesto dell'immagine, ridurre al minimo il numero di livelli nell'immagine e riduci le annotazioni.

Client di interfaccia runtime

Se utilizzi un'immagine di base solo per il sistema operativo o un'immagine di base alternativa, devi includere un client dell'interfaccia di runtime nell'immagine. Il client dell'interfaccia di runtime deve estendereUtilizzo dell'API di runtime Lambda per runtime personalizzati, che gestisce l'interazione tra Lambda e il codice della funzione. AWS fornisce client di interfaccia di runtime open source per le seguenti lingue:

- Node.js
- Python
- Java
- .NET
- Go
- Ruby
- Ruggine —

Se utilizzi un linguaggio che non dispone di un AWS client di interfaccia di runtime fornito, devi crearne uno tuo.

Autorizzazioni Amazon ECR

Prima di creare la funzione Lambda da un'immagine di container, è necessario creare un'immagine del container in locale e caricarla in un repository Amazon ECR. Quando crei la funzione, specifica l'URI del repository Amazon ECR.

Assicurati che le autorizzazioni per l'utente o il ruolo che crea la funzione contengano `GetRepositoryPolicy` e `SetRepositoryPolicy`.

Ad esempio, utilizza la console IAM per creare un ruolo con la seguente policy:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
```

```
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "ecr:SetRepositoryPolicy",
        "ecr:GetRepositoryPolicy"
      ],
      "Resource": "arn:aws:ecr:us-east-1:111122223333:repository/hello-world"
    }
  ]
}
```

Policy del repository Amazon ECR

Per una funzione nello stesso account dell'immagine di container in Amazon ECR, puoi aggiungere le autorizzazioni `ecr:BatchGetImage` e `ecr:GetDownloadUrlForLayer` alla policy del tuo repository Amazon ECR. L'esempio seguente mostra il valore minimo della policy.

```
{
        "Sid": "LambdaECRImageRetrievalPolicy",
        "Effect": "Allow",
        "Principal": {
          "Service": "lambda.amazonaws.com"
        },
        "Action": [
          "ecr:BatchGetImage",
          "ecr:GetDownloadUrlForLayer"
        ]
    }
```

Per ulteriori informazioni sulle autorizzazioni per il repository Amazon ECR, consulta la pagina Policy dei repository privati nella Guida per l'utente di Amazon Elastic Container Registry.

Se il repository Amazon ECR non include queste autorizzazioni, Lambda tenta di aggiungerle automaticamente. Lambda può aggiungere autorizzazioni solo se l'entità principale che chiama Lambda dispone delle autorizzazioni `ecr:getRepositoryPolicy` e `ecr:setRepositoryPolicy`.

Per visualizzare o modificare le autorizzazioni del repository Amazon ECR, segui le istruzioni riportate nella pagina Impostazione di un'istruzione di policy per i repository privati della Guida per l'utente di Amazon Elastic Container Registry.

Autorizzazioni multiaccount Amazon ECR

Un account diverso nella stessa Regione può creare una funzione che utilizza un'immagine container di proprietà del tuo account. Nell'esempio seguente, la policy delle autorizzazioni del repository Amazon ECR richiede le seguenti istruzioni per concedere l'accesso all'account numero 123456789012.

- CrossAccountPermission— Consente all'account 123456789012 di creare e aggiornare funzioni Lambda che utilizzano immagini da questo repository ECR.
- Lambda: ECRImage CrossAccountRetrievalPolicy Lambda alla fine imposterà lo stato di una funzione su inattivo se non viene richiamata per un periodo prolungato. Questa istruzione è necessaria affinché Lambda possa recuperare l'immagine del container per l'ottimizzazione e la memorizzazione nella cache per conto della funzione di proprietà di 123456789012.

Example - Aggiunta di autorizzazioni multi-account al repository

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"
    }
  ]
}
```

Per consentire l'accesso a più account, aggiungi l'account IDs all'elenco Principal nella `CrossAccountPermission` policy e all'elenco di valutazione delle condizioni in. `LambdaECRImageCrossAccountRetrievalPolicy`

Se lavori con più account in un' AWS organizzazione, ti consigliamo di enumerare ogni ID account nella politica di autorizzazione ECR. Questo approccio è in linea con le migliori pratiche di AWS sicurezza che prevedono l'impostazione di autorizzazioni ristrette nelle policy IAM.

Oltre alle autorizzazioni Lambda, l'utente o il ruolo che crea la funzione deve disporre anche delle autorizzazioni `BatchGetImage` e `GetDownloadUrlForLayer`.

Ciclo di vita delle funzioni

Dopo aver caricato un'immagine container nuova o aggiornata, Lambda ottimizza l'immagine prima che la funzione possa elaborare le chiamate. Il processo di ottimizzazione può richiedere alcuni secondi. La funzione rimane nello stato `Pending` fino al completamento del processo, quando lo stato passa a `Active`. Non è possibile richiamare la funzione fino a quando non raggiunge lo stato `Active`.

Se una funzione non viene richiamata per più settimane, Lambda recupera la sua versione ottimizzata e la funzione passa allo stato `Inactive`. Per riattivare la funzione, è necessario invocarla. Lambda rifiuta la prima invocazione e la funzione entra nello stato `Pending` finché Lambda non riottimizza l'immagine. La funzione ritorna quindi allo stato `Active`.

Lambda recupera periodicamente l'immagine di container associata dal repository Amazon ECR. Se l'immagine di container corrispondente non esiste più in Amazon ECR o le autorizzazioni sono state revocate, la funzione entra nello stato `Failed` e Lambda restituisce un errore per qualsiasi chiamata della funzione.

È possibile utilizzare l'API Lambda per ottenere informazioni sullo stato di una funzione. Per ulteriori informazioni, consulta Stati funzione Lambda.

Configurare la memoria di una funzione Lambda

Lambda alloca la potenza della CPU in proporzione alla quantità di memoria configurata. Memory (Memoria) indica la quantità di memoria disponibile per la funzione Lambda in fase di runtime. È possibile aumentare o diminuire la memoria e la potenza della CPU assegnate alla funzione utilizzando l'impostazione Memoria. È possibile configurare memoria compresa tra 128 MB e 10.240 MB in incrementi di 1 MB. A 1.769 MB, una funzione ha l'equivalente di una vCPU (un vCPU-secondo di crediti al secondo).

In questa pagina viene descritto come e quando aggiornare l'impostazione della memoria per una funzione Lambda.

Sections

- Determinazione dell'impostazione di memoria appropriata per una funzione Lambda
- Configurazione della memoria delle funzioni (console)
- Configurazione della memoria delle funzioni (AWS CLI)
- Configurazione della memoria delle funzioni (AWS SAM)
- Accettazione dei suggerimenti relativi alla memoria delle funzioni (console)

Determinazione dell'impostazione di memoria appropriata per una funzione Lambda

La memoria è la leva principale per controllare le prestazioni di una funzione. L'impostazione predefinita, 128 MB, è l'impostazione più bassa possibile. Ti consigliamo di utilizzare 128 MB solo per funzioni Lambda semplici, come quelle che trasformano e instradano gli eventi verso altri servizi AWS . Una allocazione maggiore della memoria può migliorare le prestazioni per le funzioni che utilizzano librerie importate, livelli Lambda, Amazon Simple Storage Service (Amazon S3) o Amazon Elastic File System (Amazon EFS). L'aggiunta di più memoria aumenta proporzionalmente la quantità di CPU, aumentando la potenza di calcolo complessiva disponibile. Se una funzione è collegata alla CPU, alla rete o alla memoria, l'aumento dell'impostazione della memoria può migliorarne notevolmente le prestazioni.

Per trovare la giusta configurazione di memoria, monitora le tue funzioni con Amazon CloudWatch e imposta allarmi se il consumo di memoria si avvicina ai massimi configurati. Questo può aiutare a identificare le funzioni legate alla memoria. Per le funzioni legate alla CPU e all'IO, anche il

monitoraggio della durata può fornire informazioni. In questi casi, l'aumento della memoria può aiutare a risolvere i problemi di elaborazione o di rete.

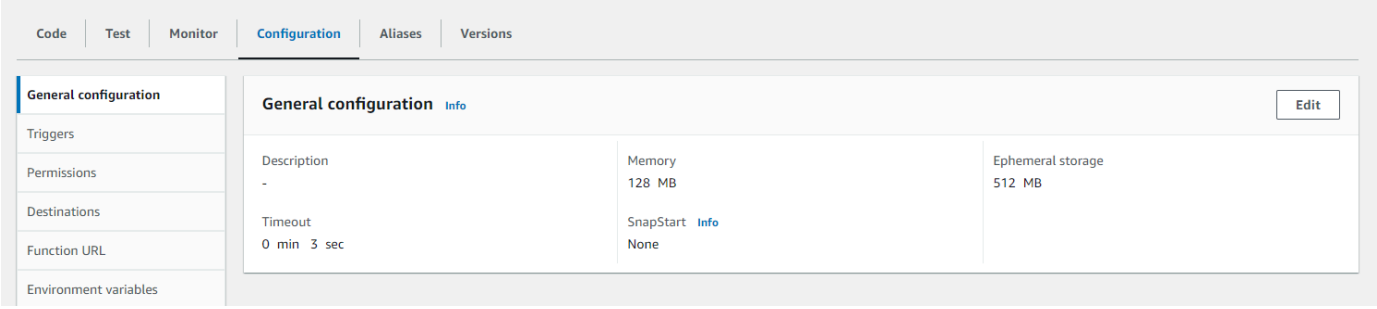
Puoi anche prendere in considerazione l'utilizzo dello strumento open source Power Tuning AWS Lambda. Questo strumento consente AWS Step Functions di eseguire più versioni simultanee di una funzione Lambda con diverse allocazioni di memoria e misurare le prestazioni. La funzione di input viene eseguita nell' AWS account ed esegue chiamate HTTP in tempo reale e interazioni SDK, per misurare le probabili prestazioni in uno scenario di produzione live. Puoi anche implementare un CI/CD processo per utilizzare questo strumento per misurare automaticamente le prestazioni delle nuove funzioni che distribuisci.

Configurazione della memoria delle funzioni (console)

È possibile configurare la memoria della funzione nella console Lambda.

Per aggiornare la memoria di una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Seleziona la scheda Configurazione, quindi scegli Configurazione generale.



4. In Configurazione generale, scegli Modifica.
5. Per Memoria, impostare un valore compreso tra 128 MB e 10.240 MB.
6. Scegli Save (Salva).

Configurazione della memoria delle funzioni (AWS CLI)

È possibile utilizzare il update-function-configurationcomando per configurare la memoria della funzione.

Example

```
aws lambda update-function-configuration \
  --function-name my-function \
  --memory-size 1024
```

Configurazione della memoria delle funzioni (AWS SAM)

Per configurare la memoria per la tua funzione, puoi usare AWS Serverless Application Model. Aggiorna la MemorySizeproprietà nel tuo `template.yaml` file e poi esegui 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...
```

Accettazione dei suggerimenti relativi alla memoria delle funzioni (console)

Se disponi delle autorizzazioni di amministratore in AWS Identity and Access Management (IAM), puoi scegliere di ricevere consigli sulle impostazioni della memoria della funzione Lambda da. AWS Compute Optimizer Per istruzioni su come attivare i suggerimenti sulla memoria per il proprio account o la propria organizzazione, consultare Accettazione delle opzioni nell'account nella Guida per l'utente di AWS Compute Optimizer .

> **Note**
>
> Compute Optimizer supporta solo le funzioni che utilizzano l'architettura x86_64.

Dopo avere accettato le opzioni e se la funzione Lambda soddisfa i requisiti del Sistema di ottimizzazione del calcolo, puoi visualizzare e accettare i suggerimenti sulla memoria della funzione del Sistema di ottimizzazione del calcolo nella console Lambda in Configurazione generale.

Configurare lo spazio di archiviazione temporaneo per le funzioni Lambda

Lambda fornisce uno spazio di archiviazione temporaneo per le funzioni nella directory `/tmp`. Questo spazio di archiviazione è temporaneo e unico per ogni ambiente di esecuzione. È possibile controllare la quantità di spazio di archiviazione temporaneo allocato alla funzione utilizzando l'impostazione Archiviazione temporanea. È possibile configurare uno spazio di archiviazione temporaneo compreso tra 512 MB e 10.240 MB in incrementi di 1 MB. Tutti i dati archiviati in `/tmp` vengono crittografati a riposo con una chiave gestita da AWS.

In questa pagina sono descritti i casi d'uso comuni e come aggiornare lo spazio di archiviazione temporanea per una funzione Lambda.

Sections

- Casi d'uso comuni per una maggiore archiviazione temporanea
- Configurazione dell'archiviazione temporanea (console)
- Configurazione dello spazio di archiviazione temporanea (AWS CLI)
- Configurazione dello spazio di archiviazione temporanea (AWS SAM)

Casi d'uso comuni per una maggiore archiviazione temporanea

Ecco alcuni casi d'uso comuni che traggono vantaggio dall'aumento dello spazio di archiviazione temporanea:

- Extract-transform-load Lavori (ETL): aumenta lo storage temporaneo quando il codice esegue calcoli intermedi o scarica altre risorse per completare l'elaborazione. Più spazio temporaneo consente l'esecuzione di lavori ETL più complessi nelle funzioni Lambda.
- Inferenza con il machine learning (ML): molte attività di inferenza si basano su file di dati di riferimento di grandi dimensioni, tra cui librerie e modelli. Con più spazio di archiviazione temporanea, puoi scaricare modelli più grandi da Amazon Simple Storage Service (Amazon S3) in `/tmp` e utilizzarli nell'elaborazione.
- Elaborazione dei dati: per i carichi di lavoro che scaricano oggetti da Amazon S3 in risposta a eventi S3, più spazio `/tmp` consente di gestire oggetti più grandi senza utilizzare l'elaborazione in memoria. Anche i carichi di lavoro che creano PDFs o elaborano contenuti multimediali traggono vantaggio da uno storage più effimero.

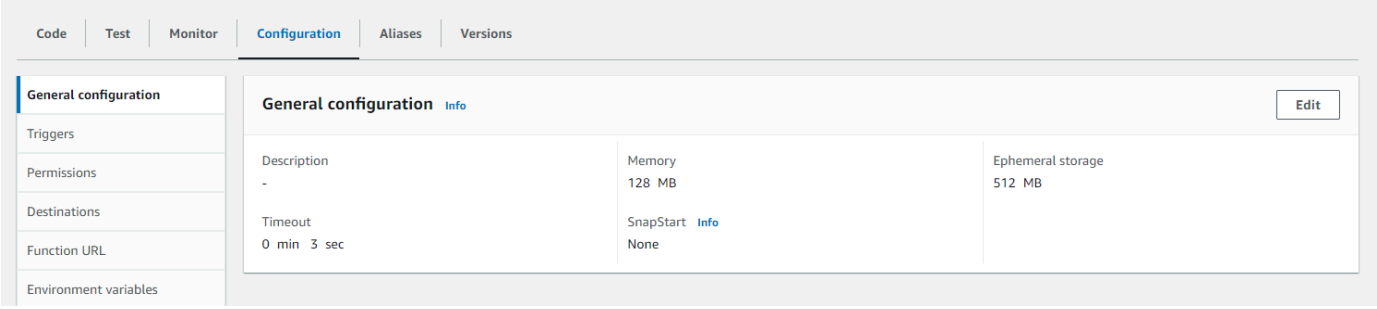
- Elaborazione grafica: l'elaborazione delle immagini è un caso d'uso comune per le applicazioni basate su Lambda. Per i carichi di lavoro che elaborano file TIFF di grandi dimensioni o immagini satellitari, uno spazio di archiviazione temporanea più grande semplifica l'uso delle librerie e l'esecuzione del calcolo in Lambda.

Configurazione dell'archiviazione temporanea (console)

Puoi configurare lo spazio di archiviazione temporanea nella console Lambda.

Per modificare lo spazio di archiviazione temporanea effimera per una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Seleziona la scheda Configurazione, quindi scegli Configurazione generale.



4. In Configurazione generale, scegli Modifica.
5. Per Archiviazione temporanea, imposta un valore compreso tra 512 MB e 10.240 MB in incrementi di 1 MB.
6. Scegli Save (Salva).

Configurazione dello spazio di archiviazione temporanea (AWS CLI)

È possibile utilizzare il update-function-configurationcomando per configurare l'archiviazione temporanea.

Example

```
aws lambda update-function-configuration \
  --function-name my-function \
  --ephemeral-storage '{"Size": 1024}'
```

Configurazione dello spazio di archiviazione temporanea (AWS SAM)

Per configurare lo spazio di archiviazione temporanea per la tua funzione, puoi usare AWS Serverless Application Model. Aggiorna la EphemeralStorageproprietà nel tuo `template.yaml` file e poi esegui 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...
```

Selezione e configurazione di un'architettura di set di istruzioni per la funzione Lambda

L'architettura del set di istruzioni di una funzione Lambda determina il tipo di processore del computer utilizzato da Lambda per eseguire la funzione. Lambda offre una scelta di architetture del set di istruzioni:

- arm64 — Architettura ARM a 64 bit, per il processore Graviton2 AWS .
- x86_64: architettura x86 a 64 bit, per processori basati su x86.

> **Note**
>
> L'architettura arm64 è disponibile nella maggior parte dei casi. Regioni AWS Per ulteriori informazioni, consultare AWS Lambda Prezzi. Nella tabella dei prezzi della memoria, scegli la scheda Arm Price, quindi apri l'elenco a discesa Region per vedere quali Regioni AWS supportano arm64 con Lambda.
> Per un esempio di come creare una funzione con l'architettura arm64, vedi AWS Lambda Functions Powered by Graviton2 Processor. AWS

Argomenti

- Vantaggi dell'utilizzo dell'architettura arm64
- Requisiti per la migrazione all'architettura arm64
- Compatibilità del codice della funzione con l'architettura arm64
- Come eseguire la migrazione all'architettura arm64
- Configurazione dell'architettura del set di istruzioni

Vantaggi dell'utilizzo dell'architettura arm64

Le funzioni Lambda che utilizzano l'architettura arm64 (processore AWS Graviton2) possono ottenere prezzi e prestazioni significativamente migliori rispetto alla funzione equivalente in esecuzione sull'architettura x86_64. Prendi in considerazione l'utilizzo di arm64 per applicazioni ad uso intensivo di calcolo come computing ad alte prestazioni, codifica video e carichi di lavoro di simulazione.

La CPU Graviton2 utilizza il core Neoverse N1 e supporta Armv8.2 (incluse le estensioni CRC e crittografiche) oltre a diverse altre estensioni architettoniche.

Graviton2 riduce il tempo di lettura della memoria fornendo una cache L2 maggiore per vCPU, che migliora le prestazioni di latenza dei back-end Web e per dispositivi mobili, dei microservizi e dei sistemi di elaborazione dati. Inoltre, Graviton2 offre prestazioni di crittografia migliorate e supporta set di istruzioni che migliorano la latenza dell'inferenza di machine learning basata sulla CPU.

Per ulteriori informazioni su Graviton2, vedere Graviton Processor. AWSAWS

Requisiti per la migrazione all'architettura arm64

Quando selezioni una funzione Lambda per migrare all'architettura arm64, per garantire una migrazione fluida assicurati che la tua funzione soddisfi i seguenti requisiti:

- Il pacchetto di implementazione contiene solo componenti open source e codice sorgente sotto il tuo controllo, in modo da poter apportare tutti gli aggiornamenti necessari per la migrazione.
- Se il codice delle funzione include dipendenze di terze parti, ogni libreria o pacchetto fornisce una versione arm64.

Compatibilità del codice della funzione con l'architettura arm64

Il codice della funzione Lambda deve essere compatibile con l'architettura del set di istruzioni della funzione. Prima di migrare una funzione all'architettura arm64, tieni presente le seguenti osservazioni sul codice della funzione corrente:

- Se hai aggiunto il codice della funzione utilizzando l'editor di codice incorporato, probabilmente è possibile eseguire il codice su entrambe le architetture senza modifiche.
- Se il codice della funzione è stato caricato, è necessario caricare un nuovo codice compatibile con l'architettura specifica.
- Se la funzione utilizza livelli, è necessario controllare ogni livello per garantire che sia compatibile con la nuova architettura. Se un livello non è compatibile, modifica la funzione per sostituire la versione del livello corrente con una versione del livello compatibile.
- Se la tua funzione utilizza estensioni Lambda, è necessario controllare ciascuna estensione per assicurarsi che sia compatibile con la nuova architettura.
- Se la funzione usa un tipo di pacchetto di distribuzione dell'immagine del container, è necessario creare una nuova immagine del container compatibile con l'architettura della funzione.

Come eseguire la migrazione all'architettura arm64

Per migrare una funzione Lambda all'architettura arm64, ti consigliamo di seguire questi passaggi:

1. Crea l'elenco di dipendenze per l'applicazione o il carico di lavoro. Le dipendenze comuni includono:
   - Tutte le librerie e i pacchetti utilizzati dalla funzione.
   - Gli strumenti utilizzati per creare, distribuire e testare la funzione, come compilatori, suite di test, pipeline di integrazione continua e distribuzione continua (CI/CD), strumenti di provisioning e script.
   - Le estensioni Lambda e gli strumenti di terza parte utilizzati per monitorare la funzione in produzione.
2. Per ciascuna delle dipendenze, controlla la versione e verifica se sono disponibili versioni arm64.
3. Crea un ambiente per eseguire la migrazione dell'applicazione.
4. Esegui il bootstrap dell'applicazione.
5. Test e debugging dell'applicazione.
6. Testa le prestazioni della funzione arm64. Confronta le prestazioni con la versione x86_64.
7. Aggiorna la pipeline dell'infrastruttura per supportare le funzioni arm64 Lambda.
8. Simula l'implementazione per la produzione.

   Ad esempio, è possibile usare la configurazione del routing dell'alias per dividere il traffico tra le versioni x86 e arm64 della funzione e confrontare le prestazioni e la latenza.

Per ulteriori informazioni su come creare un ambiente di codice per l'architettura arm64, incluse informazioni specifiche del linguaggio per Java, Go, .NET e Python, consulta il repository Getting started with Graviton. AWS GitHub

Configurazione dell'architettura del set di istruzioni

Puoi configurare l'architettura del set di istruzioni per funzioni Lambda nuove ed esistenti utilizzando la console Lambda, AWS SDKs, AWS Command Line Interface ()AWS CLI o. CloudFormation Completa questa procedura per modificare l'architettura del set di istruzioni per una funzione Lambda esistente dalla console.

1. Aprire la pagina Funzioni della console Lambda.

2. Scegli il nome della funzione per la quale desideri configurare l'architettura del set di istruzioni.
3. Nella scheda Codice principale, per la sezione Impostazioni di runtime, scegli Modifica.
4. In Architettura, scegli l'architettura del set di istruzioni da utilizzare per la funzione.
5. Scegli Save (Salva).

Configurare il timeout della funzione Lambda

Lambda esegue il codice per un determinato periodo di tempo prima del timeout. Il timeout è il tempo massimo in secondi in cui una funzione Lambda può essere eseguita. Il valore predefinito per questa impostazione è 3 secondi, ma è possibile regolarlo con incrementi di 1 secondo fino a un valore massimo di 900 secondi (15 minuti).

In questa pagina viene descritto come e quando aggiornare l'impostazione di timeout per una funzione Lambda.

Sections

- Determinazione del valore di timeout appropriato per una funzione Lambda
- Configurazione del timeout (console)
- Configurazione del timeout (AWS CLI)
- Configurazione del timeout (AWS SAM)

Determinazione del valore di timeout appropriato per una funzione Lambda

Se un valore di timeout viene impostato vicino alla durata media di una funzione, c'è un rischio elevato che la funzione scada inaspettatamente. La durata di una funzione può variare in base alla quantità di trasferimento ed elaborazione dei dati e alla latenza dei servizi con cui interagisce la funzione. Alcune cause comuni di timeout includono:

- I download da Amazon Simple Storage Service (Amazon S3) sono di dimensioni maggiori o richiedono più tempo della media.
- Una funzione invia una richiesta a un altro servizio, che impiega più tempo a rispondere.
- I parametri forniti a una funzione richiedono una maggiore complessità computazionale della funzione, il che fa sì che l'invocazione richieda più tempo.

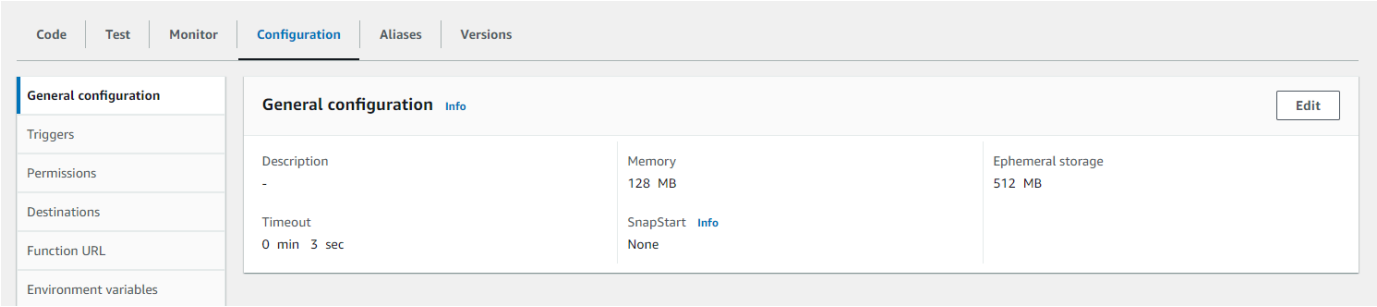
Nel testare l'applicazione, assicurati che i test riflettano accuratamente la dimensione e la quantità di dati e valori realistici dei parametri. I test utilizzano spesso campioni di piccole dimensioni per comodità, ma è consigliabile utilizzare set di dati al limite massimo di quanto ragionevolmente previsto per il carico di lavoro.

Configurazione del timeout (console)

Puoi configurare il timeout della funzione nella console Lambda.

Per modificare il timeout di una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Seleziona la scheda Configurazione, quindi scegli Configurazione generale.



4. In Configurazione generale, scegli Modifica.
5. Per Timeout, imposta un valore compreso tra 1 e 900 secondi (15 minuti).
6. Scegli Save (Salva).

Configurazione del timeout (AWS CLI)

È possibile utilizzare il update-function-configurationcomando per configurare il valore di timeout, in secondi. Il comando di esempio seguente aumenta il timeout della funzione a 120 secondi (2 minuti).

Example

```
aws lambda update-function-configuration \
  --function-name my-function \
  --timeout 120
```

Configurazione del timeout (AWS SAM)

Per configurare il valore di timeout per la tua funzione, puoi usare AWS Serverless Application Model. Aggiorna la proprietà Timeout nel file `template.yaml` e poi esegui 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...
```

Configura le funzioni durevoli Lambda

Per consentire l'esecuzione duratura della funzione Lambda, è necessario configurare impostazioni specifiche che controllino per quanto tempo può essere eseguita la funzione, per quanto tempo vengono conservati i dati sullo stato e quali autorizzazioni sono richieste.

Abilita un'esecuzione duratura

Per abilitare l'esecuzione durevole della tua funzione Lambda, configurala `DurableConfig` nella configurazione della tua funzione. Questa impostazione controlla il timeout di esecuzione, il mantenimento dello stato e il comportamento di controllo delle versioni.

AWS CLI

```
aws lambda update-function-configuration \
  --function-name my-durable-function \
  --durable-config '{
    "ExecutionTimeout": 3600,
    "RetentionPeriodInDays": 30,
    "AllowInvokeLatest": true
  }'
```

CloudFormation

```
Resources:
  MyDurableFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: my-durable-function
      Runtime: nodejs18.x
      Handler: index.handler
      Code:
        ZipFile: |
          // Your durable function code
      DurableConfig:
        ExecutionTimeout: 3600
        RetentionPeriodInDays: 30
        AllowInvokeLatest: true
```

Parametri di configurazione:

- `ExecutionTimeout`- Tempo massimo di esecuzione in secondi (fino a 31.536.000 per un anno)
- `RetentionPeriodInDays`- Per quanto tempo conservare lo stato e la cronologia di esecuzione (1-365 giorni)
- `AllowInvokeLatest`- Se consentire l'invocazione della versione $LATEST per un'esecuzione duratura

Autorizzazioni IAM per funzioni durevoli

Le funzioni durevoli richiedono autorizzazioni IAM aggiuntive oltre ai ruoli di esecuzione Lambda standard. Il ruolo di esecuzione della funzione deve includere le autorizzazioni per la gestione dello stato e l'esecuzione duratura. APIs

Autorizzazioni minime richieste:

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction",
        "lambda:GetFunction",
        "lambda:ManageDurableState",
        "lambda:GetDurableExecution",
        "lambda:ListDurableExecutions"
      ],
      "Resource": "arn:aws:lambda:*:*:function:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

CloudFormation esempio di ruolo di esecuzione:

```
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/AWSLambdaBasicExecutionRole
    Policies:
      - PolicyName: DurableFunctionPolicy
        PolicyDocument:
          Version: '2012-10-17'
```

```
          Statement:
            - Effect: Allow
              Action:
                - lambda:ManageDurableState
                - lambda:GetDurableExecution
                - lambda:ListDurableExecutions
              Resource: '*'
```

Best practice per la configurazione

Segui queste best practice per configurare funzioni durevoli per l'uso in produzione:

- Imposta i timeout di esecuzione appropriati: configura `ExecutionTimeout` in base alla durata massima prevista del flusso di lavoro. Non impostate timeout inutilmente lunghi poiché influiscono sull'allocazione dei costi e delle risorse.
- Bilancia la conservazione con i costi di archiviazione: imposta in `RetentionPeriodInDays` base ai requisiti di debug e audit. Periodi di conservazione più lunghi aumentano i costi di storage.
- Usa il controllo delle versioni in produzione: imposta l'`AllowInvokeLatest`opzione `false` negli ambienti di produzione e utilizza versioni di funzioni o alias specifici per esecuzioni durature.
- Monitora le dimensioni dello stato: gli oggetti a stato di grandi dimensioni aumentano i costi di storage e possono influire sulle prestazioni. Riduci al minimo lo stato e utilizza lo storage esterno per dati di grandi dimensioni.
- Configura la registrazione appropriata: abilita la registrazione dettagliata per la risoluzione dei problemi dei flussi di lavoro a lunga durata, ma fai attenzione al volume e ai costi dei log.

Esempio di configurazione di produzione:

```
{
  "ExecutionTimeout": 86400,
  "RetentionPeriodInDays": 7,
  "AllowInvokeLatest": false
}
```

Utilizzo delle variabili di ambiente Lambda

È possibile utilizzare le variabili di ambiente per regolare il comportamento della funzione senza aggiornare il codice. Una variabile di ambiente è una coppia di stringhe archiviata nella configurazione specifica della versione di una funzione. Il runtime Lambda rende le variabili di ambiente disponibili per il codice e imposta variabili di ambiente aggiuntive che contengono informazioni sulla richiesta di funzione e invocazione.

> **Note**
>
> Per aumentare la sicurezza, consigliamo di utilizzare Gestione dei segreti AWS al posto delle variabili di ambiente per archiviare le credenziali del database e altre informazioni sensibili come chiavi API o token di autorizzazione. Per ulteriori informazioni, consulta Uso dei segreti in Secrets Manager nelle funzioni Lambda.

Le variabili di ambiente non vengono valutate prima dell'invocazione della funzione. Qualsiasi valore definito è considerato una stringa letterale e non espanso. Eseguire la valutazione della variabile nel codice funzione.

Creazione delle variabili di ambiente Lambda

Puoi configurare le variabili di ambiente in Lambda utilizzando la console Lambda, (), AWS Command Line Interface (AWS CLI) o utilizzando un AWS SAM SDK. AWS Serverless Application Model AWS

Console

Puoi definire le variabili di ambiente sulla versione non pubblicata della funzione. Quando pubblichi una versione, le variabili di ambiente sono bloccate per quella versione insieme ad altre impostazioni di configurazioni specifiche della versione.

È possibile creare una variabile di ambiente per la funzione definendo una chiave e un valore. La funzione utilizza il nome della chiave per recuperare il valore della variabile di ambiente.

Per impostare le variabili di ambiente nella console Lambda

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegli Configurazione, quindi scegli Variabili di ambiente.

4. In Environment variables (Variabili di ambiente), scegliere Edit (Modifica).
5. Scegli Add environment variable (Aggiungi variabile d'ambiente).
6. Inserisci una coppia chiave valore.

   Requisiti

   - Le chiavi iniziano con una lettera e sono di almeno due caratteri.
   - Le chiavi contengono solo lettere, numeri e il carattere di sottolineatura (_).
   - Le chiavi non sono riservate da Lambda.
   - La dimensione totale di tutte le variabili di ambiente non supera i 4 KB.
7. Scegli Save (Salva).

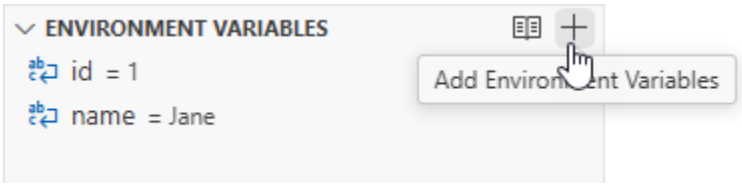
Generazione di un elenco di variabili di ambiente nell'editor di codice della console

Puoi generare un elenco di variabili di ambiente nell'editor di codice Lambda. È un modo rapido per fare riferimento alle variabili di ambiente durante la scrittura del codice.

1. Scegli la scheda Codice.
2. Scorri verso il basso fino alla sezione VARIABILI DI AMBIENTE dell'editor di codice. Le variabili di ambiente esistenti sono elencate qui:



3. Per creare nuove variabili di ambiente, scegli il segno più (+):



Le variabili di ambiente rimangono crittografate quando sono elencate nell'editor di codice della console. Se hai abilitato gli helper di crittografia per la crittografia in transito, tali impostazioni rimangono invariate. Per ulteriori informazioni, consulta Protezione delle variabili di ambiente Lambda.

L'elenco delle variabili di ambiente è di sola lettura ed è disponibile solo nella console Lambda. Questo file non è incluso quando scarichi l'archivio di file .zip della funzione e non è possibile aggiungere variabili di ambiente caricando questo file.

AWS CLI

L'esempio seguente imposta due variabili di ambiente in una funzione denominata `my-function`.

```
aws lambda update-function-configuration \
  --function-name my-function \
  --environment "Variables={BUCKET=amzn-s3-demo-bucket,KEY=file.txt}"
```

Quando applichi le variabili di ambiente con il comando `update-function-configuration`, viene sostituito l'intero contenuto della struttura `Variables`. Per mantenere le variabili di ambiente esistenti quando ne aggiungi una nuova, includi tutti i valori esistenti nella richiesta.

Per ottenere la configurazione corrente, utilizza il comando `get-function-configuration`.

```
aws lambda get-function-configuration \
  --function-name my-function
```

Verrà visualizzato l'output seguente:

```
{
    "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",
    ...
}
```

È possibile passare l'ID di revisione dall'output di `get-function-configuration` come parametro a `update-function-configuration`. Ciò garantisce che i valori non cambino da quando leggi la configurazione a quando la aggiorni.

Per configurare la chiave di crittografia di una funzione, imposta l'opzione `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

Per configurare le variabili di ambiente per la tua funzione, puoi utilizzare AWS Serverless Application Model. Aggiorna le proprietà Environment e Variables nel tuo file `template.yaml`, quindi esegui 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

Per gestire le variabili di ambiente utilizzando un AWS SDK, utilizzate le seguenti operazioni API.

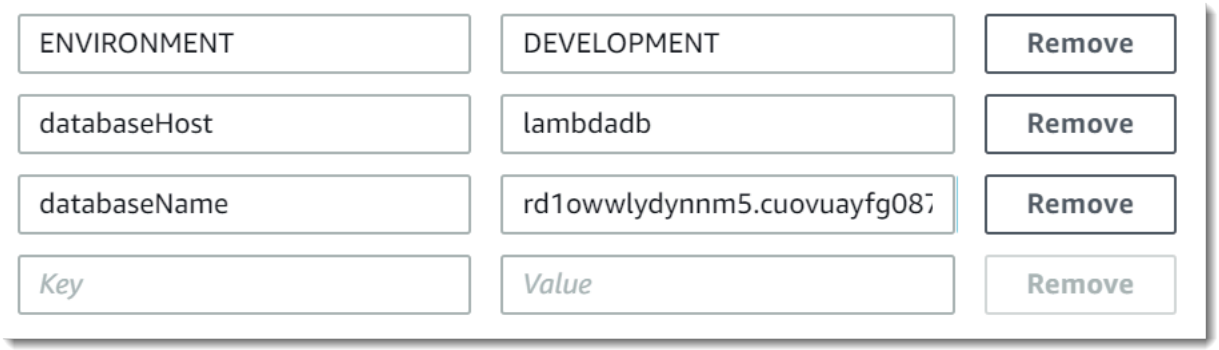
- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Per ulteriori informazioni, consulta la documentazione dell'SDK AWS relativa al linguaggio di programmazione preferito.

Scenario di esempio per le variabili di ambiente

È possibile utilizzare le variabili di ambiente per personalizzare il comportamento delle funzioni nell'ambiente di test e nell'ambiente di produzione. Ad esempio, puoi creare due funzioni con stesso codice ma configurazione diversa. Una funzione si connette a un database di test e l'altra a un database di produzione. In questo caso, puoi utilizzare le variabili di ambiente per passare alla funzione il nome host e altri dettagli di connessione per il database.

Nell'esempio seguente viene illustrato come definire l'host del database e il nome del database come variabili di ambiente.



Se si desidera che l'ambiente di test generi più informazioni di debug rispetto all'ambiente di produzione, è possibile impostare una variabile di ambiente per configurare l'ambiente di test in modo da utilizzare una registrazione più dettagliata o una traccia più dettagliata.

Ad esempio, nell'ambiente di test, è possibile impostare una variabile di ambiente con la chiave `LOG_LEVEL` e un valore che indica un livello di log di debug o trace. Nel codice della funzione Lambda, puoi quindi utilizzare questa variabile di ambiente per impostare il livello di log.

I seguenti esempi di codice in Python e Node.js illustrano come è possibile raggiungere questo obiettivo. Questi esempi presuppongono che la variabile di ambiente abbia un valore `DEBUG` in Python o `debug` in Node.js.

Python

Example Codice Python per impostare il livello di 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 Codice Node.js per impostare il livello di log

Questo esempio utilizza la libreria di registrazione `winston`. Usa npm per aggiungere questa libreria al pacchetto di implementazione della funzione. Per ulteriori informazioni, consulta the section called “Creazione di un pacchetto di implementazione .zip con dipendenze”.

```
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');

};
```

Recupero delle variabili di ambiente Lambda

Per recuperare le variabili di ambiente nel codice della funzione, utilizza il metodo standard per il linguaggio di programmazione.

Node.js

```
let region = process.env.AWS_REGION
```

Python

```
import os
  region = os.environ['AWS_REGION']
```

> **Note**
>
> In alcuni casi, potrebbe essere necessario utilizzare il seguente formato:
>
> ```
> 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 memorizza le variabili di ambiente in modo sicuro crittografandole quando inattive. Puoi configurare Lambda per utilizzare una chiave di crittografia diversa, crittografare i valori delle variabili di ambiente sul lato client o impostare le variabili di ambiente in un CloudFormation modello con. Gestione dei segreti AWS

Variabili di ambiente di runtime definite

I tempi di esecuzione Lambda impostano diverse variabili di ambiente durante l'inizializzazione. La maggior parte delle variabili di ambiente fornisce informazioni sulla funzione o sul runtime. Le chiavi per queste variabili di ambiente sono riservate e non possono essere impostate nella configurazione della funzione.

Variabili d'ambiente riservate

- `_HANDLER`: la posizione del gestore configurata nella funzione.
- `_X_AMZN_TRACE_ID`: l'intestazione di traccia X-Ray. Questa variabile di ambiente cambia a ogni invocazione.
  - Questa variabile di ambiente non è definita per i runtime solo per il sistema operativo (la famiglia di runtime `provided`). Puoi impostare `_X_AMZN_TRACE_ID` per runtime personalizzati utilizzando l'intestazione di risposta `Lambda-Runtime-Trace-Id` dal Chiamata successiva.
  - Per le versioni di runtime Java 17 e successive, questa variabile di ambiente non viene utilizzata. Lambda archivia invece le informazioni di tracciamento nella proprietà di sistema `com.amazonaws.xray.traceHeader`.
- `AWS_DEFAULT_REGION`— L'impostazione predefinita Regione AWS in cui viene eseguita la funzione Lambda.

- `AWS_REGION`— Il Regione AWS luogo in cui viene eseguita la funzione Lambda. Se immesso, questo valore sovrascrive `AWS_DEFAULT_REGION`.
  - Per ulteriori informazioni sull'utilizzo delle variabili di Regione AWS ambiente con AWS SDKs, vedete AWS Region nella AWS SDKs and Tools Reference Guide.
- `AWS_EXECUTION_ENV`: l'identificatore di runtime, preceduto da `AWS_Lambda_`, ad esempio `AWS_Lambda_java8`. Questa variabile di ambiente non è definita per i runtime solo per il sistema operativo (la famiglia di runtime `provided`).
- `AWS_LAMBDA_FUNCTION_NAME`: il nome della funzione.
- `AWS_LAMBDA_FUNCTION_MEMORY_SIZE`: la quantità di memoria disponibile per la funzione in MB.
- `AWS_LAMBDA_FUNCTION_VERSION`: la versione della funzione in esecuzione.
- `AWS_LAMBDA_INITIALIZATION_TYPE`— Il tipo di inizializzazione della funzione, che è`on-demand`, `provisioned-concurrency`o `snap-start`, `lambda-managed-instances`. Per informazioni, vedere Configurazione della concorrenza fornita, o. Migliorare le prestazioni di avvio con Lambda SnapStart Istanze gestite Lambda
- `AWS_LAMBDA_LOG_GROUP_NAME`, `AWS_LAMBDA_LOG_STREAM_NAME` — Il nome del gruppo e dello stream Amazon CloudWatch Logs per la funzione. Le variabili di `AWS_LAMBDA_LOG_STREAM_NAME` ambiente `AWS_LAMBDA_LOG_GROUP_NAME` e non sono disponibili nelle funzioni Lambda SnapStart .
- `AWS_ACCESS_KEY`, `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN`: le chiavi di accesso ottenute dal ruolo di esecuzione della funzione.
- `AWS_LAMBDA_RUNTIME_API`: (Runtime personalizzato) L'host e la porta dell'API di runtime.
- `LAMBDA_TASK_ROOT`: il percorso del codice della funzione Lambda.
- `LAMBDA_RUNTIME_DIR`: il percorso delle librerie di runtime.
- `AWS_LAMBDA_MAX_CONCURRENCY`— (Solo istanze gestite Lambda) Il numero massimo di chiamate simultanee che Lambda invierà a un ambiente di esecuzione.

Le seguenti variabili di ambiente aggiuntive non sono riservate e possono essere estese nella configurazione della funzione.

Variabili d'ambiente non riservate

- `LANG` – Le impostazioni locali del runtime (`en_US.UTF-8`).
- `PATH`: il percorso di esecuzione (`/usr/local/bin:/usr/bin/:/bin:/opt/bin`).

- `LD_LIBRARY_PATH`: il percorso della libreria di sistema (`/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) Il percorso della libreria Node.js (`/opt/nodejs/node12/node_modules/:/opt/nodejs/node_modules:$LAMBDA_RUNTIME_DIR/node_modules`).
- `NODE_OPTIONS`: (Node.js) per i runtime Node.js, puoi utilizzare `NODE_OPTIONS` per riattivare le funzionalità sperimentali che Lambda disabilita per impostazione predefinita.
- `PYTHONPATH`: (Python) il percorso della libreria Python (`$LAMBDA_RUNTIME_DIR`).
- `GEM_PATH` – (Ruby) Il percorso della libreria Ruby (`$LAMBDA_TASK_ROOT/vendor/bundle/ruby/3.3.0:/opt/ruby/gems/3.3.0`).
- `AWS_XRAY_CONTEXT_MISSING` – Per il tracciamento X-Ray, Lambda imposta questo valore su `LOG_ERROR` per evitare di generare errori di runtime dall'SDK X-Ray.
- `AWS_XRAY_DAEMON_ADDRESS` – Per il tracciamento X-Ray, l'indirizzo IP e la porta del daemon X-Ray.
- `AWS_LAMBDA_DOTNET_PREJIT`: (.NET) imposta questa variabile per abilitare o disabilitare le ottimizzazioni di runtime specifiche di .NET. I valori includono `always`, `never` e `provisioned-concurrency`. Per ulteriori informazioni, consulta Configurazione della simultaneità fornita per una funzione.
- `TZ`: il fuso orario dell'ambiente (`:UTC`). L'ambiente di esecuzione utilizza NTP per sincronizzare l'orologio di sistema.

I valori di esempio mostrati riflettono i runtime più recenti. La presenza di variabili specifiche o dei loro valori può variare nei runtime precedenti.

Protezione delle variabili di ambiente Lambda

Per proteggere le variabili di ambiente, è possibile utilizzare la crittografia lato server per proteggere i dati inattivi e la crittografia lato client per proteggere i dati in transito.

> **Note**
>
> Per aumentare la sicurezza del database, si consiglia di utilizzare al Gestione dei segreti AWS posto delle variabili di ambiente per memorizzare le credenziali del database. Per ulteriori informazioni, consulta Uso dei segreti in Secrets Manager nelle funzioni Lambda.

Sicurezza dei dati inattivi

Lambda fornisce sempre la crittografia lato server dei dati inattivi con un AWS KMS key. Per impostazione predefinita, Lambda utilizza un Chiave gestita da AWS. Se questo comportamento predefinito si adatta al flusso di lavoro, non è necessario impostare altro. Lambda li crea Chiave gestita da AWS nel tuo account e gestisce le autorizzazioni per te. AWS non ti addebita alcun costo per l'utilizzo di questa chiave.

Se preferisci, puoi invece fornire una chiave gestita AWS KMS dal cliente. È possibile eseguire questa operazione per avere il controllo sulla rotazione della chiave KMS o per soddisfare i requisiti dell'organizzazione per la gestione delle chiavi KMS. Quando si utilizza la chiave gestita dal cliente, solo gli utenti del tuo account con accesso alla chiave possono visualizzare o gestire le variabili di ambiente sulla funzione.

Le chiavi gestite dal cliente sono soggette a costi standard AWS KMS . Per ulteriori informazioni, consultare Prezzi di AWS Key Management Service.

Sicurezza in transito

Per una maggiore sicurezza, è possibile abilitare gli helper per la crittografia in transito, assicurando così che le variabili di ambiente siano crittografate lato client per la protezione in transito.

Per configurare la crittografia per le variabili di ambiente

1. Usa AWS Key Management Service (AWS KMS) per creare qualsiasi chiave gestita dal cliente che Lambda possa utilizzare per la crittografia lato server e lato client. Per ulteriori informazioni, consulta Creazione di chiavi nella AWS Key Management Service Guida per gli sviluppatori di .
2. Utilizzando la console Lambda, accedere alla pagina Edit environment variables (Modifica delle variabili di ambiente).
    a. Aprire la pagina Funzioni della console Lambda.
    b. Scegliere una funzione.
    c. Scegliere Configurazione, quindi scegliere Variabili di ambiente nella barra di navigazione sinistra.
    d. Nella sezione Variabili di ambiente, scegliere Modifica.
    e. Espandere Encryption configuration (Configurazione della crittografia).
3. (Facoltativo) Abilita gli helper di crittografia della console per utilizzare la crittografia lato client per proteggere i dati in transito.

a. In Crittografia in transito, scegliere Enable helpers for encryption in transit (Abilita helper per la crittografia in transito).

b. Per ogni variabile di ambiente per cui si desidera abilitare gli helper di crittografia della console, scegliere Encrypt (Crittografia) accanto alla variabile di ambiente.

c. In AWS KMS key Per crittografare in transito, scegli una chiave gestita dal cliente che hai creato all'inizio di questa procedura.

d. Scegliere Execution role policy (Policy del ruolo di esecuzione) e copiare la policy. Questa policy concede l'autorizzazione al ruolo di esecuzione della funzione per decrittare le variabili di ambiente.

   Salvare la policy da utilizzare nell'ultima fase di questa procedura.

e. Aggiungi il codice alla funzione che decritta le variabili di ambiente. Per visualizzare un esempio, scegli Decritta frammento di segreto.

4. (Facoltativo) Specifica la chiave gestita dal cliente per la crittografia dei dati inattivi.

   a. Scegliere Use a customer master key (Utilizza una chiave master del cliente).

   b. Scegliere una chiave gestita dal cliente creata all'inizio di questa procedura.

5. Scegli Save (Salva).

6. Impostare le autorizzazioni.

   Se stai utilizzando una chiave gestita dal cliente con crittografia lato server, concedi le autorizzazioni a qualsiasi utente o ruolo che desideri possa visualizzare o gestire le variabili di ambiente sulla funzione. Per ulteriori informazioni, consulta Gestione delle autorizzazioni per la chiave KMS di crittografia lato server.

   Se si abilita la crittografia lato client per la sicurezza in transito, la funzione richiede l'autorizzazione per chiamare l'operazione API `kms:Decrypt`. Aggiungere la policy salvata in precedenza in questa procedura al ruolo di esecuzione della funzione.

Gestione delle autorizzazioni per la chiave KMS di crittografia lato server

Non sono necessarie AWS KMS autorizzazioni affinché l'utente o il ruolo di esecuzione della funzione utilizzino la chiave di crittografia predefinita. Per utilizzare una chiave gestita dal cliente, occorre l'autorizzazione all'utilizzo della chiave. Lambda utilizza queste autorizzazioni per creare una concessione sulla chiave. Questo consente a Lambda di usarla per la crittografia.

- `kms:ListAliases` – Per visualizzare i tasti nella console Lambda.
- `kms:CreateGrant`, `kms:Encrypt` – Per configurare una chiave gestita dal cliente su una funzione.
- `kms:Decrypt` – Per visualizzare e gestire le variabili di ambiente crittografate con la chiave gestita dal cliente.

Puoi ottenere queste autorizzazioni dalla tua politica di autorizzazioni basata sulle risorse Account AWS o sulla politica delle autorizzazioni basata sulle risorse di una chiave. `ListAliases`è fornito dalle policy gestite per Lambda. Le policy della chiave concedono le autorizzazioni rimanenti agli utenti del gruppo Key users (Utenti chiave).

Gli utenti senza autorizzazioni `Decrypt` possono comunque gestire le funzioni, ma non possono visualizzare le variabili di ambiente o gestirle nella console Lambda. Per impedire a un utente di visualizzare le variabili di ambiente, aggiungere un'istruzione alle autorizzazioni dell'utente che nega l'accesso alla chiave predefinita, a una chiave gestita dal cliente o a tutte le chiavi.

Example Policy IAM: negano l'accesso in base all'ARN della chiave

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"
        }
    ]
}
```

Per informazioni dettagliate sulla gestione delle autorizzazioni delle chiavi, consulta la pagina Utilizzo delle policy delle chiavi in AWS KMS nella Guida per gli sviluppatori di AWS Key Management Service .

Consentire alle funzioni Lambda l'accesso alle risorse in un Amazon VPC

Con Amazon Virtual Private Cloud (Amazon VPC), puoi creare reti private Account AWS per ospitare risorse come istanze Amazon Elastic Compute Cloud (Amazon EC2), istanze Amazon Relational Database Service (Amazon RDS) e istanze Amazon. ElastiCache Puoi consentire alla tua funzione Lambda di accedere alle risorse ospitate in un Amazon VPC collegando la funzione al VPC tramite le sottoreti private che contengono le risorse. Segui le istruzioni nelle seguenti sezioni per collegare una funzione Lambda a un Amazon VPC utilizzando la console Lambda, il () o. AWS Command Line Interface AWS CLI AWS SAM

> **Note**
>
> Ogni funzione Lambda viene eseguita all'interno di un VPC di proprietà e gestito dal servizio Lambda. Questi VPCs vengono gestiti automaticamente da Lambda e non sono visibili ai clienti. La configurazione della funzione per accedere ad altre AWS risorse in un Amazon VPC non ha alcun effetto sul VPC gestito da Lambda all'interno del quale viene eseguita la funzione.

Sections

- Autorizzazioni IAM richieste
- Collegamento di funzioni Lambda a un Amazon VPC nel tuo Account AWS
- Accesso a Internet se collegato a un VPC
- IPv6 supporto
- Le migliori pratiche per l'utilizzo di Lambda con Amazon VPCs
- Comprensione delle interfacce di rete elastiche Hyperplane () ENIs
- Utilizzo dei tasti di condizione IAM per le impostazioni VPC
- Tutorial VPC

Autorizzazioni IAM richieste

Per collegare una funzione Lambda a un Amazon VPC nel tuo, Account AWS Lambda necessita delle autorizzazioni per creare e gestire le interfacce di rete che utilizza per consentire alla funzione di accedere alle risorse del VPC.

Le interfacce di rete create da Lambda sono note come Hyperplane Elastic Network Interfaces o Hyperplane. ENIs Per ulteriori informazioni su queste interfacce di rete, consulta the section called “Comprensione delle interfacce di rete elastiche Hyperplane () ENIs”.

Puoi concedere alla tua funzione le autorizzazioni necessarie allegando la policy AWS AWSLambdaVPCAccessExecutionRolegestita al ruolo di esecuzione della funzione. Quando crei una nuova funzione nella console Lambda e la colleghi a un VPC, Lambda aggiunge automaticamente questa policy di autorizzazioni per tuo conto.

Se preferisci creare la tua policy di autorizzazioni IAM, assicurati di aggiungere tutte le seguenti autorizzazioni e di consentirle su tutt le risorse (`"Resource": "*"`):

- ec2: CreateNetworkInterface
- ec2: DescribeNetworkInterfaces
- ec2: DescribeSubnets
- ec2: DeleteNetworkInterface
- ec2: AssignPrivateIpAddresses
- ec2: UnassignPrivateIpAddresses

Tieni presente che il ruolo della tua funzione necessita di queste autorizzazioni solo per creare le interfacce di rete, non per richiamare la tua funzione. Puoi comunque richiamare correttamente la funzione quando è collegata a un Amazon VPC, anche se si rimuovono queste autorizzazioni dal ruolo di esecuzione della funzione.

Per collegare la tua funzione a un VPC, Lambda deve anche verificare le risorse di rete utilizzando il tuo ruolo utente IAM. Assicurati che il tuo ruolo utente disponga delle seguenti autorizzazioni IAM:

- ec2: DescribeSecurityGroups
- ec2: DescribeSubnets
- ec2: DescribeVpcs
- ec2: GetSecurityGroupsForVpc

> **Note**
>
> Le EC2 autorizzazioni Amazon concesse al ruolo di esecuzione della funzione vengono utilizzate dal servizio Lambda per collegare la funzione a un VPC. Tuttavia, stai anche

concedendo implicitamente queste autorizzazioni al codice della tua funzione. Ciò significa che il codice della funzione è in grado di effettuare queste chiamate EC2 API Amazon. Per ulteriori informazioni su come seguire la best practice di sicurezza, consulta the section called "Best practice di sicurezza".

Collegamento di funzioni Lambda a un Amazon VPC nel tuo Account AWS

Collega la tua funzione a un Amazon VPC del tuo dispositivo Account AWS utilizzando la console Lambda, oppure. AWS CLI AWS SAM Se utilizzi AWS CLI o o AWS SAM colleghi una funzione esistente a un VPC utilizzando la console Lambda, assicurati che il ruolo di esecuzione della funzione disponga delle autorizzazioni necessarie elencate nella sezione precedente.

Le funzioni Lambda non possono connettersi direttamente a un VPC con la tenancy dell'istanza dedicata. Per connetterti alle risorse in un VPC dedicato, esegui il peering a un secondo VPC con la tenancy predefinita.

Lambda console

Per collegare una funzione a un Amazon VPC al momento della creazione

1. Apri la pagina Funzioni della console Lambda e scegli Crea funzione.
2. In Informazioni di base, immettere un nome per la funzione in Nome funzione.
3. Configura le impostazioni del VPC per la funzione effettuando le seguenti operazioni:
    a. Espandere Advanced settings (Impostazioni avanzate).
    b. Seleziona Abilita VPC, quindi scegli il VPC a cui desideri collegare la funzione.
    c. (Facoltativo) Per consentire il traffico in uscita, seleziona Consenti IPv6 il traffico per sottoreti dual-stack IPv6 .
    d. Scegli le sottoreti e i gruppi di sicurezza per cui creare l'interfaccia di rete. Se hai selezionato Consenti il IPv6 traffico per le sottoreti dual-stack, tutte le sottoreti selezionate devono avere un blocco CIDR e un blocco CIDR. IPv4 IPv6

> **Note**
>
> Connetti la tua funzione a sottoreti private per accedere alle risorse private. Se la tua funzione ha bisogno dell'accesso a Internet, consulta the section called

"Accesso Internet per funzioni del VPC". La connessione di una funzione a una sottorete pubblica non fornisce l'accesso a Internet o a un indirizzo IP pubblico.

4. Scegli Crea funzione.

Per collegare una funzione esistente a un Amazon VPC

1. Apri la pagina Funzioni della console Lambda e scegli la tua funzione.
2. Scegli la scheda Configurazione, quindi scegli VPC.
3. Scegli Modifica.
4. In VPC, seleziona l'Amazon VPC a cui desideri collegare la funzione.
5. (Facoltativo) Per consentire il traffico in uscita, seleziona Consenti il traffico per sottoreti dual-stack IPv6 . IPv6
6. Scegli le sottoreti e i gruppi di sicurezza per cui creare l'interfaccia di rete. Se hai selezionato Consenti il IPv6 traffico per le sottoreti dual-stack, tutte le sottoreti selezionate devono avere un blocco CIDR e un blocco CIDR. IPv4 IPv6

> **Note**
>
> Connetti la tua funzione a sottoreti private per accedere alle risorse private. Se la tua funzione ha bisogno dell'accesso a Internet, consulta the section called "Accesso Internet per funzioni del VPC". La connessione di una funzione a una sottorete pubblica non fornisce l'accesso a Internet o a un indirizzo IP pubblico.

7. Scegli Save (Salva).

AWS CLI

Per collegare una funzione a un Amazon VPC al momento della creazione

- Per creare una funzione Lambda e collegarla a un VPC, esegui il comando della CLI `create-function` seguente.

```
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
```

Specifica le tue sottoreti e i tuoi gruppi di sicurezza e imposta `Ipv6AllowedForDualStack` su `true` o `false` in base al tuo caso d'uso.

Per collegare una funzione esistente a un Amazon VPC

- Per collegare una funzione esistente a un VPC, esegui il comando della CLI `update-function-configuration` seguente.

```
aws lambda update-function-configuration --function-name my-function \
--vpc-config Ipv6AllowedForDualStack=true,
 SubnetIds=subnet-071f712345678e7c8,subnet-07fd123456788a036,SecurityGroupIds=sg-085912
```

Per scollegare la tua funzione da un VPC

- Per scollegare la funzione da un VPC, esegui il comando della CLI `update-function-configuration` seguente con un elenco vuoto di sottoreti VPC e gruppi di sicurezza.

```
aws lambda update-function-configuration --function-name my-function \
--vpc-config SubnetIds=[],SecurityGroupIds=[]
```

AWS SAM

Per collegare la tua funzione a un VPC

- Per collegare una funzione Lambda a un Amazon VPC, aggiungi la proprietà `VpcConfig` alla definizione della funzione come mostrato nel seguente modello di esempio. Per ulteriori informazioni su questa proprietà, vedere AWS: :Lambda: :Function VpcConfig nella Guida per l'CloudFormation utente (la AWS SAM `VpcConfig` proprietà viene passata direttamente alla `VpcConfig` proprietà di una CloudFormation `AWS::Lambda::Function` risorsa).

```
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
```

Per ulteriori informazioni sulla configurazione del tuo VPC AWS SAM in, AWS consulta::EC2: :VPC nella Guida per l'utente.CloudFormation

Accesso a Internet se collegato a un VPC

Per impostazione predefinita, le funzioni Lambda hanno accesso a Internet pubblico. Quando si collega la funzione a un VPC, può accedere solo alle risorse disponibili all'interno di tale VPC. Per concedere alla funzione l'accesso a Internet, è inoltre necessario configurare il VPC per l'accesso a Internet. Per ulteriori informazioni, consulta the section called “Accesso Internet per funzioni del VPC”.

IPv6 supporto

La tua funzione può connettersi alle risorse nelle sottoreti VPC dual-stack tramite. IPv6 Per impostazione predefinita, questa opzione è disabilitata. Per consentire il IPv6 traffico in uscita, usa la console o l'opzione con la funzione o il `--vpc-config Ipv6AllowedForDualStack=true` comando create-. update-function-configuration

> **Note**
>
> Per consentire il IPv6 traffico in uscita in un VPC, tutte le sottoreti connesse alla funzione devono essere sottoreti dual-stack. Lambda non supporta IPv6 connessioni in uscita per IPv6 sole sottoreti in un VPC o connessioni in IPv6 uscita per funzioni che non sono connesse a un VPC.

Puoi aggiornare il codice della funzione per connetterti in modo esplicito alle risorse della sottorete. IPv6 Il seguente esempio di Python apre un socket e si connette a un IPv6 server.

Example — Connect al IPv6 server

```
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()
```

Le migliori pratiche per l'utilizzo di Lambda con Amazon VPCs

Per garantire che la configurazione del VPC Lambda soddisfi le linee guida delle best practice, segui i consigli nelle sezioni seguenti.

Best practice di sicurezza

Per collegare la tua funzione Lambda a un VPC, devi assegnare al ruolo di esecuzione della funzione una serie di autorizzazioni Amazon. EC2 Queste autorizzazioni sono necessarie per creare le interfacce di rete utilizzate dalla funzione per accedere alle risorse nel VPC. Tuttavia, stai anche concedendo implicitamente queste autorizzazioni al codice della tua funzione. Ciò significa che il codice della funzione è autorizzato a effettuare queste chiamate EC2 API Amazon.

Per seguire il principio dell'accesso con privilegio minimo, aggiungi una policy di negazione come quella riportata nell'esempio seguente al ruolo di esecuzione della funzione. Questa policy impedisce al codice della funzione di effettuare chiamate ad Amazon EC2 APIs, pur consentendo al servizio Lambda di gestire le risorse VPC per tuo conto. La policy utilizza la chiave di condizione `lambda:SourceFunctionArn`, che si applica solo alle chiamate API effettuate dal codice della funzione durante l'esecuzione. Per ulteriori informazioni, consulta the section called “ARN della funzione di origine”.

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 fornisce gruppi di sicurezza e liste di controllo degli accessi alla rete (ACLs) per aumentare la sicurezza nel tuo VPC. I gruppi di sicurezza controllano il traffico in entrata e in uscita per le risorse e la rete ACLs controllano il traffico in entrata e in uscita per le sottoreti. I gruppi di sicurezza forniscono un controllo di accesso sufficiente per la maggior parte delle sottoreti. Puoi usare la rete ACLs se desideri un ulteriore livello di sicurezza per il tuo VPC. Per linee guida generali sulle best practice di sicurezza per l'utilizzo di Amazon VPCs, consulta le best practice di sicurezza per il tuo VPC nella Amazon Virtual Private Cloud User Guide.

Best practice sulle prestazioni

Quando colleghi la tua funzione a un VPC, Lambda verifica se esiste una risorsa di rete disponibile (Hyperplane ENI) a cui può connettersi. Gli Hyperplane ENIs sono associati a una particolare combinazione di gruppi di sicurezza e sottoreti VPC. Se hai già collegato una funzione a un VPC, specificare le stesse sottoreti e gli stessi gruppi di sicurezza quando colleghi un'altra funzione implica che Lambda può condividere le risorse di rete ed evitare la necessità di creare una nuova Hyperplane ENI. Per ulteriori informazioni su Hyperplane ENIs e il relativo ciclo di vita, vedere. the section called “Comprensione delle interfacce di rete elastiche Hyperplane () ENIs”

Comprensione delle interfacce di rete elastiche Hyperplane () ENIs

Una Hyperplane ENI è una risorsa gestita che funge da interfaccia di rete tra la tua funzione Lambda e le risorse a cui desideri che la tua funzione si connetta. Il servizio Lambda li crea e li gestisce ENIs automaticamente quando colleghi la funzione a un VPC.

Gli Hyperplane non ENIs sono visibili direttamente all'utente e non è necessario configurarli o gestirli. Tuttavia, conoscerne il funzionamento può aiutarti a comprendere il comportamento della tua funzione quando la colleghi a un VPC.

La prima volta che si collega una funzione a un VPC utilizzando una particolare combinazione di sottorete e gruppo di sicurezza, Lambda crea una Hyperplane ENI. Anche altre funzioni del tuo account che utilizzano la stessa combinazione di sottorete e gruppo di sicurezza possono utilizzare questa ENI. Ove possibile, Lambda riutilizza le risorse esistenti ENIs per ottimizzare l'utilizzo delle risorse e ridurre al minimo la creazione di nuove. ENIs Ogni Hyperplane ENI supporta fino a 65.000 connessioni/porte. Se il numero di connessioni supera questo limite, Lambda ridimensiona automaticamente il numero in base al traffico ENIs di rete e ai requisiti di concorrenza.

Per le nuove funzioni, mentre Lambda crea una Hyperplane ENI, la funzione rimane nello stato In sospeso e non è possibile richiamarla. La funzione passa allo stato Attivo solo quando l'Hyperplane ENI è pronta, il che può richiedere diversi minuti. Per le funzioni esistenti, non è possibile eseguire ulteriori operazioni per tale funzione, ad esempio la creazione di versioni o l'aggiornamento del codice della funzione, ma è possibile continuare a richiamare le sue versioni precedenti.

Come parte della gestione del ciclo di vita ENI, Lambda può eliminare e ricreare ENIs per bilanciare il carico del traffico di rete ENIs o per risolvere problemi riscontrati nei controlli sanitari ENI. Inoltre, se una funzione Lambda rimane inattiva per 14 giorni, Lambda recupera qualsiasi ENIs iperpiano inutilizzato e imposta lo stato della funzione su. `Inactive` L'invocazione successiva avrà esito negativo e la funzione entrerà di nuovo nello stato in sospeso fino a quando Lambda non completa la creazione o l'allocazione di una Hyperplane ENI. Ti consigliamo di non basare il tuo design sulla persistenza di. ENIs

Quando aggiorni una funzione per rimuoverne la configurazione VPC, Lambda richiede fino a 20 minuti per eliminare l'ENI Hyperplane allegato. Lambda elimina l'ENI solo se nessun'altra funzione (o versione della funzione pubblicata) utilizza l'Hyperplane ENI.

Per eliminare l'Hyperplane ENI, Lambda si basa sulle autorizzazioni del ruolo di esecuzione della funzione. Se elimini il ruolo di esecuzione prima che Lambda abbia eliminato l'Hyperplane

ENI, Lambda non sarà più in grado di eliminare l'Hyperplane ENI. Puoi eseguire manualmente l'eliminazione.

Utilizzo dei tasti di condizione IAM per le impostazioni VPC

È possibile utilizzare chiavi di condizione Lambda specifiche per le impostazioni VPC per fornire controlli di autorizzazione aggiuntivi per le funzioni Lambda. Ad esempio, è possibile richiedere che tutte le funzioni dell'organizzazione siano connesse a un VPC. È inoltre possibile specificare le sottoreti e i gruppi di sicurezza che gli utenti della funzione possono e non possono utilizzare.

Lambda supporta inoltre le seguenti chiavi di condizione nelle policy IAM:

- lambda: VpcIds — Consenti o nega uno o più. VPCs
- lambda: SubnetIds — Consenti o nega una o più sottoreti.
- lambda: SecurityGroupIds — Consenti o nega uno o più gruppi di sicurezza.

L'API Lambda opera CreateFunctione UpdateFunctionConfigurationsupporta queste chiavi di condizione. Per ulteriori informazioni sull'uso delle chiavi di condizione nelle policy IAM, consulta elementi della policy IAM JSON: condizione nella Guida per l'utente di IAM.

> **Tip**
>
> Se la funzione include già una configurazione VPC da una richiesta API precedente, è possibile inviare una richiesta `UpdateFunctionConfiguration` senza la configurazione VPC.

Policy di esempio con chiavi di condizione per le impostazioni VPC

Negli esempi seguenti viene illustrato come utilizzare le chiavi di condizione per le impostazioni VPC. Dopo aver creato un'istruzione delle policy con le restrizioni desiderate, aggiungere l'istruzione delle policy per l'utente o il ruolo di destinazione.

Assicurarsi che gli utenti distribuiscano solo funzioni connesse a VPC

Per garantire che tutti gli utenti distribuiscano solo funzioni connesse a VPC, è possibile negare le operazioni di creazione e aggiornamento delle funzioni che non includono un ID VPC valido.

Si noti che l'ID VPC non è un parametro di input per la richiesta `CreateFunction` o `UpdateFunctionConfiguration`. Lambda recupera il valore dell'ID VPC in base ai parametri della sottorete e del gruppo di sicurezza.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "EnforceVPCFunction",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "Null": {
           "lambda:VpcIds": "true"
        }
      }
    }
  ]
}
```

Nega agli utenti l'accesso a sottoreti o gruppi di sicurezza specifici VPCs

Per negare agli utenti l'accesso a informazioni specifiche VPCs, usa `StringEquals` per verificare il valore della condizione. `lambda:VpcIds` Nell'esempio seguente viene negato agli utenti l'accesso a `vpc-1` e `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"
                    ]
                }
            }
        }
    ]
}
```

Per negare agli utenti l'accesso a subnet specifiche, utilizzare `StringEquals` per verificare il valore della condizione `lambda:SubnetIds`. Nell'esempio seguente viene negato agli utenti l'accesso a `subnet-1` e `subnet-2`.

```
{
      "Sid": "EnforceOutOfSubnet",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
            "lambda:SubnetIds": ["subnet-1", "subnet-2"]
        }
      }
    }
```

Per negare agli utenti l'accesso a specifici gruppi di sicurezza, utilizzare `StringEquals` per verificare il valore della condizione `lambda:SecurityGroupIds`. Nell'esempio seguente viene negato agli utenti l'accesso a `sg-1` e `sg-2`.

```
{
      "Sid": "EnforceOutOfSecurityGroups",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Deny",
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
            "lambda:SecurityGroupIds": ["sg-1", "sg-2"]
        }
      }
    }
  ]
}
```

Consenti agli utenti di creare e aggiornare funzioni con impostazioni VPC specifiche

Per consentire agli utenti di accedere a condizioni specifiche VPCs, utilizza `StringEquals` per verificare il valore della `lambda:VpcIds` condizione. L'esempio seguente consente agli utenti di accedere a `vpc-1` e `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"
```

```
                    ]
                }
            }
        }
    ]
}
```

Per consentire agli utenti di accedere a sottoreti specifiche, utilizzare `StringEquals` per verificare il valore della condizione `lambda:SubnetIds`. L'esempio seguente consente agli utenti di accedere a `subnet-1` e `subnet-2`.

```
{
      "Sid": "EnforceStayInSpecificSubnets",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
            "lambda:SubnetIds": ["subnet-1", "subnet-2"]
        }
      }
    }
```

Per consentire agli utenti di accedere a gruppi di sicurezza specifici, utilizzare `StringEquals` per verificare il valore della condizione `lambda:SecurityGroupIds`. L'esempio seguente consente agli utenti di accedere a `sg-1` e `sg-2`.

```
{
      "Sid": "EnforceStayInSpecificSecurityGroup",
      "Action": [
          "lambda:CreateFunction",
          "lambda:UpdateFunctionConfiguration"
       ],
      "Effect": "Allow",
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
```

```
            "lambda:SecurityGroupIds": ["sg-1", "sg-2"]
        }
      }
    }
  ]
}
```

Tutorial VPC

Nelle esercitazioni seguenti è possibile connettere una funzione Lambda alle risorse del VPC.

- Tutorial: Utilizzo di una funzione Lambda per l'accesso ad Amazon RDS in un VPC Amazon
- Tutorial: Configurazione di una funzione Lambda per accedere ad Amazon in un ElastiCache Amazon VPC

Consentire alle funzioni Lambda di accedere a una risorsa in un Amazon VPC in un altro account

Puoi consentire alla tua AWS Lambda funzione di accedere a una risorsa in un Amazon VPC in Amazon Virtual Private Cloud gestito da un altro account, senza esporre nessuno dei due VPC a Internet. Questo modello di accesso consente di condividere i dati con altre organizzazioni utilizzando. AWS Utilizzando questo modello di accesso, è possibile condividere i dati VPCs con un livello di sicurezza e prestazioni maggiore rispetto a Internet. Configura la tua funzione Lambda per utilizzare una connessione peering Amazon VPC per accedere a queste risorse.

> **Warning**
>
> Quando consenti l'accesso tra account o VPCs, verifica che il tuo piano soddisfi i requisiti di sicurezza delle rispettive organizzazioni che gestiscono questi account. Seguire le istruzioni contenute in questo documento influirà sul livello di sicurezza delle risorse.

In questo tutorial, colleghi due account con una connessione peering utilizzando IPv4. Si configura una funzione Lambda che non è già connessa a un Amazon VPC. La risoluzione DNS viene configurata per connettere la funzione a risorse che non forniscono dati statici. IPs Per adattare queste istruzioni ad altri scenari di peering, consulta la Guida al peering VPC.

Prerequisiti

Per consentire a una funzione Lambda di accedere a una risorsa in un altro account, devi avere:

- Una funzione Lambda configurata per l'autenticazione e la successiva lettura dalla risorsa.
- Una risorsa in un altro account, ad esempio un cluster Amazon RDS, disponibile tramite Amazon VPC.
- Le credenziali per l'account della funzione Lambda e l'account della risorsa. Se non sei autorizzato a utilizzare l'account della tua risorsa, contatta un utente autorizzato per preparare quell'account.
- L'autorizzazione per creare e aggiornare un VPC (e le risorse del Amazon VPC di supporto) da associare alla funzione Lambda.
- L'autorizzazione per aggiornare il ruolo di esecuzione e la configurazione VPC per la funzione Lambda.
- L'autorizzazione per creare una connessione peering VPC nell'account della funzione Lambda.

- L'autorizzazione per accettare una connessione peering VPC nell'account della risorsa.
- L'autorizzazione per aggiornare la configurazione del VPC della risorsa (e le risorse Amazon VPC di supporto).
- Autorizzazione per richiamare la funzione Lambda.

Creare un Amazon VPC nell'account della tua funzione

Crea un Amazon VPC, le sottoreti, le tabelle di routing e un gruppo di sicurezza nell'account della tua funzione Lambda.

Come creare un VPC, sottoreti e altre risorse VPC tramite la console

1. Apri la console Amazon VPC all'indirizzo https://console.aws.amazon.com/vpc/.
2. Nel pannello di controllo, scegli Crea VPC.
3. Per il blocco IPv4 CIDR, fornisci un blocco CIDR privato. Il blocco CIDR non deve sovrapporsi ai blocchi utilizzati nel VPC della risorsa. Non scegliere un blocco utilizzato dal VPC delle risorse da IPs assegnare alle risorse o un blocco già definito nelle tabelle di routing nel VPC delle risorse. Per ulteriori informazioni sulla definizione dei blocchi CIDR appropriati, consulta Blocchi CIDR VPC.
4. Scegliere Customize (Personalizza) AZs.
5. Seleziona lo stesso della tua risorsa. AZs
6. Per Numero di sottoreti pubbliche, scegli 0.
7. Per VPC endpoints (Endpoint VPC), scegli None (Nessuno).
8. Seleziona Crea VPC.

Concedere le autorizzazioni VPC al ruolo di esecuzione della funzione

AWSLambdaVPCAccessExecutionRoleCollegati al ruolo di esecuzione della tua funzione per consentirle di connettersi a VPCs.

Per concedere le autorizzazioni VPC al ruolo di esecuzione della funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione .
3. Scegliere Configuration (Configurazione).

4. Seleziona Autorizzazioni.
5. In Nome del ruolo, scegli il ruolo di esecuzione.
6. Nella sezione Policy di autorizzazioni, seleziona Aggiungi autorizzazioni.
7. Dall'elenco a discesa scegli Collega policy.
8. Nella casella di ricerca immetti `AWSLambdaVPCAccessExecutionRole`.
9. Seleziona la casella di controllo sulla sinistra del nome della policy.
10. Scegli Add Permissions (Aggiungi autorizzazioni).

Per collegare la tua funzione al tuo Amazon VPC

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione .
3. Scegli la scheda Configurazione, quindi scegli VPC.
4. Scegli Modifica.
5. In VPC, seleziona il tuo VPC.
6. In Sottoreti, scegli le tue sottoreti.
7. Per Gruppi di sicurezza, scegli il gruppo di sicurezza predefinito del VPC.
8. Scegli Save (Salva).

Creare una richiesta di connessione peering VPC

Crea una richiesta di connessione peering VPC dal VPC della tua funzione (il VPC richiedente) al VPC della tua risorsa (il VPC accettante).

Per richiedere una connessione peering VPC al VPC della funzione

1. Apri la https://console.aws.amazon.com/vpc/.
2. Nel riquadro di navigazione, seleziona Connessioni peering.
3. Scegli Create peering connection (Crea connessione peering).
4. Per ID VPC (richiedente), seleziona il VPC della tua funzione.
5. In ID account immetti l'ID dell'account della risorsa.
6. Per ID VPC (accettante), inserisci il VPC della tua risorsa.

Preparare l'account della tua risorsa

Per creare la connessione peering e preparare il VPC della risorsa all'utilizzo della connessione, accedi all'account della risorsa con un ruolo che detiene le autorizzazioni elencate nei prerequisiti. I passaggi per accedere possono essere diversi in base al modo in cui l'account è protetto. Per ulteriori informazioni su come accedere a un AWS account, consulta la Guida per AWS l'utente di accesso. Nell'account della risorsa, completa le seguenti procedure.

Per accettare una richiesta di connessione peering VPC.

1. Apri la https://console.aws.amazon.com/vpc/.
2. Nel riquadro di navigazione, seleziona Connessioni peering.
3. Seleziona la connessione peering VPC in attesa (lo stato è pending-acceptance).
4. Scegli Azioni
5. Dal menù a discesa, scegli Accetta richiesta.
6. Quando viene chiesta la conferma, seleziona Accetta richiesta.
7. Scegli Modifica subito le tabelle di routing per aggiungere una route alla tabella di routing principale del VPC in modo da poter inviare e ricevere traffico attraverso la connessione peering.

Ispeziona le tabelle di routing per il VPC della risorsa. Il percorso generato da Amazon VPC potrebbe non stabilire la connettività a seconda di come è configurato il VPC della risorsa. Verifica la presenza di conflitti tra la nuova route e la configurazione esistente per il VPC. Per ulteriori informazioni sulla risoluzione dei problemi, consulta Risoluzione dei problemi di una connessione peering VPC nella Guida al peering VPC di Amazon Virtual Private Cloud Guide.

Per aggiornare il gruppo di sicurezza per la risorsa

1. Apri la https://console.aws.amazon.com/vpc/.
2. Nel pannello di navigazione, seleziona Gruppi di sicurezza.
3. Seleziona il gruppo di sicurezza per la tua risorsa.
4. Scegli Azioni.
5. Dall'elenco a discesa, scegli Modifica regole in entrata.
6. Scegli Aggiungi regola.
7. Per Origine, inserisci l'ID account della funzione e l'ID del gruppo di sicurezza, separati da una barra (ad esempio 111122223333/sg-1a2b3c4d).

8. Scegli Edit outbound rules (Modifica regole in uscita).
9. Verifica se il traffico in uscita è limitato. Le impostazioni VPC predefinite autorizzano tutto il traffico in uscita. Se il traffico in uscita è limitato, continua con il passaggio successivo.
10. Scegli Aggiungi regola.
11. Per Destinazione, inserisci l'ID account della funzione e l'ID del gruppo di sicurezza, separati da una barra (ad esempio 111122223333/sg-1a2b3c4d).
12. Scegliere Salva regole.

Per abilitare la risoluzione DNS per la connessione peering

1. Apri la https://console.aws.amazon.com/vpc/.
2. Nel riquadro di navigazione, seleziona Connessioni peering.
3. Seleziona la connessione peering.
4. Scegli Azioni.
5. Seleziona Modifica impostazioni DNS.
6. In Risoluzione DNS accettante, seleziona Consenti al VPC richiedente di risolvere il DNS degli host VPC accettanti su IP privato.
7. Scegli Save changes (Salva modifiche).

Aggiornare la configurazione VPC nell'account della tua funzione

Accedi all'account della funzione, quindi aggiorna la configurazione VPC.

Per aggiungere una route per la connessione peering VPC

1. Apri la https://console.aws.amazon.com/vpc/.
2. Nel riquadro di navigazione, seleziona Tabelle di routing.
3. Seleziona la casella di controllo accanto al nome della tabella di routing per la sottorete associata alla tua funzione.
4. Scegli Azioni.
5. Selezionare Modifica route.
6. Seleziona Aggiungi route.
7. Per Destinazione, inserisci il blocco CIDR per il VPC della risorsa.

8. Per Destinazione seleziona la connessione peering VPC.
9. Scegli Save changes (Salva modifiche).

Per ulteriori informazioni sulle considerazioni che potresti incontrare durante l'aggiornamento delle tabelle di routing, consulta Aggiornamento delle tabelle di routing per una connessione peering VPC.

Per aggiornare il gruppo di sicurezza per la tua funzione Lambda

1. Apri la https://console.aws.amazon.com/vpc/.
2. Nel pannello di navigazione, seleziona Gruppi di sicurezza.
3. Scegli Azioni.
4. Sceglere Edit inbound rules (Modifica regole in entrata).
5. Scegli Aggiungi regola.
6. Per Origine, inserisci l'ID account della risorsa e l'ID del gruppo di sicurezza, separati da una barra (ad esempio 111122223333/sg-1a2b3c4d).
7. Scegliere Salva regole.

Per abilitare la risoluzione DNS per la connessione peering

1. Apri la https://console.aws.amazon.com/vpc/.
2. Nel riquadro di navigazione, seleziona Connessioni peering.
3. Seleziona la connessione peering.
4. Scegli Azioni.
5. Seleziona Modifica impostazioni DNS.
6. In Risoluzione DNS richiedente, seleziona Consenti al VPC accettante di risolvere il DNS degli host VPC richiedenti su IP privato.
7. Scegli Save changes (Salva modifiche).

Test della funzione

Per creare un evento di test e controllare l'output della funzione

1. Nel riquadro Origine del codice, scegli Test.
2. Seleziona Crea nuovo evento.

3. Nel pannello JSON evento, sostituisci i valori predefiniti con un input appropriato per la tua funzione Lambda.
4. Scegli Richiama .
5. Nella scheda Risultati di esecuzione, conferma che la sezione Risposta contenga l'output previsto.

Inoltre, puoi controllare i log della tua funzione per verificare che siano quelli previsti.

Per visualizzare i record di invocazione della funzione in Logs CloudWatch

1. Selezionare la scheda Monitor (Monitora).
2. Scegli Visualizza registri. CloudWatch
3. Nella scheda Flussi di log, scegli il flusso di log per l'invocazione della funzione.
4. Conferma che i log siano quelli che ti aspetti.

Abilitare l'accesso a Internet per funzioni Lambda connesse a un VPC

Per impostazione predefinita, le funzioni Lambda vengono eseguite in un VPC gestito da Lambda con accesso a Internet. Per accedere alle risorse in un VPC nel tuo account, puoi aggiungere una configurazione VPC a una funzione. Ciò limita la funzione alle risorse all'interno di quel VPC, a meno che il VPC non abbia accesso a Internet. Questa pagina spiega come fornire l'accesso a Internet alle funzioni Lambda connesse al VPC.

Non ho ancora un VPC

Creazione del VPC

Il flusso di lavoro Crea VPC crea tutte le risorse VPC necessarie per una funzione Lambda per accedere alla rete Internet pubblica da una sottorete privata, incluse sottoreti, gateway NAT, gateway Internet e voci della tabella di routing.

Per creare il VPC

1. Apri alla console Amazon VPC all'indirizzo https://console.aws.amazon.com/vpc/.
2. Nel pannello di controllo, scegli Crea VPC.
3. Per Risorse da creare, scegli VPC e altro.
4. Configurazione del VPC
    a. Per Name tag auto-generation (Generazione automatica di tag nome), immetti un nome per il VPC.
    b. Per Blocco CIDR IPv4, mantieni il suggerimento predefinito o, in alternativa, inserisci il blocco CIDR richiesto dall'applicazione o dalla rete.
    c. Se l'applicazione comunica utilizzando indirizzi IPv6, scegli Blocco CIDR IPv6, Blocco CIDR IPv6 fornito da Amazon.
5. Configurazione delle sottoreti
    a. In Numero di zone di disponibilità, scegli 2. Per una disponibilità elevata consigliamo almeno due zone di disponibilità.
    b. Per Number of public subnets (Numero di sottoreti pubbliche), scegli 2.
    c. Per Number of private subnets (Numero di sottoreti private), scegli 2.

d. Puoi mantenere il blocco CIDR predefinito per la sottorete pubblica o, in alternativa, espandere Personalizza blocchi CIDR della sottorete e inserire un blocco CIDR. Per ulteriori informazioni, consulta Blocchi CIDR della sottorete.

6. Per Gateway NAT, scegli 1 per AZ per migliorare la resilienza.
7. Per Gateway Internet solo in uscita, scegli Sì se hai scelto di includere un blocco CIDR IPv6.
8. Per Endpoint VPC, mantieni il valore predefinito, Gateway S3. Questa opzione non prevede alcun costo. Per ulteriori informazioni, consulta Tipi di endpoint VPC per Amazon S3.
9. Per Opzioni DNS, mantieni le impostazioni predefinite.
10. Seleziona Crea VPC.

Configura la funzione Lambda

Per configurare un VPC quando si crea una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli Crea funzione.
3. In Informazioni di base, immettere un nome per la funzione in Nome funzione.
4. Espandere Advanced settings (Impostazioni avanzate).
5. Seleziona Abilita VPC, quindi scegli un VPC.
6. (Facoltativo) Per consentire il traffico IPv6 in uscita, selezionare Consenti il traffico IPv6 per sottoreti dual-stack.
7. Per Sottoreti, seleziona tutte le sottoreti private. Le sottoreti private possono accedere a Internet attraverso un gateway NAT. La connessione di una funzione a una sottorete pubblica non fornisce l'accesso a Internet.

> **Note**
>
> Se hai selezionato Consenti il traffico IPv6 per sottoreti dual-stack, tutte le sottoreti selezionate devono avere un blocco CIDR IPv4 e un blocco CIDR IPv6.

8. Per Gruppi di sicurezza, seleziona un gruppo di sicurezza che consenta il traffico in uscita.
9. Scegli Crea funzione.

Lambda crea automaticamente un ruolo di esecuzione con la policy gestita da AWS AWSLambdaVPCAccessExecutionRole. Le autorizzazioni in questa policy sono necessarie solo per creare interfacce di rete elastiche per la configurazione VPC, non per richiamare la funzione. Per applicare le autorizzazioni con privilegio minimo, puoi rimuovere la policy AWSLambdaVPCAccessExecutionRole dal tuo ruolo di esecuzione dopo aver creato la funzione e la configurazione VPC. Per ulteriori informazioni, consulta Autorizzazioni IAM richieste.

Per configurare un VPC per una funzione esistente

Per aggiungere una configurazione VPC a una funzione esistente, il ruolo di esecuzione della funzione deve disporre dell'autorizzazione per creare e gestire interfacce di rete elastiche. La policy gestita da AWS AWSLambdaVPCAccessExecutionRole include le autorizzazioni richieste. Per applicare le autorizzazioni con privilegio minimo, puoi rimuovere la policy AWSLambdaVPCAccessExecutionRole dal tuo ruolo di esecuzione dopo aver creato la funzione e la configurazione VPC.

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegli la scheda Configurazione, quindi scegli VPC.
4. In VPC, scegli Modifica.
5. Seleziona il VPC
6. (Facoltativo) Per consentire il traffico IPv6 in uscita, selezionare Consenti il traffico IPv6 per sottoreti dual-stack.
7. Per Sottoreti, seleziona tutte le sottoreti private. Le sottoreti private possono accedere a Internet attraverso un gateway NAT. La connessione di una funzione a una sottorete pubblica non fornisce l'accesso a Internet.

   > **Note**
   >
   > Se hai selezionato Consenti il traffico IPv6 per sottoreti dual-stack, tutte le sottoreti selezionate devono avere un blocco CIDR IPv4 e un blocco CIDR IPv6.

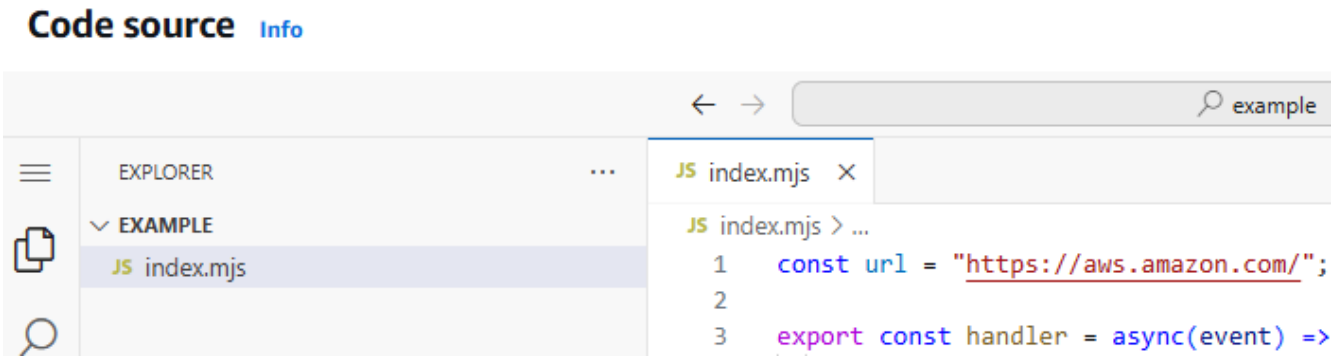
8. Per Gruppi di sicurezza, seleziona un gruppo di sicurezza che consenta il traffico in uscita.
9. Selezionare Salva.

Test della funzione

Usa il seguente codice di esempio per verificare che la tua funzione connessa al VPC possa raggiungere la rete Internet pubblica. In caso di successo, il codice restituisce un codice di stato `200`. Se non ha esito positivo, la funzione scade.

Node.js

1. Nel riquadro Codice sorgente della console Lambda, incolla il codice seguente nel file index.mjs. La funzione effettua una richiesta HTTP GET a un endpoint pubblico e restituisce il codice di risposta HTTP per verificare se la funzione ha accesso alla rete Internet pubblica.

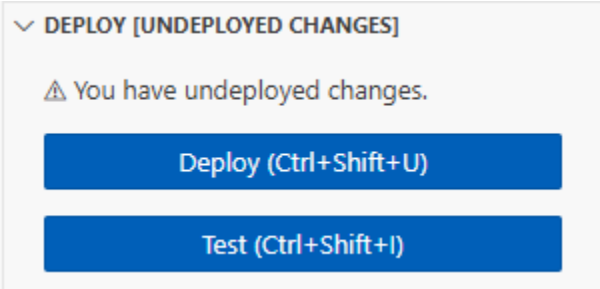


   Example Richiesta HTTP con async/await

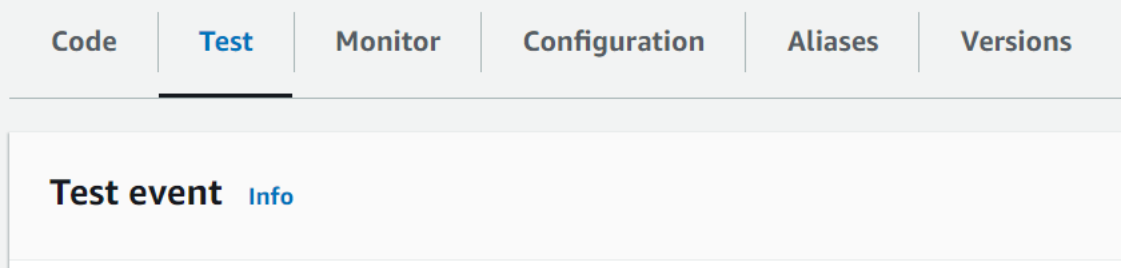
```
const url = "https://aws.amazon.com/";

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;
    }
};
```

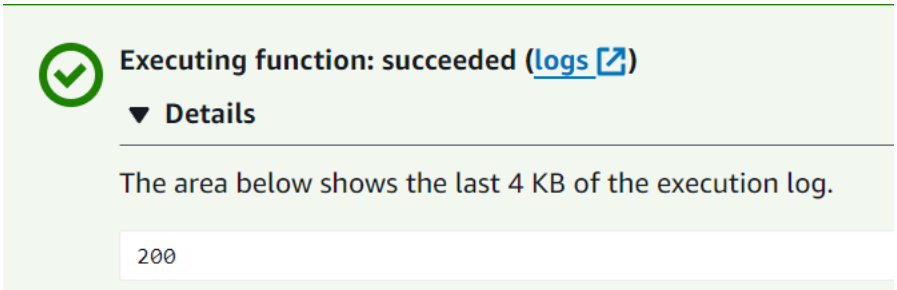
2. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



3. Seleziona la scheda Test.



4. Scegli Test (Esegui test).
5. La funzione restituisce un codice di stato `200`. Ciò significa che la funzione dispone di un accesso a Internet in uscita.



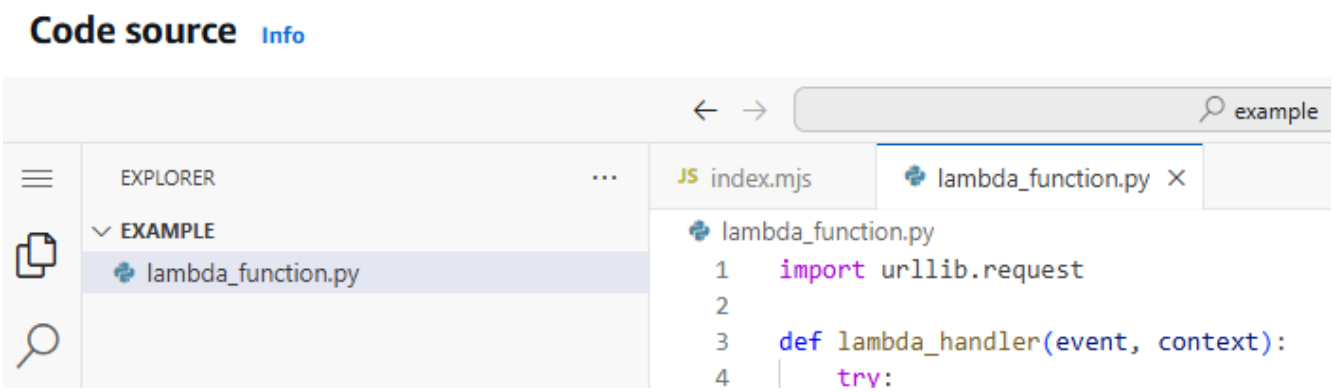
Se la funzione non riesce a raggiungere la rete Internet pubblica, ti verrà mostrato un messaggio di errore come questo:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

Python

1. Nel riquadro Codice sorgente della console Lambda, incolla il seguente codice seguente nel file lambda_function.py. La funzione effettua una richiesta HTTP GET a un endpoint pubblico

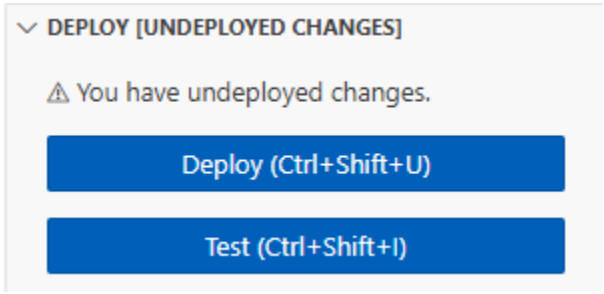
e restituisce il codice di risposta HTTP per verificare se la funzione ha accesso alla rete Internet pubblica.



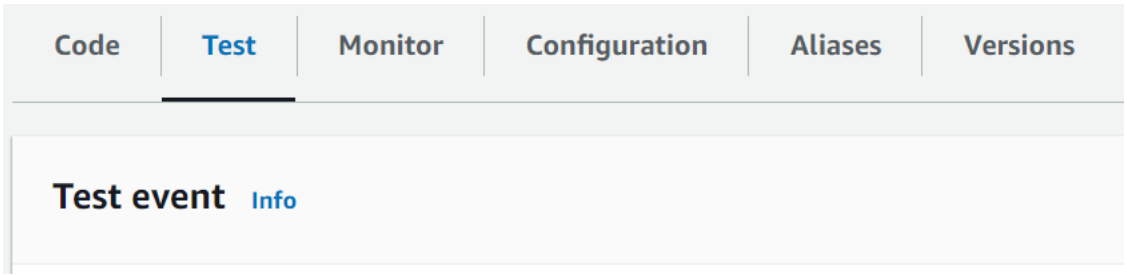

```
import urllib.request

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
```

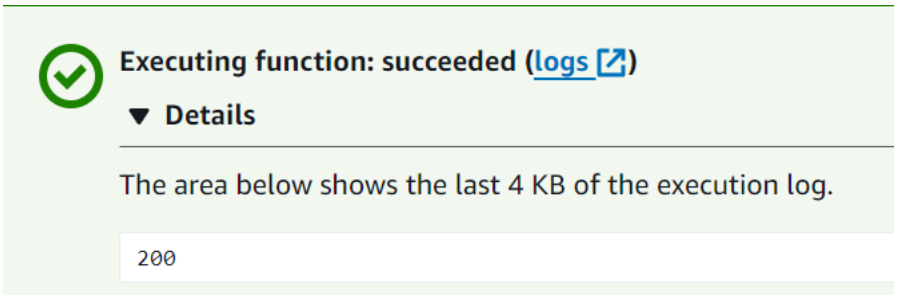
2. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



3. Seleziona la scheda Test.



4. Scegli Test (Esegui test).
5. La funzione restituisce un codice di stato `200`. Ciò significa che la funzione dispone di un accesso a Internet in uscita.



Se la funzione non riesce a raggiungere la rete Internet pubblica, ti verrà mostrato un messaggio di errore come questo:

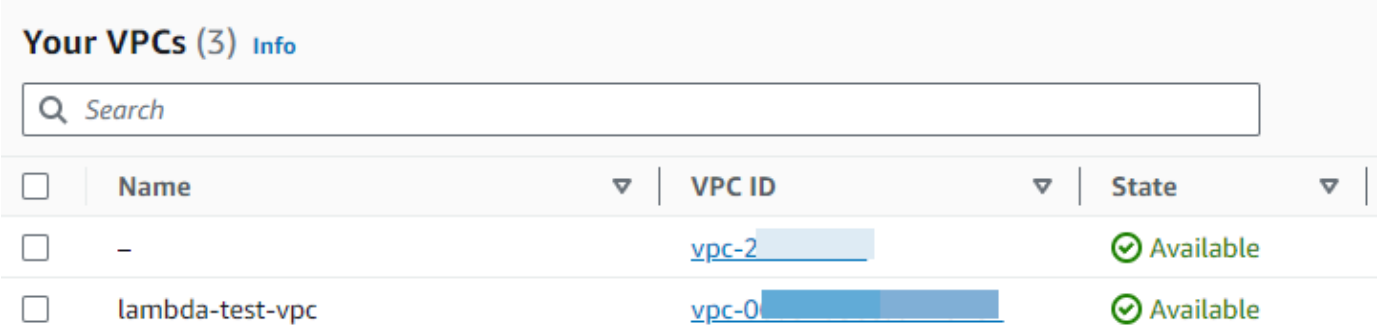
```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

Ho già un VPC

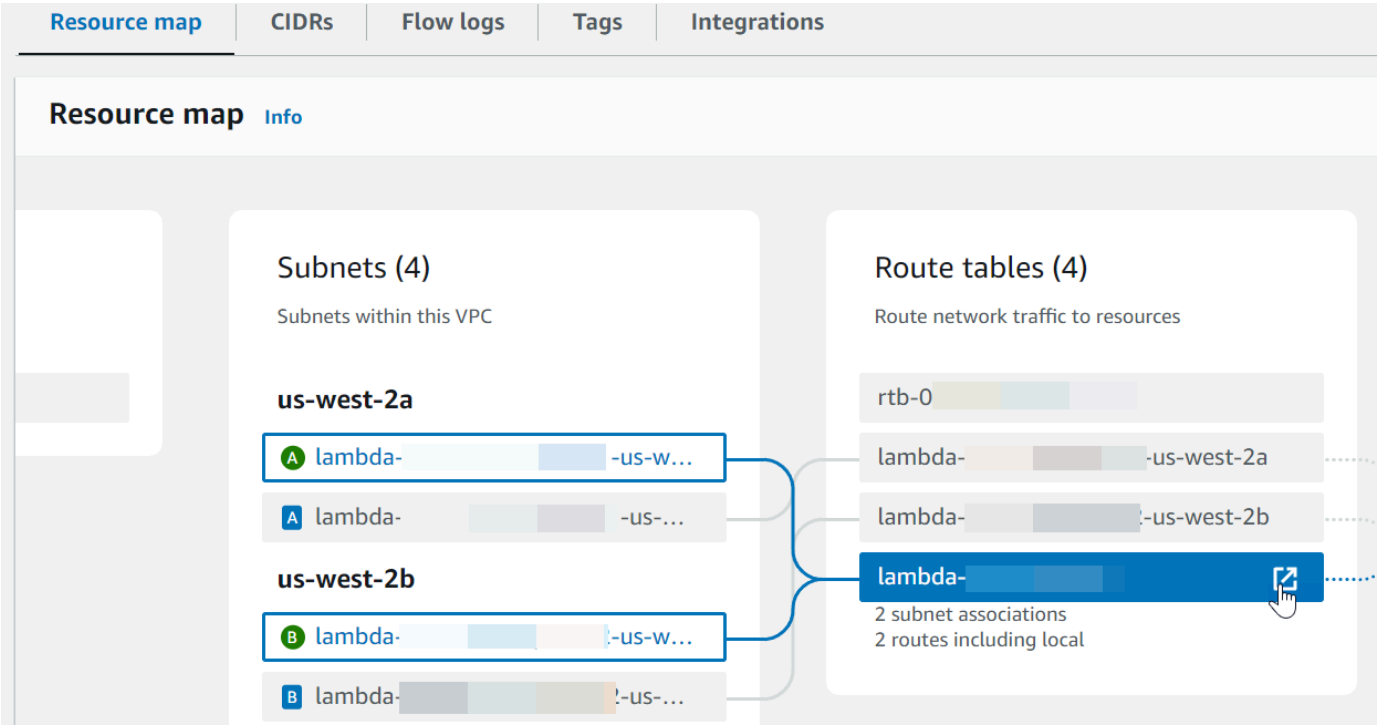
Se disponi già di un VPC ma devi configurare l'accesso pubblico a Internet per una funzione Lambda, completa questa procedura. Questa procedura presuppone che il VPC abbia almeno due sottoreti. Se non disponi di due sottoreti, consulta Creare una sottorete nella Guida per l'utente di Amazon VPC.

Verificare la configurazione della tabella di routing

1. Apri alla console Amazon VPC all'indirizzo https://console.aws.amazon.com/vpc/.
2. Scegli l'ID VPC.



3. Scorri verso il basso fino alla sezione Mappa delle risorse. Prendi nota delle mappature delle tabelle di routing. Apri ogni tabella di routing mappata a una sottorete.

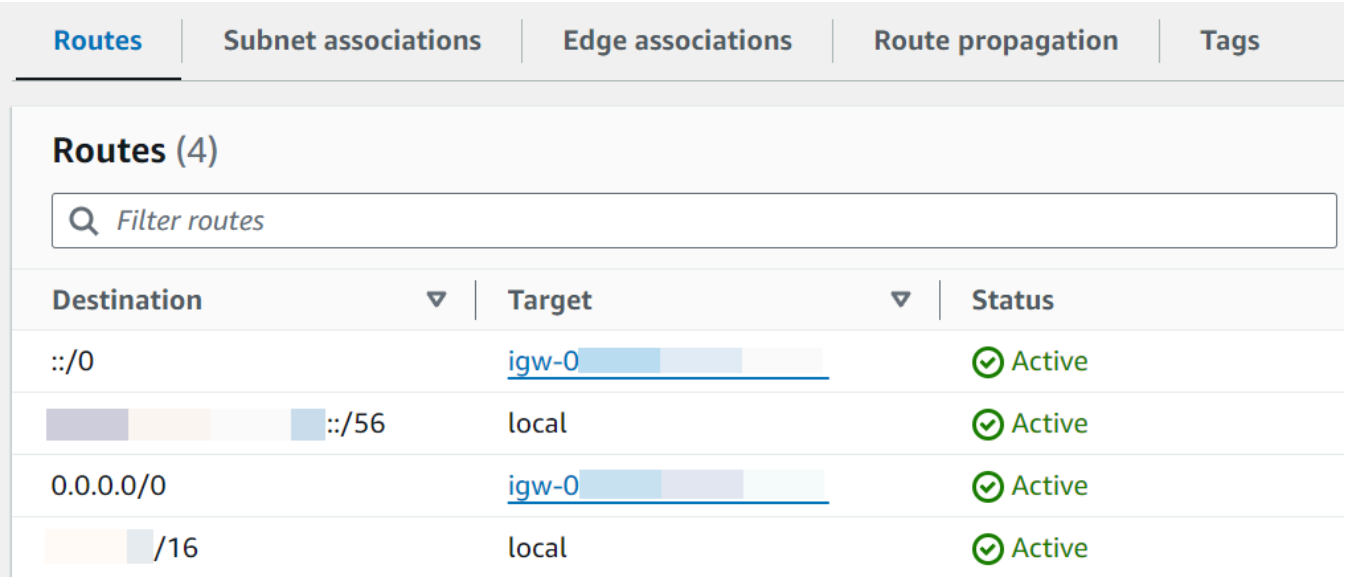


4. Scorri verso il basso fino alla scheda Route. Esamina i routing per determinare se il tuo VPC ha entrambe le seguenti tabelle di routing. Ciascuno di questi requisiti deve essere soddisfatto da una tabella di routing separata.

   - Il traffico collegato a Internet (`0.0.0.0/0` per IPv4, `::/0` per IPv6) viene indirizzato a un gateway Internet (`igw-xxxxxxxxxx`). Ciò significa che la sottorete associata alla tabella di routing è una sottorete pubblica.

     > **Note**
     >
     > Se la sottorete non ha un blocco CIDR IPv6, sarà visualizzata solo la route IPv4 (`0.0.0.0/0`).

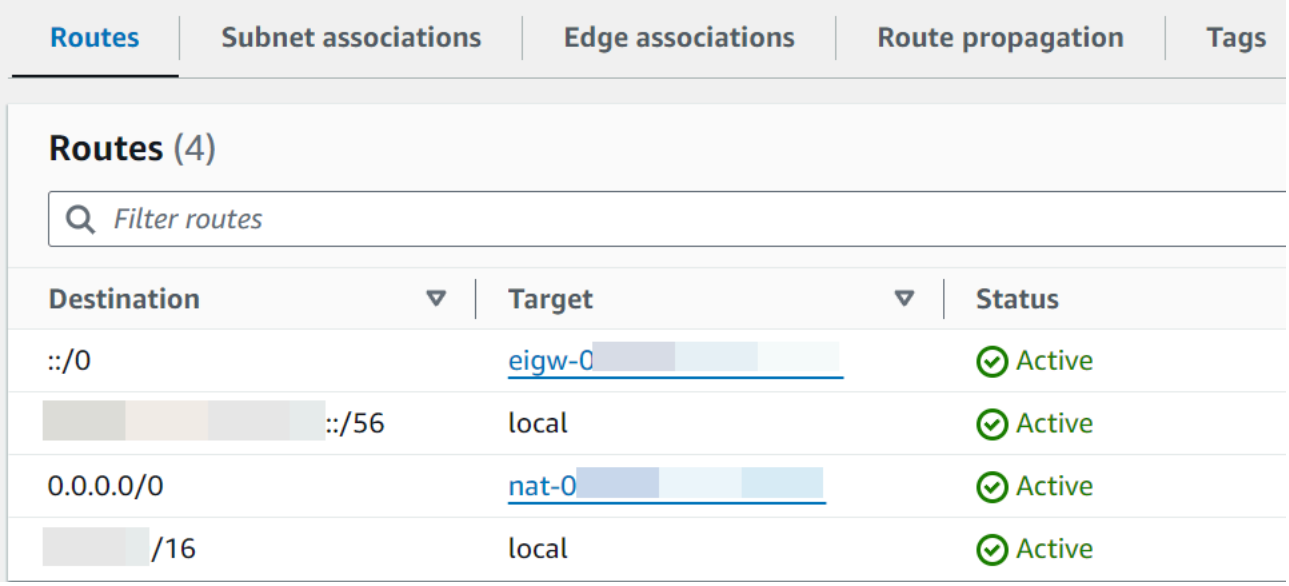
Example tabella di routing della sottorete pubblica



- Il traffico collegato a Internet per IPv4 (`0.0.0.0/0`) viene indirizzato a un gateway NAT (`nat-xxxxxxxxxx`) associato a una sottorete pubblica. Ciò significa che la sottorete è una sottorete privata che può accedere a Internet tramite il gateway NAT.

> **Note**
>
> Se la sottorete ha un blocco CIDR IPv6, la tabella di routing deve instradare il traffico IPv6 vincolato a Internet (`::/0`) a un gateway Internet di sola uscita (`eigw-xxxxxxxxxx`). Se la sottorete non ha un blocco CIDR IPv6, sarà visualizzata solo la route IPv4 (`0.0.0.0/0`).

Example tabella di routing della sottorete privata



5. Ripeti il passaggio precedente fino a quando non avrai esaminato ogni tabella di routing associata a una sottorete nel tuo VPC e avrai confermato di avere una tabella di routing con un gateway Internet e una tabella di routing con un gateway NAT.

   Se non disponi di due tabelle di routing, una con una route verso un gateway Internet e una con una route verso un gateway NAT, segui questi passaggi per creare le risorse mancanti e le voci della tabella di routing.

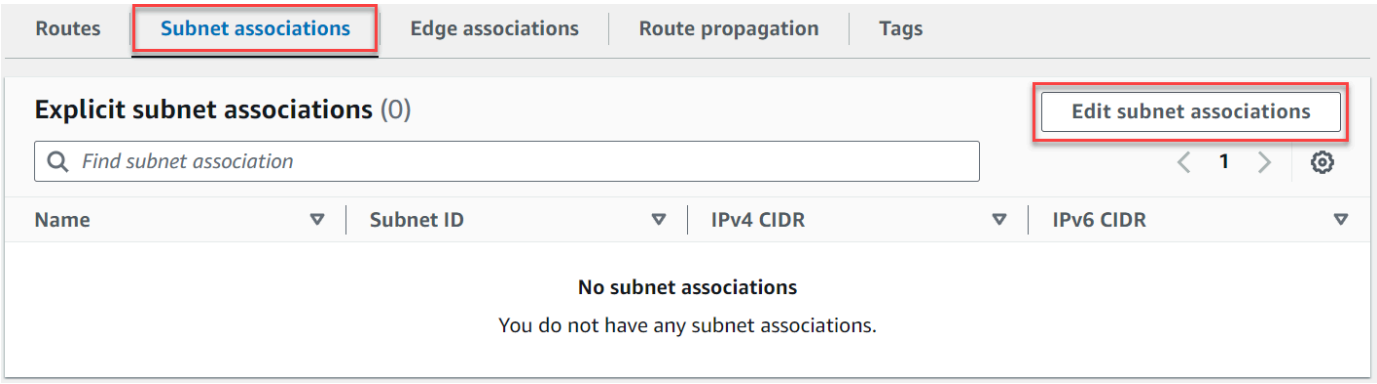
Creazione di una tabella di routing

Completa la seguente procedura per creare una tabella di routing e associarla a una sottorete.

Per creare una tabella di routing personalizzata utilizzando la console Amazon VPC

1. Apri la console Amazon VPC all'indirizzo https://console.aws.amazon.com/vpc/.
2. Nel riquadro di navigazione, seleziona Tabelle di routing.
3. Selezionare Create route table (Crea tabella di instradamento).
4. (Facoltativo) In Name (Nome), inserisci un nome per la tabella di instradamento.
5. In VPC, seleziona il VPC.
6. (Facoltativo) Per aggiungere un tag, scegli Add new tag (Aggiungi nuovo tag) e inserisci la chiave e il valore del tag.

7. Selezionare Create route table (Crea tabella di instradamento).
8. Nella scheda Associazioni sottorete scegli Modifica associazioni sottorete.



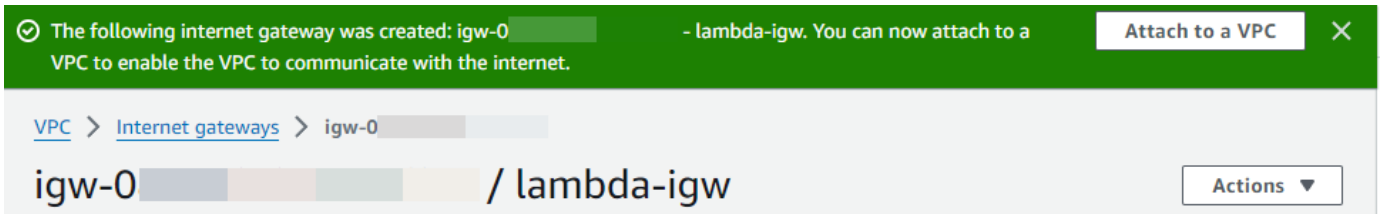
9. Seleziona la casella di controllo per la sottorete da associare alla tabella di instradamento.
10. Scegli Salva associazioni.

Creazione di un Internet Gateway

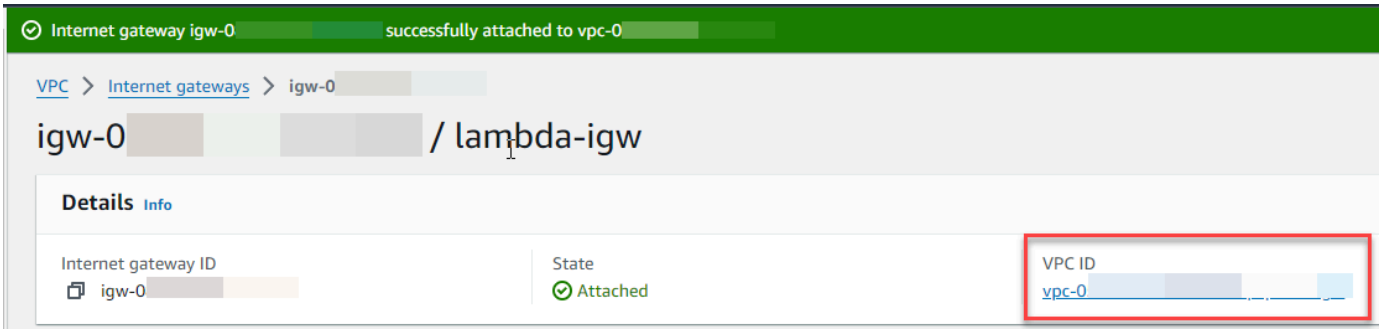
Segui questi passaggi per creare un gateway Internet, collegarlo al tuo VPC e aggiungerlo alla tabella di routing della sottorete pubblica.

Creare un gateway Internet

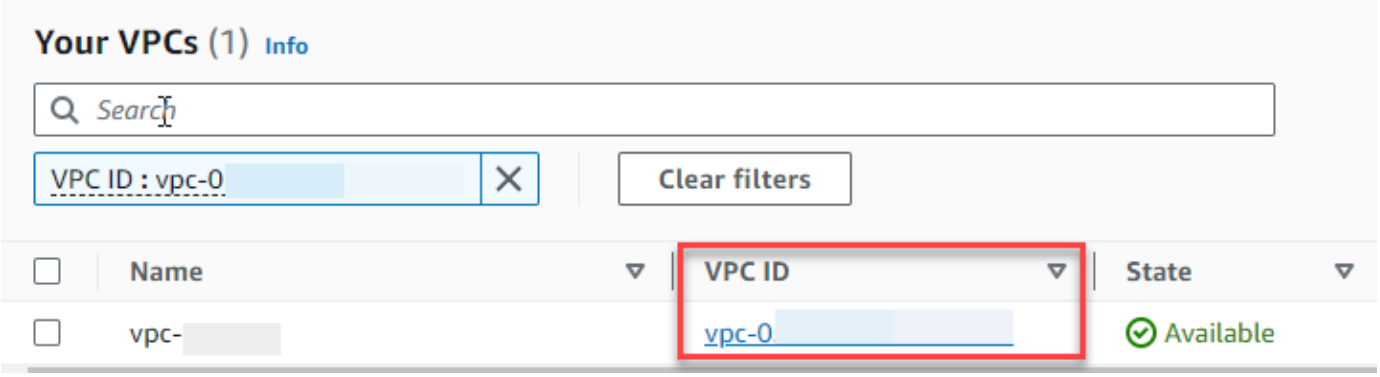
1. Apri alla console Amazon VPC all'indirizzo https://console.aws.amazon.com/vpc/.
2. Nel pannello di navigazione, scegli Internet gateways (Gateway Internet).
3. Scegliere Crea gateway Internet.
4. (Facoltativo) Inserisci un nome per il gateway Internet.
5. (Facoltativo) Per aggiungere un tag, scegli Add new tag (Aggiungi nuovo tag) e immetti la chiave e il valore del tag.
6. Scegliere Crea gateway Internet.
7. Scegli Collega a un VPC dal banner nella parte superiore dello schermo, seleziona un VPC disponibile e scegli Collega un gateway Internet.



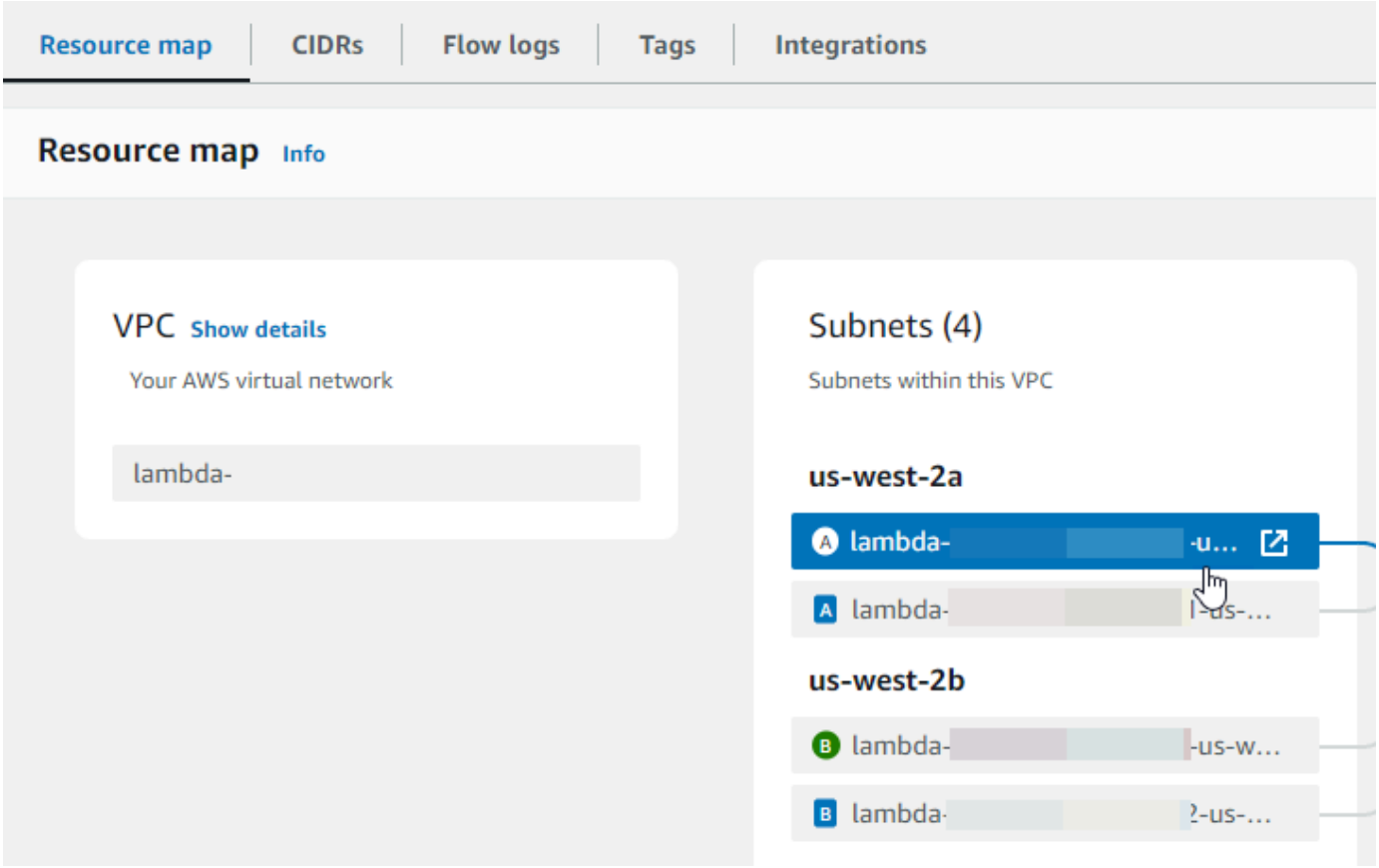
8. Scegli l'ID VPC.



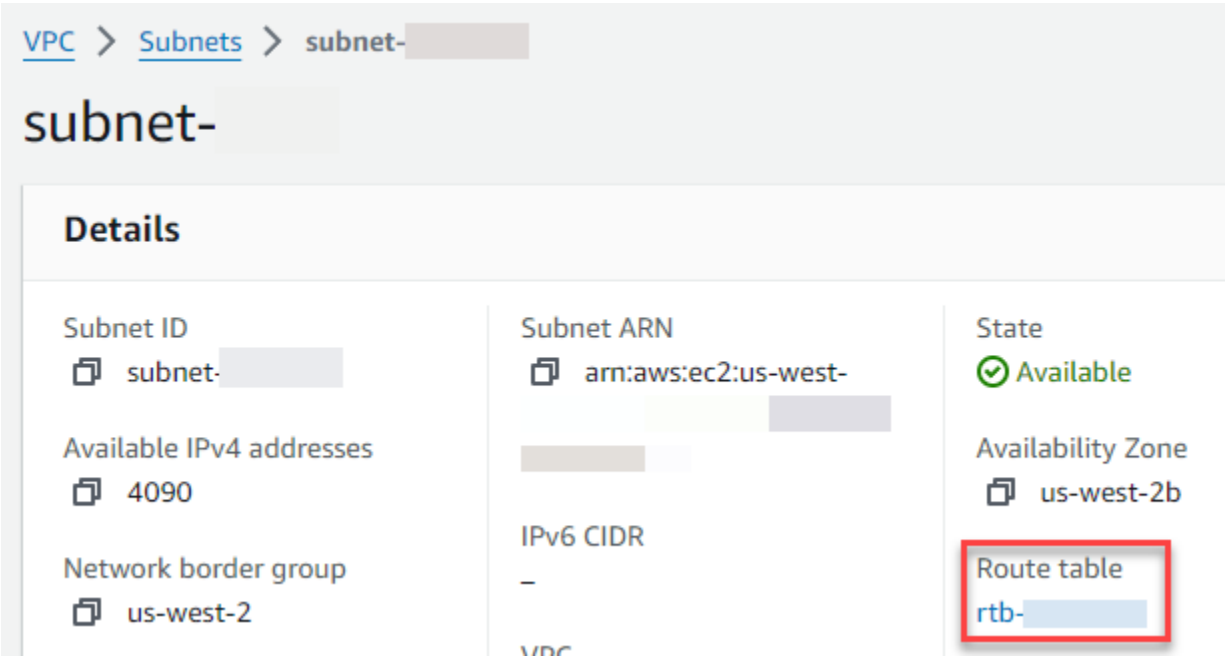
9. Scegli di nuovo l'ID VPC per aprire la pagina dei dettagli del VPC.



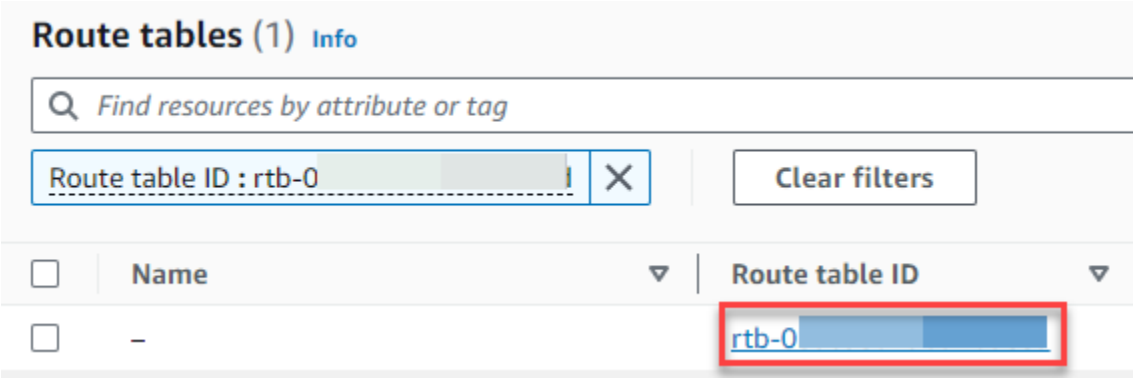
10. Scorri verso il basso fino alla sezione Mappa delle risorse, quindi scegli una sottorete. I dettagli della sottorete vengono visualizzati in una nuova scheda.



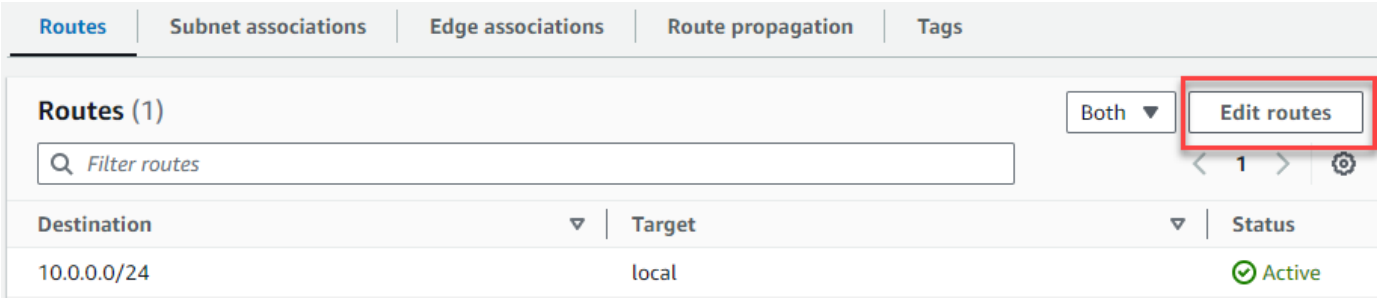
11. Scegli il link sotto Tabella di routing.



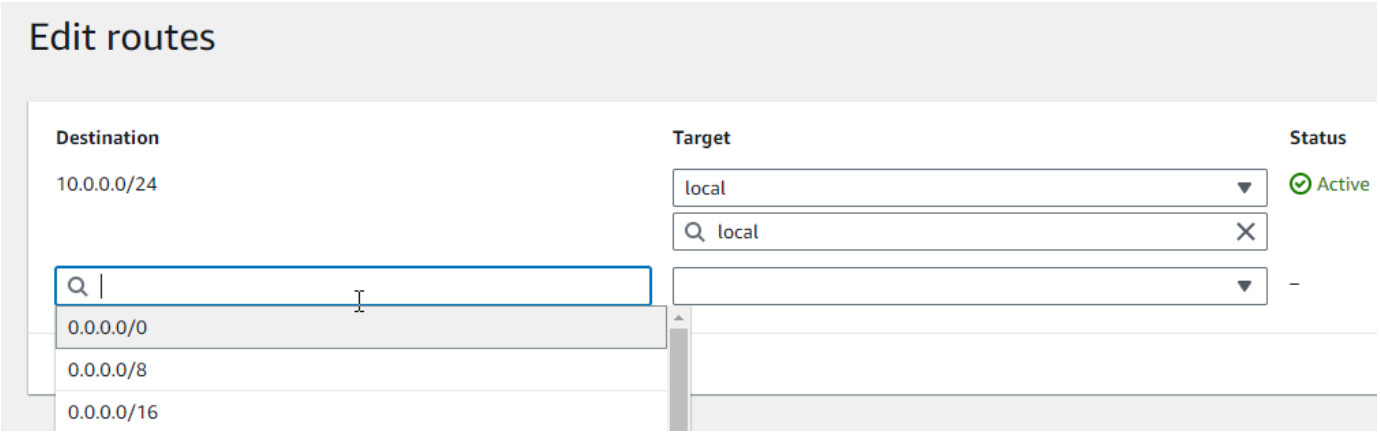
12. Seleziona ID tabella di routing per aprire la pagina dei dettagli della tabella di routing.



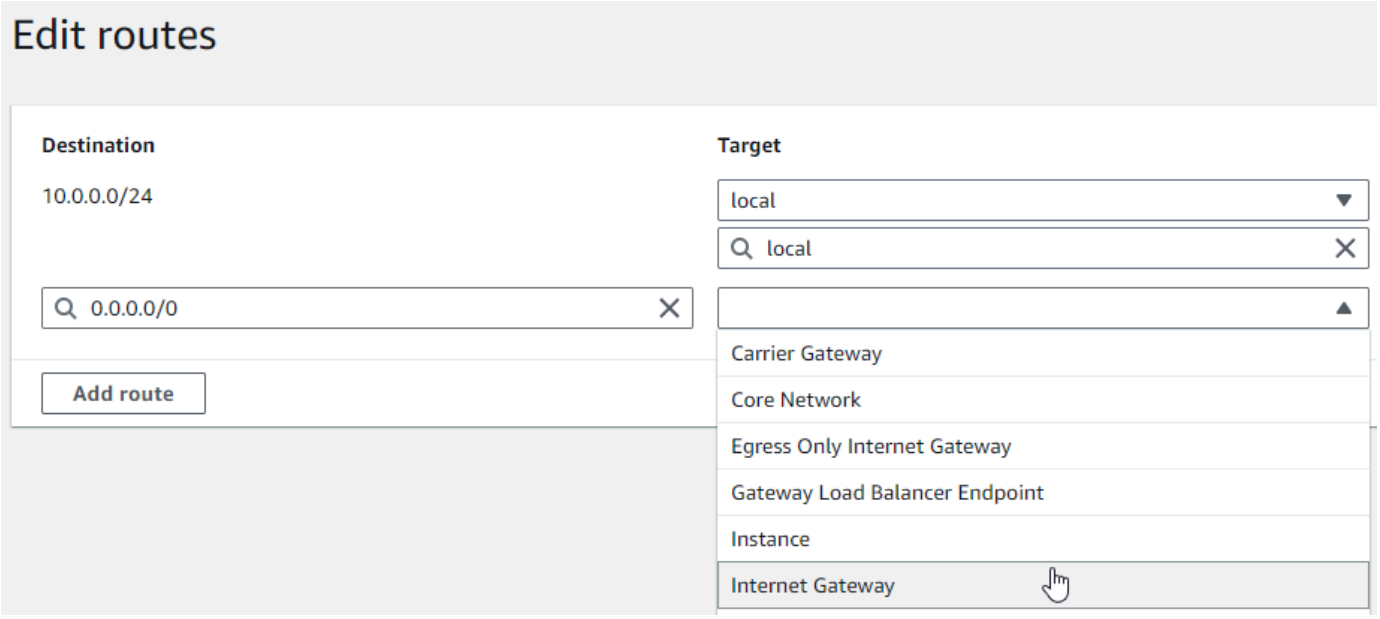
13. In Route, seleziona Modifica route.



14. Scegli Aggiungi route, quindi inserisci `0.0.0.0/0` nella casella Destinazione.



15. Per Destinazione, seleziona Internet gateway, quindi scegli il gateway Internet creato in precedenza. Se la sottorete ha un blocco CIDR IPv6, devi anche aggiungere una route per `::/0` per lo stesso gateway Internet.



16. Scegli Save changes (Salva modifiche).

Creazione di un gateway NAT

Segui questi passaggi per creare un gateway NAT, associarlo a una sottorete pubblica e aggiungerlo alla tabella di routing della sottorete privata.

Per creare un gateway NAT e associarlo a una sottorete pubblica

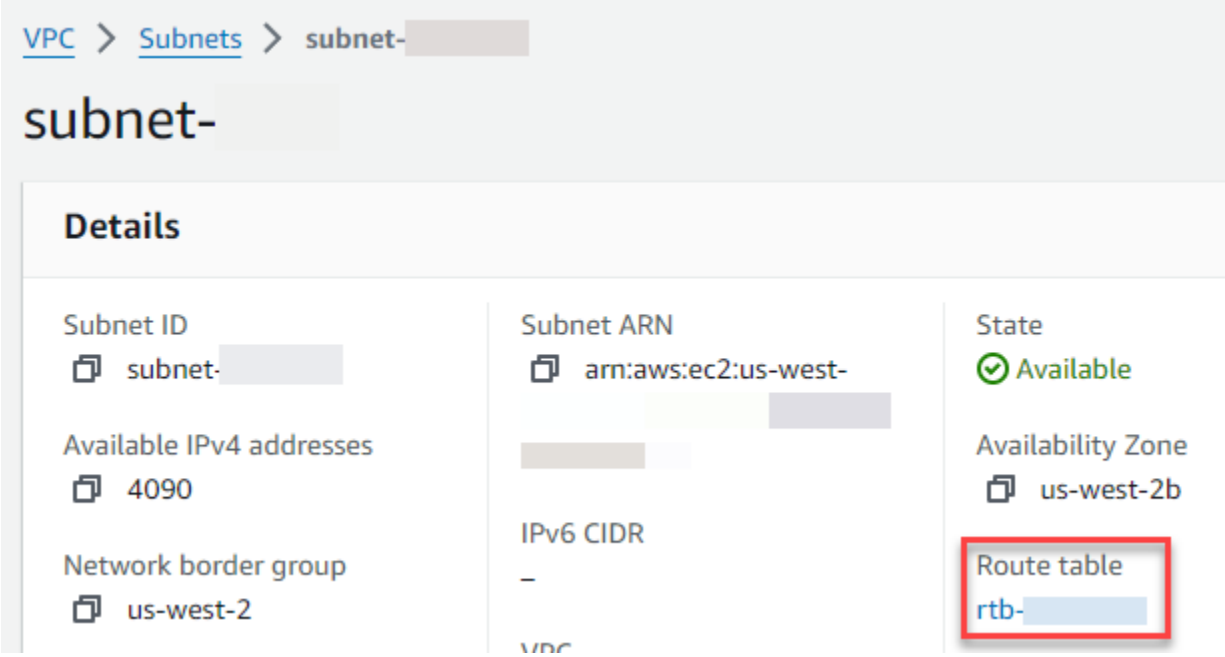
1. Nel riquadro di navigazione, scegli Gateway NAT.
2. Scegli Crea gateway NAT.
3. (Facoltativo) Inserisci un nome per il gateway NAT.
4. In Sottorete seleziona una sottorete pubblica del VPC. Una sottorete pubblica è una sottorete che ha una route diretta a un gateway Internet nella sua tabella di routing.

> **Note**
>
> I gateway NAT sono associati a una sottorete pubblica, ma la voce della tabella di routing si trova nella sottorete privata.

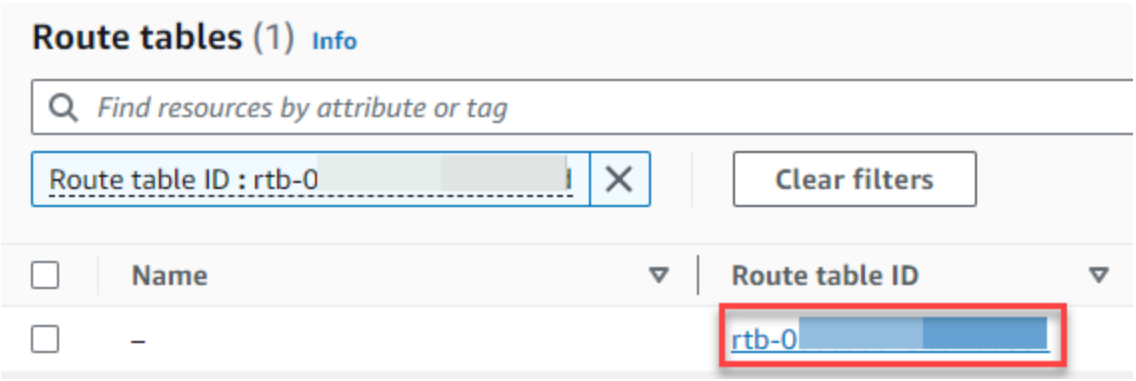
5. Per ID allocazione IP elastico, seleziona un indirizzo IP elastico o scegli Alloca IP elastico.
6. Scegli Crea gateway NAT.

Per aggiungere una route al gateway NAT nella tabella di routing della sottorete privata

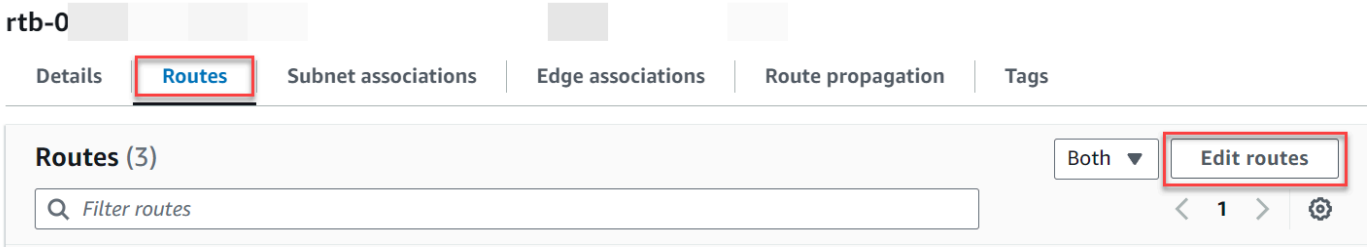
1. Nel pannello di navigazione, scegli Subnets (Sottoreti).
2. Seleziona una sottorete privata nel VPC. Una sottorete privata è una sottorete che non dispone di una route a un gateway Internet nella sua tabella di routing.
3. Scegli il link sotto Tabella di routing.



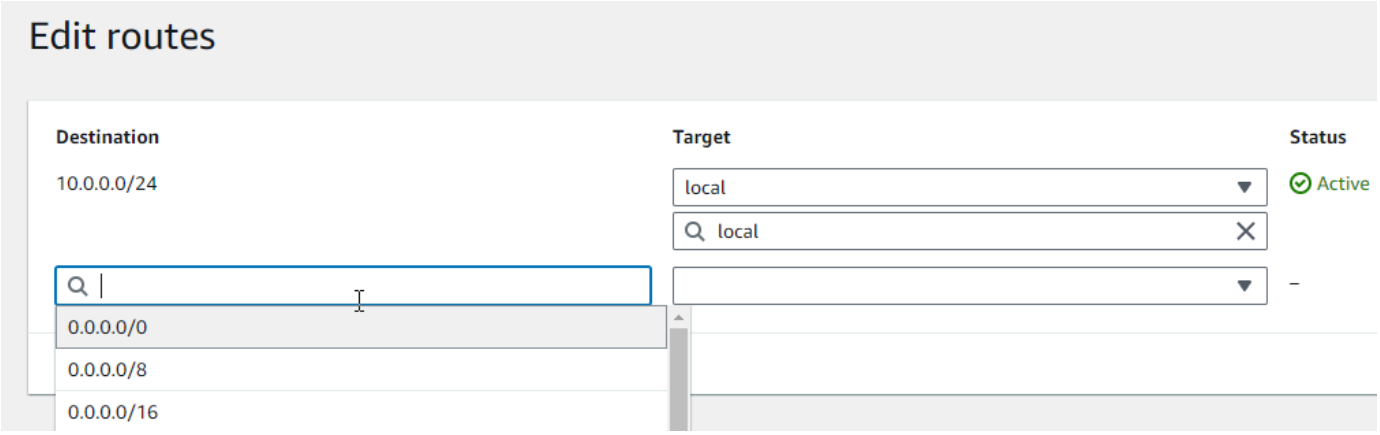
4. Seleziona ID tabella di routing per aprire la pagina dei dettagli della tabella di routing.



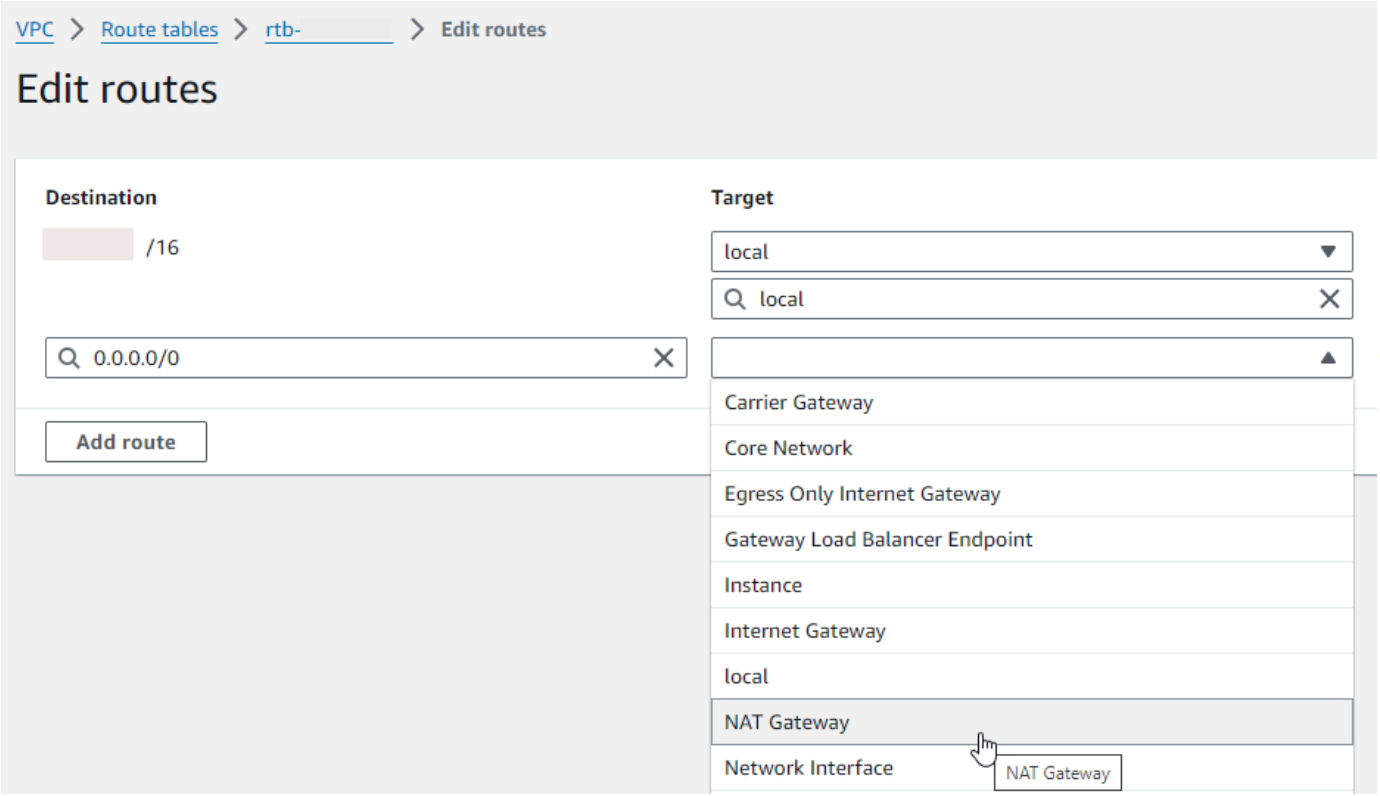
5. Scorri verso il basso e seleziona la scheda Route, quindi scegli Modifica route.



6. Scegli Aggiungi route, quindi inserisci `0.0.0.0/0` nella casella Destinazione.



7. Per Destinazione, seleziona Gateway NAT, quindi scegli il gateway NAT creato in precedenza.



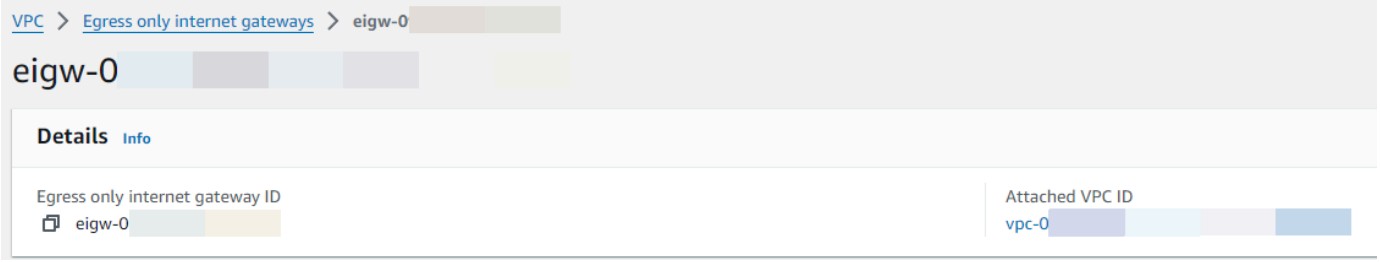
8. Scegli Save changes (Salva modifiche).

Creare un gateway Internet di sola uscita (solo IPv6)

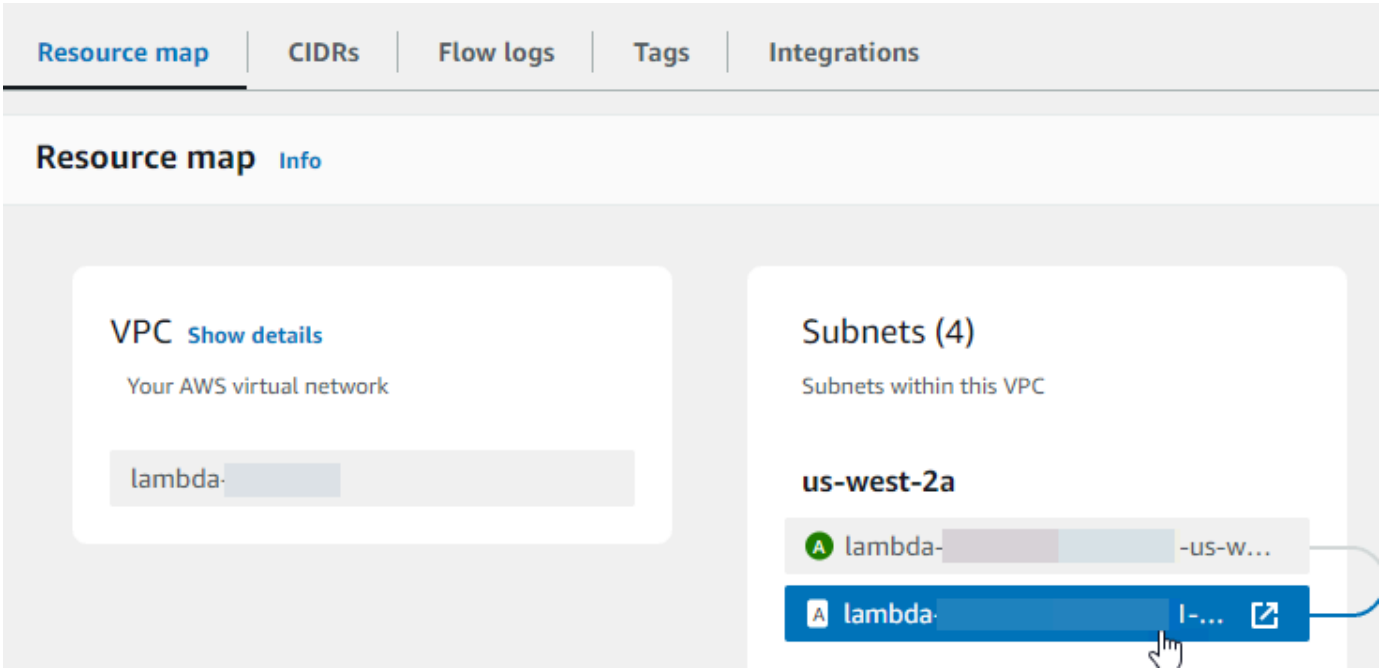
Segui questi passaggi per creare un gateway Internet solo in uscita e aggiungerlo alla tabella di routing della tua sottorete privata.

Per creare un gateway internet egress-only

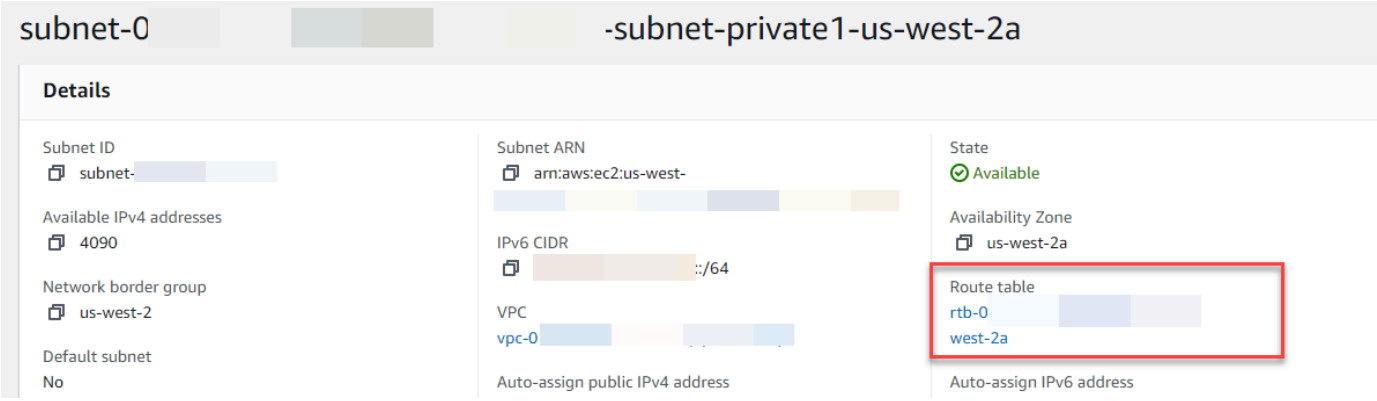
1. Nel riquadro di navigazione, seleziona Gateway Internet solo in uscita.
2. Seleziona Crea gateway Internet solo in uscita.
3. (Facoltativo) Immetti un nome.
4. Selezionare il VPC nel quale creare l'Internet Gateway egress-only.
5. Seleziona Crea gateway Internet solo in uscita.
6. Scegli il link sotto ID VPC collegato.



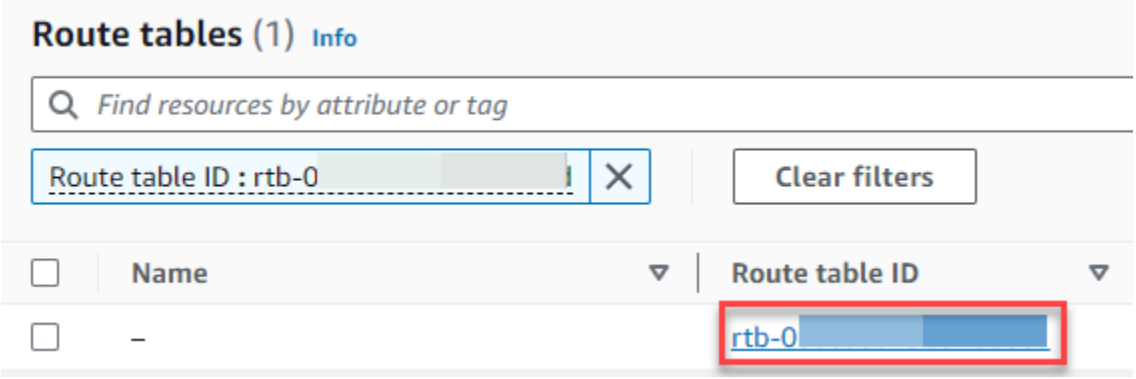
7. Scegliere il link sotto ID VPC per aprire la pagina dei dettagli del VPC.
8. Scorri verso il basso fino alla sezione Mappa delle risorse, quindi scegli una sottorete privata. Una sottorete privata è una sottorete che non dispone di una route a un gateway Internet nella sua tabella di routing. I dettagli della sottorete vengono visualizzati in una nuova scheda.



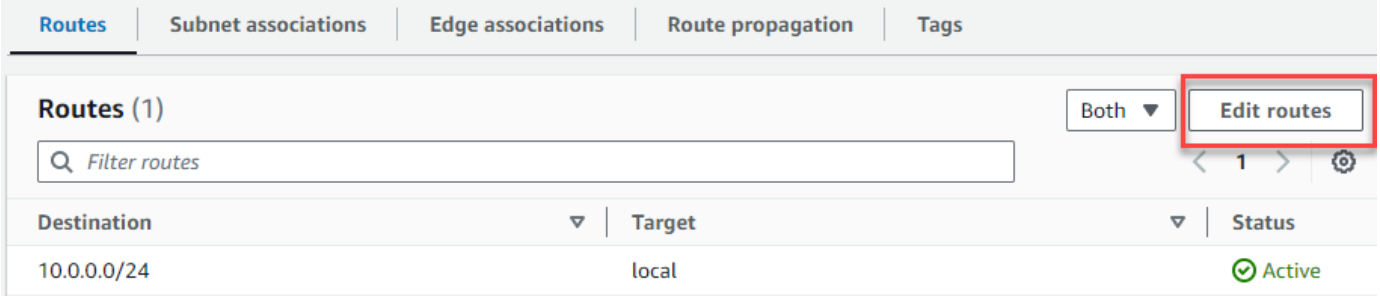
9. Scegli il link sotto Tabella di routing.



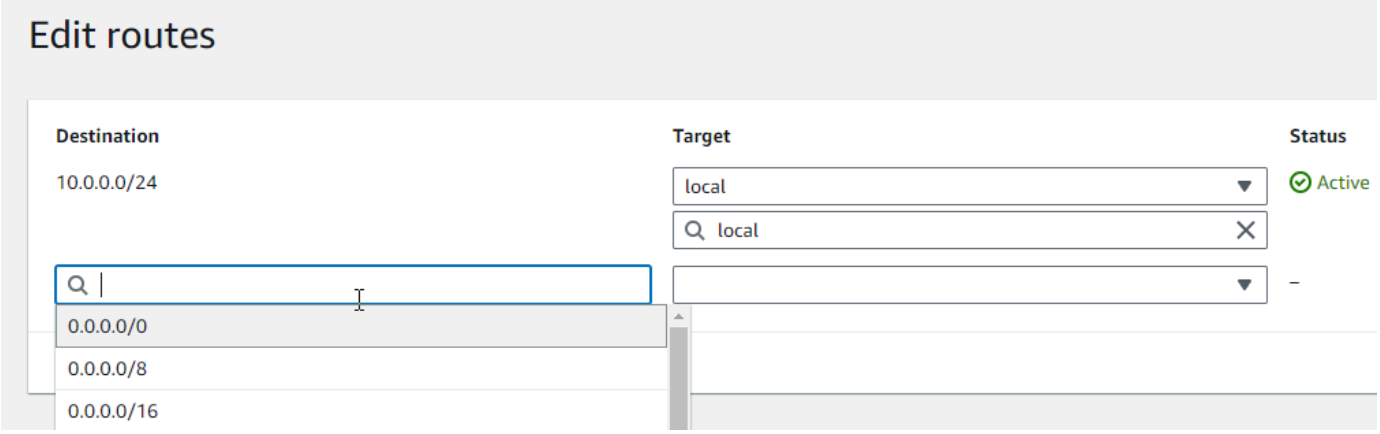
10. Seleziona ID tabella di routing per aprire la pagina dei dettagli della tabella di routing.



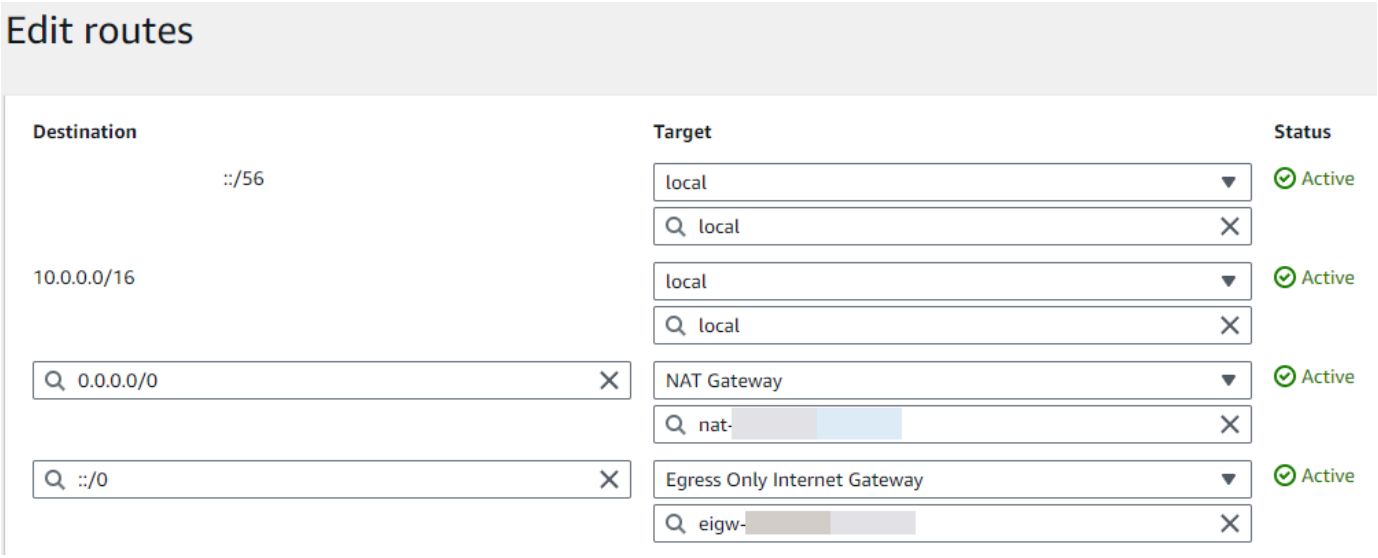
11. In Route, seleziona Modifica route.



12. Scegli Aggiungi route, quindi inserisci `::/0` nella casella Destinazione.



13. Per Destinazione, seleziona Gateway Internet solo in uscita, quindi scegli il gateway creato in precedenza.



14. Scegli Save changes (Salva modifiche).

Configura la funzione Lambda

Per configurare un VPC quando si crea una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli Crea funzione.
3. In Informazioni di base, immettere un nome per la funzione in Nome funzione.
4. Espandere Advanced settings (Impostazioni avanzate).
5. Seleziona Abilita VPC, quindi scegli un VPC.
6. (Facoltativo) Per consentire il traffico IPv6 in uscita, selezionare Consenti il traffico IPv6 per sottoreti dual-stack.
7. Per Sottoreti, seleziona tutte le sottoreti private. Le sottoreti private possono accedere a Internet attraverso un gateway NAT. La connessione di una funzione a una sottorete pubblica non fornisce l'accesso a Internet.

> **Note**
>
> Se hai selezionato Consenti il traffico IPv6 per sottoreti dual-stack, tutte le sottoreti selezionate devono avere un blocco CIDR IPv4 e un blocco CIDR IPv6.

8. Per Gruppi di sicurezza, seleziona un gruppo di sicurezza che consenta il traffico in uscita.
9. Scegli Crea funzione.

Lambda crea automaticamente un ruolo di esecuzione con la policy gestita da AWS AWSLambdaVPCAccessExecutionRole. Le autorizzazioni in questa policy sono necessarie solo per creare interfacce di rete elastiche per la configurazione VPC, non per richiamare la funzione. Per applicare le autorizzazioni con privilegio minimo, puoi rimuovere la policy AWSLambdaVPCAccessExecutionRole dal tuo ruolo di esecuzione dopo aver creato la funzione e la configurazione VPC. Per ulteriori informazioni, consulta Autorizzazioni IAM richieste.

Per configurare un VPC per una funzione esistente

Per aggiungere una configurazione VPC a una funzione esistente, il ruolo di esecuzione della funzione deve disporre dell'autorizzazione per creare e gestire interfacce di rete elastiche. La policy gestita da AWS AWSLambdaVPCAccessExecutionRole include le autorizzazioni richieste. Per applicare le autorizzazioni con privilegio minimo, puoi rimuovere la policy AWSLambdaVPCAccessExecutionRole dal tuo ruolo di esecuzione dopo aver creato la funzione e la configurazione VPC.

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegli la scheda Configurazione, quindi scegli VPC.
4. In VPC, scegli Modifica.
5. Seleziona il VPC
6. (Facoltativo) Per consentire il traffico IPv6 in uscita, selezionare Consenti il traffico IPv6 per sottoreti dual-stack.
7. Per Sottoreti, seleziona tutte le sottoreti private. Le sottoreti private possono accedere a Internet attraverso un gateway NAT. La connessione di una funzione a una sottorete pubblica non fornisce l'accesso a Internet.

   > **Note**
   >
   > Se hai selezionato Consenti il traffico IPv6 per sottoreti dual-stack, tutte le sottoreti selezionate devono avere un blocco CIDR IPv4 e un blocco CIDR IPv6.

8. Per Gruppi di sicurezza, seleziona un gruppo di sicurezza che consenta il traffico in uscita.

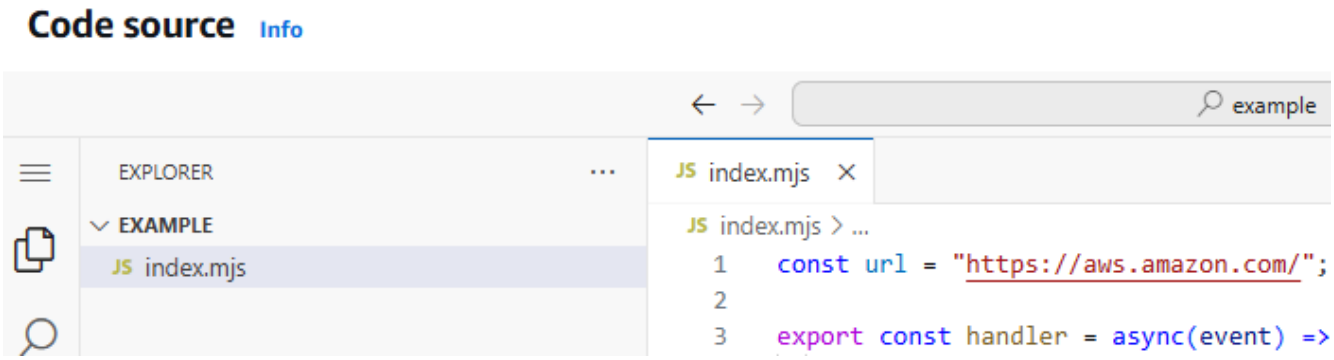
9. Selezionare Salva.

Test della funzione

Usa il seguente codice di esempio per verificare che la tua funzione connessa al VPC possa raggiungere la rete Internet pubblica. In caso di successo, il codice restituisce un codice di stato `200`. Se non ha esito positivo, la funzione scade.

Node.js

1. Nel riquadro Codice sorgente della console Lambda, incolla il codice seguente nel file index.mjs. La funzione effettua una richiesta HTTP GET a un endpoint pubblico e restituisce il codice di risposta HTTP per verificare se la funzione ha accesso alla rete Internet pubblica.

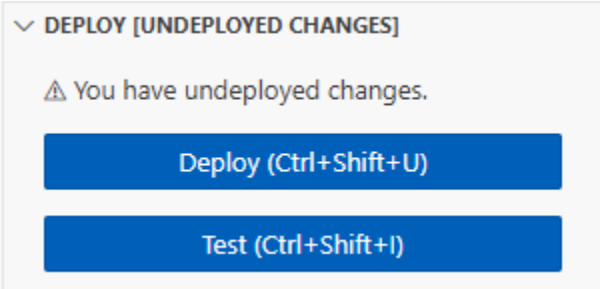


   Example Richiesta HTTP con async/await

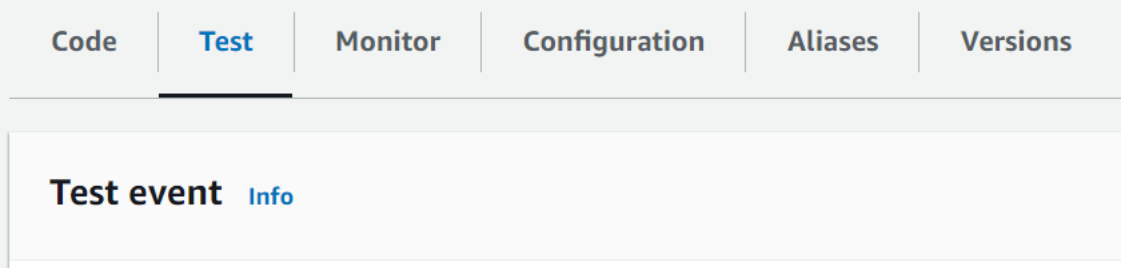
```
const url = "https://aws.amazon.com/";

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;
    }
};
```

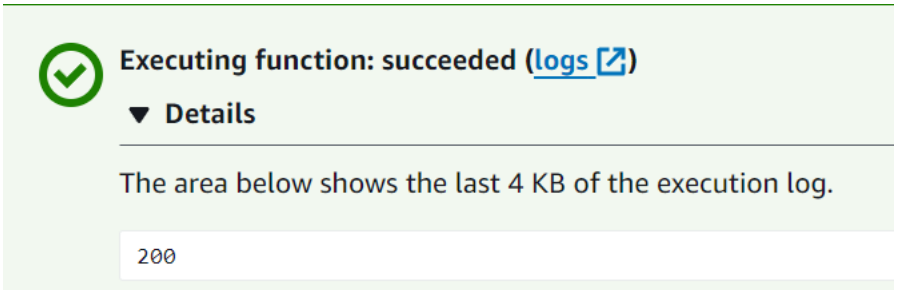
2. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



3. Seleziona la scheda Test.



4. Scegli Test (Esegui test).
5. La funzione restituisce un codice di stato `200`. Ciò significa che la funzione dispone di un accesso a Internet in uscita.



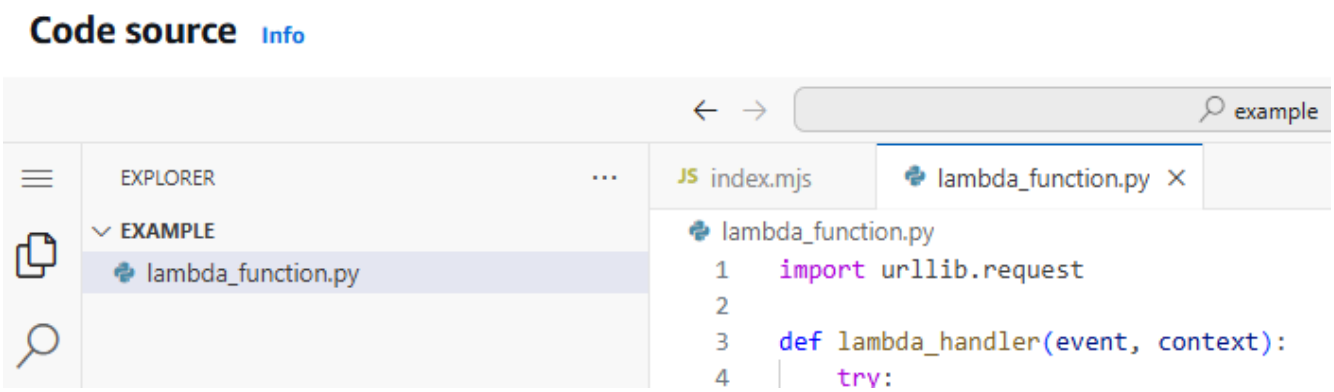
Se la funzione non riesce a raggiungere la rete Internet pubblica, ti verrà mostrato un messaggio di errore come questo:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

Python

1. Nel riquadro Codice sorgente della console Lambda, incolla il seguente codice seguente nel file lambda_function.py. La funzione effettua una richiesta HTTP GET a un endpoint pubblico

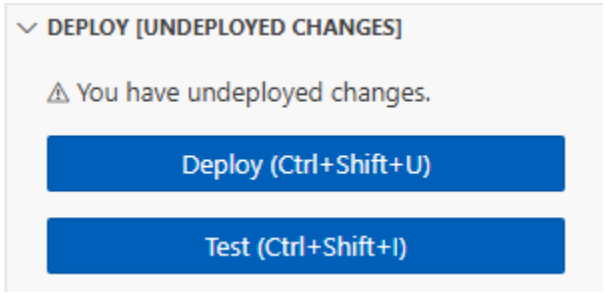
e restituisce il codice di risposta HTTP per verificare se la funzione ha accesso alla rete Internet pubblica.



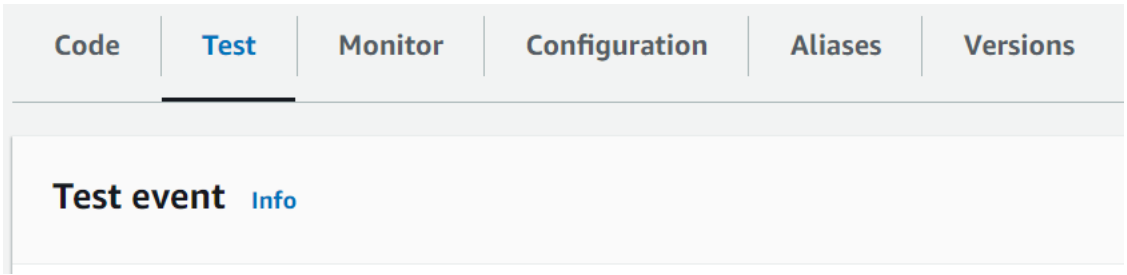

```
import urllib.request

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
```

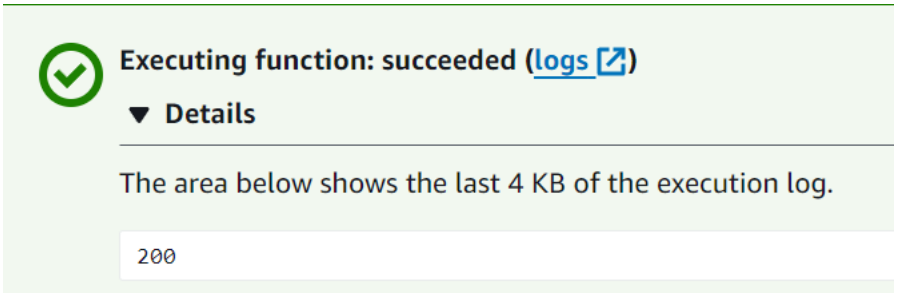
2. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



3. Seleziona la scheda Test.



4. Scegli Test (Esegui test).
5. La funzione restituisce un codice di stato `200`. Ciò significa che la funzione dispone di un accesso a Internet in uscita.



Se la funzione non riesce a raggiungere la rete Internet pubblica, ti verrà mostrato un messaggio di errore come questo:

```
{
  "errorMessage": "2024-04-11T17:22:20.857Z abe12jlc-640a-8157-0249-9be825c2y110
 Task timed out after 3.01 seconds"
}
```

Connessione di endpoint VPC con interfaccia in entrata per Lambda

Se usi Amazon Virtual Private Cloud (Amazon VPC) per l'hosting delle risorse AWS, puoi stabilire una connessione tra il cloud privato virtuale (VPC) e Lambda. È possibile utilizzare questa connessione per richiamare la propria funzione Lambda senza attraversare l'Internet pubblico.

È possibile stabilire una connessione privata tra il VPC e Lambda creando un endpoint VPC di interfaccia. Gli endpoint di interfaccia sono alimentati da AWS PrivateLink, che consente di accedere privatamente alle API Lambda APIs senza un gateway Internet, un dispositivo NAT, una connessione VPN o una connessione AWS Direct Connect. Le istanze presenti nel VPC non richiedono indirizzi IP pubblici per comunicare con le API Lambda. Il traffico tra il tuo VPC e Lambda non esce dalla rete AWS.

Ogni endpoint di interfaccia è rappresentato da una o più interfacce di rete elastiche nelle sottoreti. Un'interfaccia di rete fornisce un indirizzo IP privato che funge da punto di ingresso per il traffico verso Lambda.

Sezioni

- Considerazioni per gli endpoint di interfaccia Lambda
- Creazione di un endpoint di interfaccia per Lambda
- Creazione di una policy degli endpoint di interfaccia per Lambda

Considerazioni per gli endpoint di interfaccia Lambda

Prima di impostare un endpoint di interfaccia per Lambda, esaminare l'argomento Proprietà e limitazioni degli endpoint dell'interfaccia nella Guida per l'utente di Amazon VPC.

Puoi chiamare qualsiasi operazione API Lambda dal tuo VPC. Ad esempio, è possibile richiamare la funzione Lambda invocando l'API `Invoke` dall'interno del VPC. Per l'elenco completo delle API Lambda, vedere Azioni nella Guida di riferimento alle API Lambda.

`use1-az3` è una regione a capacità limitata per le funzioni Lambda VPC. L'utilizzo di sottoreti con le funzioni Lambda in questa zona di disponibilità non è consigliato perché, in caso di interruzione del servizio, potrebbe provocare una riduzione della ridondanza zonale.

Keep-alive per connessioni persistenti

Lambda elimina le connessioni inattive nel tempo, quindi occorre utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedere Riutilizzo delle connessioni con Keep-Alive in Node.js nella Guida per gli sviluppatori di AWS SDK per JavaScript.

Considerazioni sulla fatturazione

Non vi è alcun costo aggiuntivo per accedere a una funzione Lambda tramite un endpoint di interfaccia. Per ulteriori informazioni sui prezzi di Lambda, consulta Prezzi di AWS Lambda.

Prezzi standard per AWS PrivateLink si applica agli endpoint di interfaccia per Lambda. L'account AWS viene fatturato ogni ora che viene eseguito il provisioning di un endpoint di interfaccia in ogni zona di disponibilità e per i dati elaborati tramite l'endpoint dell'interfaccia. Per ulteriori informazioni sui prezzi degli endpoint di interfaccia, consulta Prezzi di AWS PrivateLink.

Considerazioni sul peering VPC

È possibile connettere altri VPC al VPC con endpoint di interfaccia utilizzando Peering VPC. Il peering VPC è una connessione di rete tra due VPC. Puoi creare una connessione peering VPC tra i tuoi VPC oppure con un VPC in un altro account AWS. I VPC possono essere anche in due diverse regioni AWS.

Il traffico tra VPC peered rimane sulla rete AWS e non attraversa la rete Internet pubblica. Una volta eseguito il peering dei vPC, risorse come istanze Amazon Elastic Compute Cloud (Amazon EC2), istanze Amazon Relational Database Service (Amazon RDS) o funzioni Lambda abilitate per VPC in entrambi i VPC possono accedere all'API Lambda tramite gli endpoint di interfaccia creati in uno dei VPC.

Creazione di un endpoint di interfaccia per Lambda

Puoi creare un endpoint di interfaccia per Lambda utilizzando la console Amazon VPC o AWS Command Line Interface (AWS CLI). Per ulteriori informazioni, consultare Creazione di un endpoint di interfaccia nella Guida per l'utente di Amazon VPC.

Per creare un endpoint di interfaccia per Lambda (console)

1. Aprire la pagina Endpoint della console Amazon VPC.

2. Scegliere Create Endpoint (Crea endpoint).
3. In Categoria del servizio, assicurati che sia selezionato Servizi AWS.
4. Per Nome servizio, scegli com.amazonaws.***region***.lambda. Verificare che il tipo sia Interfaccia.
5. Creazione di un VPC e delle sottoreti
6. Per abilitare il DNS privato per l'endpoint di interfaccia, in Enable DNS Name (Abilita nome DNS), selezionare la casella di controllo. Ti consigliamo di abilitare nomi host DNS privati per gli endpoint VPC per Servizi AWS. Ciò garantisce che le richieste che utilizzano gli endpoint del servizio pubblico, come le richieste effettuate tramite un SDK AWS, vengano risolte sull'endpoint VPC.
7. Per gruppo di sicurezza, scegliere uno o più gruppi di sicurezza.
8. Seleziona Crea endpoint.

Per utilizzare l'opzione DNS privato, è necessario impostare `enableDnsHostnames` e `enableDnsSupportattributes` del VPC. Per ulteriori informazioni, consulta Visualizzazione e aggiornamento del supporto DNS per il VPC nella Guida per l'utente di Amazon VPC. Se si abilita il DNS privato per l'endpoint di interfaccia, è possibile effettuare richieste API verso Lambda utilizzando il nome DNS predefinito per la regione, ad esempio ., `lambda.us-east-1.amazonaws.com`. Per ulteriori endpoint di servizio, vedere Endpoint e quote del servizio in Riferimenti generali di AWS.

Per ulteriori informazioni, consultare Accesso a un servizio tramite un endpoint di interfaccia in Guida per l'utente di Amazon VPC.

Per informazioni sulla creazione e sulla configurazione di un endpoint utilizzando CloudFormation, consulta la risorsa AWS::EC2::VPCEndpoint nella Guida per l'utente di AWS CloudFormation.

Per creare un endpoint di interfaccia per Lambda (AWS CLI)

Utilizza il comando create-vpc-endpoint e specifica l'ID VPC, il tipo di endpoint VPC (interfaccia), il nome del servizio, le sottoreti per utilizzare l'endpoint e i gruppi di sicurezza da associare alle interfacce di rete dell'endpoint. Ad esempio:

```
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
```

Creazione di una policy degli endpoint di interfaccia per Lambda

Per controllare chi può utilizzare l'endpoint di interfaccia e a quali funzioni Lambda l'utente può accedere, è possibile allegare una policy endpoint all'endpoint. La policy specifica le informazioni riportate di seguito:

- Il principale che può eseguire operazioni.
- Le azioni che l'entità può eseguire.
- Le risorse su cui l'utente/gruppo/ruolo può eseguire azioni.

Per ulteriori informazioni, consulta Controllo degli accessi ai servizi con endpoint VPC nella Guida per l'utente di Amazon VPC.

Esempio: policy endpoint di interfaccia per le azioni Lambda

Di seguito è riportato un esempio di una policy endpoint per Lambda. Quando è collegata a un endpoint, questa policy consente all'utente `MyUser` di richiamare la funzione `my-function`.

> **Note**
>
> Nella risorsa è necessario includere l'ARN della funzione qualificata e non qualificata.

```
{
   "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:*"
            ]
      }
   ]
```

```
}
```

Configurazione dell'accesso al file system per le funzioni Lambda

È possibile configurare una funzione per montare un file system Amazon Elastic File System (Amazon EFS) in una directory locale. Con Amazon EFS, il codice della funzione può accedere e modificare le risorse condivise in modo sicuro e ad alta concorrenza.

Regioni supportate

Amazon EFS per Lambda è disponibile in tutte le regioni commerciali a eccezione di Asia Pacifico (Nuova Zelanda), Asia Pacifico (Taipei), Asia Pacifico (Malesia), Messico (centrale), Asia Pacifico (Thailandia) e Canada occidentale (Calgary).

Sezioni

- Autorizzazioni del ruolo di esecuzione e dell'utente
- Configurazione di un file system e di un punto di accesso
- Connessione a un file system (console)

Autorizzazioni del ruolo di esecuzione e dell'utente

Se il file system non dispone di una policy AWS Identity and Access Management (IAM) configurata dall'utente, EFS utilizza una policy predefinita che consente l'accesso completo a qualsiasi client in grado di connettersi al file system tramite una destinazione di montaggio del file system. Se il file system dispone di una Policy IAM configurata dall'utente, il ruolo di esecuzione della funzione deve disporre delle autorizzazioni `elasticfilesystem` corrette.

Autorizzazioni del ruolo di esecuzione

- elasticfilesystem:ClientMount
- elasticfilesystem:ClientWrite (non richiesto per connessioni di sola lettura)

Queste autorizzazioni sono incluse nella policy gestita di AmazonElasticFileSystemClientReadWriteAccess. Inoltre, il ruolo di esecuzione deve avere le autorizzazioni necessarie per connettersi al VPC del file system.

Quando si configura un file system, Lambda utilizza le autorizzazioni dell'utente per verificare le destinazioni di montaggio. Per configurare una funzione affinché possa connettersi a un file system, l'utente deve disporre delle seguenti autorizzazioni:

Autorizzazioni degli utenti

- elasticfilesystem:DescribeMountTargets

Configurazione di un file system e di un punto di accesso

Creare un file system in Amazon EFS con una destinazione di montaggio in ogni zona di disponibilità a cui la funzione si connette. Per prestazioni e resilienza, utilizzare almeno due zone di disponibilità. Ad esempio, in una configurazione semplice è possibile avere un VPC con due sottoreti private in zone di disponibilità separate. La funzione si connette a entrambe le sottoreti e in ciascuna di esse è disponibile una destinazione di montaggio. Assicurarsi che il traffico NFS (porta 2049) sia consentito dai gruppi di sicurezza utilizzati dalla funzione e dalle destinazioni di montaggio.

> **Note**
>
> Quando si crea un file system, si sceglie una modalità di prestazioni che non può essere modificata in un secondo momento. La modalità Uso generale ha una latenza inferiore e la modalità I/O max supporta throughput e IOPS massimi più elevati. Per assistenza nella scelta, consulta Prestazioni di Amazon EFS nella Amazon Elastic File System User Guide.

Un punto di accesso collega ogni istanza della funzione alla destinazione di montaggio corretta per la zona di disponibilità a cui si connette. Per ottenere prestazioni ottimali, creare un punto di accesso con un percorso non root e limitare il numero di file creati in ogni directory. Nell'esempio seguente viene creata una directory denominata `my-function` nel file system e viene impostato l'ID proprietario su 1001 con autorizzazioni di directory standard (755).

Example Configurazione del punto di accesso

- Nome – `files`
- ID utente – `1001`
- ID gruppo – `1001`
- Percorso – `/my-function`
- Autorizzazioni – `755`
- ID utente proprietario – `1001`
- ID utente gruppo – `1001`

Quando una funzione utilizza il punto di accesso, riceve l'ID utente 1001 e ha accesso completo alla directory.

Per ulteriori informazioni, consulta i seguenti argomenti nella Amazon Elastic File System User Guide.

- Creazione di risorse per Amazon EFS
- Utilizzo di utenti, gruppi e autorizzazioni

Connessione a un file system (console)

Una funzione si connette a un file system tramite la rete locale in un VPC. Le sottoreti a cui si connette la funzione possono essere le stesse sottoreti che contengono punti di montaggio per il file system o sottoreti nella stessa zona di disponibilità che possono instradare il traffico NFS (porta 2049) al file system.

> **Note**
>
> Se la funzione non è già connessa a un VPC, consulta Consentire alle funzioni Lambda l'accesso alle risorse in un Amazon VPC.

Configurazione dell'accesso al file system

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione) e quindi scegliere File systems (File system).
4. In File system, scegliere Aggiungi file system.
5. Configurare le proprietà seguenti:
   - File system EFS: il punto di accesso per un file system nello stesso VPC.
   - Percorso di montaggio locale: la posizione in cui il file system è montato sulla funzione Lambda, a partire da `/mnt/`.

> **Prezzi**
>
> Amazon EFS prevede addebiti per lo storage e il throughput, con tariffe che variano in base alla classe di storage. Per informazioni dettagliate, consulta Prezzi di Amazon EFS.

Lambda prevede addebiti per il trasferimento dei dati tra VPC. Questo vale solo se il VPC della funzione è collegato in peering a un altro VPC con un file system. Le tariffe sono le stesse del trasferimento di dati Amazon EC2 tra VPC nella stessa regione. Per informazioni dettagliate, consulta Prezzi di Lambda.

Creare un alias per una funzione Lambda

È possibile creare alias per la funzione Lambda. Un alias Lambda è un puntatore a una versione della funzione che è possibile aggiornare. Gli utenti possono accedere alla versione della funzione utilizzando l'alias nome della risorsa Amazon (ARN). Quando si distribuisce una nuova versione, è possibile aggiornare l'alias per utilizzare la nuova versione o dividere il traffico tra due versioni.

Console

Per creare un alias tramite la console

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegliere Aliases (Alias) e quindi Create alias (Crea alias).
4. Nella pagina Create alias (Crea alias), eseguire le operazioni seguenti:
   a. Immettere un Name (Nome) per la query.
   b. (Facoltativo) Immettere una Description (Descrizione) per l'alias.
   c. Per Version (Versione), scegliere una versione della funzione a cui si desidera puntare l'alias.
   d. (Facoltativo) Per configurare il routing sull'alias, espandere Weighted alias (Alias ponderato). Per ulteriori informazioni, consulta Implementare le implementazioni canary Lambda utilizzando un alias ponderato.
   e. Seleziona Save (Salva.

AWS CLI

Per creare un alias tramite la AWS Command Line Interface (AWS CLI), utilizza il comando create-alias.

```
aws lambda create-alias \
  --function-name my-function \
  --name alias-name \
  --function-version version-number \
  --description " "
```

Per modificare un alias in modo che punti a una nuova versione della funzione, utilizza il comando update-alias.

```
aws lambda update-alias \
  --function-name my-function \
  --name alias-name \
  --function-version version-number
```

Per eliminare un alias, utilizza il comando delete-alias.

```
aws lambda delete-alias \
  --function-name my-function \
  --name alias-name
```

I comandi AWS CLI nei passaggi precedenti corrispondono alle seguenti operazioni API Lambda:

- CreateAlias
- UpdateAlias
- DeleteAlias

Utilizzo degli alias Lambda nelle origini eventi e nelle policy di autorizzazioni

Ogni alias ha un ARN univoco. Un alias può puntare solo a una versione della funzione, non a un altro alias. È possibile aggiornare un alias in modo che punti a una nuova versione della funzione.

Le origini eventi, ad esempio Amazon Simple Storage Service (Amazon S3), richiamano la funzione Lambda. Queste origini eventi mantengono un mapping che identifica la funzione da richiamare quando si verificano gli eventi. Se si specifica un alias di funzione Lambda nella configurazione del mapping, non è necessario aggiornare il mapping quando cambia la versione della funzione. Per ulteriori informazioni, consulta In che modo Lambda elabora i record provenienti da origini eventi basate su flussi e code.

In una policy di risorsa, è possibile concedere le autorizzazioni per le origini eventi per utilizzare la funzione Lambda. Se si specifica un ARN di alias nella policy, non è necessario aggiornare la policy quando cambia la versione della funzione.

Policy delle risorse

È possibile utilizzare una policy basata sulle risorse per concedere a un servizio, a una risorsa o a un account l'accesso alla funzione. L'ambito di tale autorizzazione dipende dal fatto che venga applicata a un alias, a una versione o all'intera funzione. Ad esempio, se si utilizza un nome alias (ad esempio `helloworld:PROD`), l'autorizzazione consente di richiamare la funzione `helloworld` utilizzando l'ARN dell'alias (`helloworld:PROD`).

Se si tenta di richiamare la funzione senza un alias o una versione specifica, viene visualizzato un errore di autorizzazione. Questo errore di autorizzazione si verifica anche se si tenta di richiamare direttamente la versione della funzione associata all'alias.

Ad esempio, il comando AWS CLI seguente concede le autorizzazioni Amazon S3 per richiamare l'alias PROD della funzione `helloworld` se Amazon S3 agisce per conto di `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
```

Per ulteriori informazioni sull'utilizzo dei nomi delle risorse nelle policy, consulta Ottimizzazione delle sezioni Risorse e Condizioni delle policy.

Implementare le implementazioni canary Lambda utilizzando un alias ponderato

È possibile utilizzare un alias ponderato per suddividere il traffico tra due diverse versioni della stessa funzione. Con questo approccio, puoi testare nuove versioni delle tue funzioni con una piccola percentuale di traffico e ripristinarle rapidamente se necessario. Questa operazione è nota come implementazione canary. Le implementazioni Canary differiscono dalle blue/green distribuzioni in quanto espongono la nuova versione solo a una parte delle richieste anziché spostare tutto il traffico contemporaneamente.

È possibile puntare un alias a un massimo di due versioni della funzione Lambda. Le versioni devono soddisfare i seguenti criteri:

- Entrambe le versioni devono disporre dello stesso ruolo di esecuzione.
- Entrambe le versioni devono avere la stessa configurazione della coda dead-letter o nessuna configurazione della coda dead-letter.
- Entrambe le versioni devono essere pubblicate. L'alias non può puntare a `$LATEST`.

> **Note**
>
> Lambda utilizza un modello probabilistico semplice per distribuire il traffico tra le due versioni delle funzioni. A livelli di traffico bassi, è possibile che si verifichi una variazione elevata tra la percentuale di traffico configurata e quella effettiva in ciascuna versione. Se la tua funzione utilizza la concorrenza con provisioning, puoi evitare invocazioni spillover configurando un numero maggiore di istanze di concorrenza sottoposte a provisioning durante il periodo in cui il routing degli alias è attivo.

Creare un alias ponderato

Console

Per configurare il routing utilizzando la console

> **Note**
>
> Verificare che la funzione abbia almeno due versioni pubblicate. Per creare versioni aggiuntive, seguire le istruzioni in Creazione di versioni delle funzioni.

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegliere Aliases (Alias) e quindi Create alias (Crea alias).
4. Nella pagina Create alias (Crea alias), eseguire le operazioni seguenti:
   a. Immettere un Name (Nome) per la query.
   b. (Facoltativo) Immettere una Description (Descrizione) per l'alias.
   c. Per Version (Versione), scegliere la prima versione della funzione a cui si desidera puntare l'alias.
   d. Espandere Weighted alias (Alias ponderato).
   e. In Additional version (Versione aggiuntiva), scegliere la seconda versione della funzione a cui si desidera puntare l'alias.
   f. Per Weight (%) (Ponderazione %), digitare un valore di ponderazione per la funzione. Weight (Ponderazione) è la percentuale di traffico assegnata alla versione quando l'alias viene invocato. La prima versione riceve la ponderazione residua. Ad esempio se si specifica il 10 percento per Additional version (Versione aggiuntiva), alla prima versione viene automaticamente assegnato il 90 percento.
   g. Scegli Save (Salva).

AWS CLI

Utilizzate i comandi create-alias e AWS CLI update-alias per configurare i pesi del traffico tra due versioni di funzioni. Quando crei o aggiorni l'alias, specifica il peso del traffico nel parametro `routing-config`.

Nell'esempio seguente viene creato un alias della funzione Lambda denominato routing-alias che punta alla versione 1 della funzione. La versione 2 della funzione riceve il 3% del traffico. Il restante 97 percento del traffico viene instradato alla versione 1.

```
aws lambda create-alias \
  --name routing-alias \
  --function-name my-function \
  --function-version 1  \
  --routing-config AdditionalVersionWeights={"2"=0.03}
```

Utilizza il comando `update-alias` per aumentare la percentuale di traffico in ingresso alla versione 2. Nell'esempio seguente, aumenti il traffico al 5%.

```
aws lambda update-alias \
  --name routing-alias \
  --function-name my-function \
  --routing-config AdditionalVersionWeights={"2"=0.05}
```

Per instradare tutto il traffico alla versione 2, utilizza il comando `update-alias` per modificare la proprietà `function-version` in modo che l'alias punti alla versione 2. Il comando reimposta anche la configurazione di routing.

```
aws lambda update-alias \
  --name routing-alias \
  --function-name my-function  \
  --function-version 2 \
  --routing-config AdditionalVersionWeights={}
```

I AWS CLI comandi nei passaggi precedenti corrispondono alle seguenti operazioni dell'API Lambda:

- CreateAlias
- UpdateAlias

Determinazione della versione richiamata

Quando configuri i pesi del traffico tra due versioni di funzioni, esistono due modi per determinare la versione della funzione Lambda invocata:

- CloudWatch Registri: Lambda emette automaticamente START una voce di registro che contiene l'ID di versione richiamato per ogni chiamata di funzione. Esempio:

  ```
  START RequestId: 1dh194d3759ed-4v8b-a7b4-1e541f60235f Version: 2
  ```

  Per le invocazioni di alias Lambda utilizza la dimensione `ExecutedVersion` per filtrare i dati del parametro dalla versione richiamata. Per ulteriori informazioni, consulta Visualizzazione di parametri per le funzioni Lambda.
- Payload della risposta (invocazioni sincrone) - Le risposte a invocazioni sincrone della funzione includono un'intestazione `x-amz-executed-version` per indicare quale versione della funzione è stata invocata.

Creare una implementazione continua con alias ponderati

Usa AWS CodeDeploy and AWS Serverless Application Model (AWS SAM) per creare una distribuzione continua che rilevi automaticamente le modifiche al codice della funzione, distribuisca una nuova versione della funzione e aumenti gradualmente la quantità di traffico che fluisce verso la nuova versione. La quantità di traffico e la velocità dell'aumento sono parametri che è possibile configurare.

In una distribuzione continua, AWS SAM esegue le seguenti attività:

- Configura la funzione Lambda e crea un alias. La configurazione del routing con alias ponderati è la funzionalità che implementa l'implementazione in sequenza.
- Crea un' CodeDeploy applicazione e un gruppo di distribuzione. Il gruppo di distribuzione gestisce l'implementazione in sequenza e il rollback (se necessario).
- Rileva quando viene creata una nuova versione della funzione Lambda.
- Trigger CodeDeploy per avviare la distribuzione della nuova versione.

Modello di esempio AWS SAM

Nell'esempio seguente viene illustrato un modello AWS SAM per una semplice distribuzione in sequenza.

```
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
```

Questo modello definisce una funzione Lambda denominata `myDateTimeFunction` con le seguenti proprietà.

AutoPublishAlias

La proprietà `AutoPublishAlias` crea un alias denominato `live`. Inoltre, il framework AWS SAM rileva automaticamente quando viene salvato il nuovo codice per la funzione. Il framework pubblica quindi una nuova versione di funzione e aggiorna l'alias `live` in modo che punti alla nuova versione.

DeploymentPreference

La `DeploymentPreference` proprietà determina la velocità con cui l' CodeDeploy applicazione sposta il traffico dalla versione originale della funzione Lambda alla nuova versione. Il valore `Linear10PercentEvery2Minutes` sposta un ulteriore dieci percento del traffico alla nuova versione ogni due minuti.

Per l'elenco delle configurazioni di distribuzione predefinite, consulta Configurazioni di distribuzione.

Per ulteriori informazioni su come creare distribuzioni continue con CodeDeploy and AWS SAM, consulta quanto segue:

- Tutorial: Implementa una funzione CodeDeploy Lambda aggiornata con e AWS Serverless Application Model
- Distribuzione graduale di applicazioni serverless con AWS SAM

Gestire le versioni delle funzioni Lambda

Puoi utilizzare le versioni per gestire la distribuzione delle funzioni . Ad esempio, è possibile pubblicare una nuova versione di una funzione per il test beta senza influire sugli utenti della versione di produzione stabile. Lambda crea una nuova versione della funzione ogni volta che pubblichi la funzione. La nuova versione è una copia della versione non pubblicata della funzione. La versione non pubblicata è denominata `$LATEST`.

È importante sottolineare che ogni volta che implementi il codice della funzione, sovrascrivi il codice corrente in `$LATEST`. Per salvare l'iterazione corrente di `$LATEST`, crea una nuova versione della funzione. Se `$LATEST` è identica a una versione pubblicata in precedenza, non potrai creare una nuova versione finché non implementerai le modifiche su `$LATEST`. Queste modifiche possono includere l'aggiornamento del codice o la modifica delle impostazioni di configurazione della funzione.

Dopo aver pubblicato la versione di una funzione, il codice, il runtime, l'architettura, la memoria, i livelli e la maggior parte delle altre impostazioni di configurazione saranno immutabili. Ciò significa che non potrai modificare queste impostazioni senza pubblicare una nuova versione da `$LATEST`. È possibile configurare i seguenti elementi per una versione pubblicata della funzione:

- Trigger
- Destinazioni
- Concorrenza fornita
- Invocazione asincrona
- Connessioni a database e proxy

> **Note**
>
> Quando si utilizzano i controlli di gestione del runtime con la modalità Auto, la versione di runtime utilizzata dalla versione della funzione viene aggiornata automaticamente. Quando si utilizza la modalità Function update (Aggiornamento delle funzioni) o Manual (Manuale), la versione di runtime non viene aggiornata. Per ulteriori informazioni, consulta the section called “Aggiornamenti della versione di runtime”.

Sections

- Creazione di versioni delle funzioni

- Utilizzo delle versioni
- Concessione di autorizzazioni

Creazione di versioni delle funzioni

Puoi modificare il codice della funzione e le impostazioni solo sulla versione non pubblicata di una funzione. Quando pubblichi una versione, il codice e la maggior parte delle impostazioni sono bloccati da Lambda per garantire un'esperienza coerente agli utenti di quella versione.

È possibile creare una versione della funzione utilizzando la console Lambda.

Per creare una nuova versione della funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli una funzione e quindi seleziona la scheda Versioni.
3. Nella pagina di configurazione delle versioni, scegliere Publish new version (Pubblica nuova versione).
4. (Facoltativo) Immettere una descrizione della versione.
5. Seleziona Pubblica.

In alternativa, puoi pubblicare una versione di una funzione utilizzando l'operazione PublishVersionAPI.

Il AWS CLI comando seguente pubblica una nuova versione di una funzione. La risposta restituisce le informazioni di configurazione sulla nuova versione, tra cui il numero della versione e l'ARN della funzione con il suffisso della versione.

```
aws lambda publish-version --function-name my-function
```

Verrà visualizzato l'output seguente:

```
{
  "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 assegna numeri di sequenza crescenti in modo monotono per il controllo delle versioni. Lambda non riutilizza mai i numeri di versione, anche dopo aver eliminato e ricreato una funzione.

Utilizzo delle versioni

Puoi fare riferimento alla tua funzione Lambda usando un ARN qualificato o un ARN non qualificato.

- ARN qualificato – L'ARN della funzione con il suffisso della versione. L'esempio seguente fa riferimento alla versione 42 della funzione `helloworld`.

  ```
  arn:aws:lambda:aws-region:acct-id:function:helloworld:42
  ```

- ARN non qualificato – L'ARN della funzione senza il suffisso della versione.

  ```
  arn:aws:lambda:aws-region:acct-id:function:helloworld
  ```

È possibile utilizzare un ARN qualificato o non qualificato in tutte le operazioni API pertinenti. Tuttavia, non è possibile utilizzare un ARN non qualificato per creare un alias.

Se decidi di non pubblicare le versioni di funzione, puoi richiamare la funzione utilizzando l'ARN qualificato o non qualificato nel mapping dell'origine eventi. Quando richiami una funzione utilizzando un ARN non completo, Lambda richiama implicitamente `$LATEST`.

L'ARN qualificato per ogni versione di funzione Lambda è univoco. Dopo aver pubblicato una versione, non è possibile modificare l'ARN o il codice di funzione.

Lambda pubblica una nuova versione della funzione solo se il codice non è mai stato pubblicato o se il codice è cambiato rispetto all'ultima versione pubblicata. Se non ci sono modifiche, la versione della funzione rimane la versione pubblicata più di recente.

Quando pubblichi una versione, Lambda crea uno snapshot immutabile del codice e della configurazione della funzione. Non tutte le modifiche alla configurazione attivano la pubblicazione

di una nuova versione. Le seguenti modifiche alla configurazione qualificano una funzione per la pubblicazione della versione:

- Codice della funzione
- Variabili di ambiente
- Runtime
- Gestore
- Livelli
- Dimensioni della memoria
- Timeout
- Configurazione VPC
- Configurazione coda DLQ
- Ruolo IAM
- Description
- Architettura (x86_64 o arm64)
- Dimensioni di archiviazione temporanea
- Tipo di pacchetto
- Configurazione della registrazione
- Configurazione del file system
- SnapStart
- Configurazione del tracciamento

Impostazioni operative come la simultaneità riservata non attivano la pubblicazione di una nuova versione quando vengono modificate.

Concessione di autorizzazioni

È possibile utilizzare una policy basata sulle risorse o una policy basata sull'identità per concedere l'accesso alla funzione. L'ambito dell'autorizzazione dipende dal fatto che la policy venga applicata a una funzione o a una versione di una funzione. Per ulteriori informazioni sui nomi delle risorse delle funzioni nelle policy, consulta Ottimizzazione delle sezioni Risorse e Condizioni delle policy.

È possibile semplificare la gestione delle fonti di eventi e delle politiche AWS Identity and Access Management (IAM) utilizzando gli alias delle funzioni. Per ulteriori informazioni, consulta Creare un alias per una funzione Lambda.

Uso dei tag sulle funzioni Lambda

È possibile taggare le funzioni per organizzare e gestire le risorse. I tag sono coppie chiave-valore a forma libera associate alle risorse supportate su Servizi AWS. Per ulteriori informazioni sui casi d'uso dei tag, consulta Strategie di tagging comuni nella Guida AWS alle risorse di etichettatura e all'editor di tag.

I tag si applicano a livello di funzione, non a versioni o alias. I tag non fanno parte della configurazione specifica della versione che AWS Lambda crea un'istantanea di quando si pubblica una versione. Per visualizzare e aggiornare i tag, puoi utilizzare l'API Lambda. Puoi anche visualizzare e aggiornare i tag mentre gestisci una funzione specifica nella console Lambda.

Sections

- Autorizzazioni necessarie per lavorare con i tag
- Utilizzo di tag con la console Lambda
- Usare i tag con AWS CLI

Autorizzazioni necessarie per lavorare con i tag

Per consentire a un'identità AWS Identity and Access Management (IAM) (utente, gruppo o ruolo) di leggere o impostare tag su una risorsa, concedile le autorizzazioni corrispondenti:

- lambda: ListTags —Quando una risorsa ha dei tag, concedi questa autorizzazione a chiunque abbia bisogno di richiamarla. `ListTags` Per le funzioni con tag, questa autorizzazione è necessaria anche per `GetFunction`.
- lambda: TagResource —Concedi questa autorizzazione a chiunque abbia bisogno di chiamare `TagResource` o eseguire un tag durante la creazione.

Facoltativamente, prendi in considerazione la possibilità di concedere anche l'UntagResourceautorizzazione lambda: per consentire `UntagResource` le chiamate alla risorsa.

Per ulteriori informazioni, consulta Policy IAM basate sull'identità per Lambda.

Utilizzo di tag con la console Lambda

Puoi utilizzare la console Lambda per creare funzioni che hanno tag, aggiungere tag a funzioni esistenti e filtrare le funzioni in base ai tag aggiunti.

Per aggiungere tag durante la creazione di una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli Crea funzione.
3. Scegliere Author from scratch (Crea da zero) o Container image (Immagine di container).
4. In Informazioni di base, configura la tua funzione. Per ulteriori informazioni sulla configurazione delle funzioni, consulta Configurazione delle funzioni.
5. Espandi Advanced settings (Impostazioni avanzate), quindi seleziona Enable tags (Abilita tag).
6. Scegli Add new tag (Aggiungi nuovo tag) e completa i campi Key (Chiave) e facoltativamente Value (Valore). Ripetere questa fase per aggiungere altri tag.
7. Scegli Crea funzione.

Per aggiungere tag a una funzione esistente

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Scegli Configuration (Configurazione), quindi Tags (Tag).
4. In Tag, scegli Gestisci tag.
5. Scegli Add new tag (Aggiungi nuovo tag) e completa i campi Key (Chiave) e facoltativamente Value (Valore). Ripetere questa fase per aggiungere altri tag.
6. Scegli Save (Salva).

Per filtrare le funzioni con i tag

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la barra di ricerca per visualizzare un elenco di proprietà della funzione e chiavi di tag.
3. Scegliete una chiave di tag per visualizzare un elenco di valori in uso nella regione corrente. AWS
4. Seleziona Usa: "nome-tag" per vedere tutte funzioni taggate con questa chiave, oppure scegli un operatore per filtrare ulteriormente in base al valore.
5. Seleziona il valore del tag da filtrare in base a una combinazione di chiave e valore del tag.

La barra di ricerca supporta anche la ricerca di chiavi tag. Immetti `tag` per visualizzare solo un elenco di chiavi di tag oppure immetti il nome di una chiave per trovarla nell'elenco.

Usare i tag con AWS CLI

Puoi aggiungere e rimuovere tag sulle risorse Lambda esistenti, incluse le funzioni, con l'API Lambda. Puoi aggiungere i tag anche quando crei una funzione, il che ti consente di mantenere etichettata una risorsa per tutto il suo ciclo di vita.

Aggiornamento dei tag con il tag Lambda APIs

Puoi aggiungere e rimuovere tag per le risorse Lambda supportate tramite le operazioni TagResourcee UntagResourceAPI.

Puoi chiamare queste operazioni tramite la AWS CLI. Per aggiungere i tag a una risorsa esistente, utilizza il comando `tag-resource`. Questo esempio aggiunge due tag, uno con la chiave *`Department`* e uno con la chiave*`CostCenter`*.

```
aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:resource-type:my-resource \
--tags Department=Marketing,CostCenter=1234ABCD
```

Pr rimuovere i tag, utilizza il comando `untag-resource`. Questo esempio rimuove il tag con la chiave*`Department`*.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:resource-
type:resource-identifier \
--tag-keys Department
```

Aggiunta di tag durante la creazione di una funzione

Per creare una nuova funzione Lambda con tag, usa l'operazione CreateFunctionAPI. Specifica il parametro `Tags`. È possibile richiamare questa operazione con il comando della CLI `create-function` e l'opzione --tags. Prima di utilizzare il parametro tags con `CreateFunction`, assicurati che il tuo ruolo disponga dell'autorizzazione per taggare le risorse oltre alle normali autorizzazioni necessarie per questa operazione. Per ulteriori informazioni sulle autorizzazioni per il tagging, consulta the section called “Autorizzazioni necessarie per lavorare con i tag”. Questo esempio aggiunge due tag, uno con la chiave *`Department`* e uno con la chiave*`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
```

Visualizzazione dei tag su una funzione

Per visualizzare i tag applicati a una risorsa Lambda specifica, utilizza l'operazione API `ListTags`. Per ulteriori informazioni, consulta ListTags.

Puoi richiamare questa operazione con il `list-tags` AWS CLI comando fornendo un ARN (Amazon Resource Name).

```
aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:resource-type:resource-identifier
```

Puoi visualizzare i tag applicati a una risorsa specifica con l'operazione GetFunctionAPI. Funzionalità comparabili non sono disponibili per altri tipi di risorse.

È possibile richiamare questa operazione utilizzando il comando `get-function` della CLI:

```
aws lambda get-function --function-name my-function
```

Filtro delle risorse per tag

È possibile utilizzare l'operazione AWS Resource Groups Tagging API GetResourcesAPI per filtrare le risorse in base ai tag. L'operazione `GetResources` riceve fino a 10 filtri, ognuno dei quali contenente una chiave di tag e un massimo di 10 valori di tag. Fornisci `GetResources` con un `ResourceType` per filtrare in base a tipi di risorse specifiche.

È possibile richiamare questa operazione utilizzando il `get-resources` AWS CLI comando. Per esempi di utilizzo di `get-resources`, consulta get-resources nella Riferimento ai comandi CLI di AWS .

Streaming di risposte per le funzioni Lambda

Le funzioni Lambda possono trasmettere nativamente i payload di risposta ai client tramite la funzione Lambda URLs o utilizzando l'InvokeWithResponseStreamAPI (tramite l' AWS SDK o le chiamate API dirette). La tua funzione Lambda può anche trasmettere i payload di risposta tramite l'integrazione proxy di Amazon API Gateway, che utilizza l'InvokeWithResponseStreamAPI per richiamare la tua funzione. Lo streaming delle risposte può favorire le applicazioni sensibili alla latenza migliorando le prestazioni del time to first byte (TTFB). Questo perché consente di inviare risposte parziali al client non appena diventano disponibili. Inoltre, le funzioni di streaming di risposta possono restituire payload fino a 200 MB, rispetto al massimo di 6 MB per le risposte bufferizzate. Lo streaming di una risposta significa anche che la funzione non deve contenere l'intera risposta in memoria. Per risposte molto grandi, ciò può ridurre la quantità di memoria necessaria per configurare la funzione.

> **Note**
>
> Lo streaming di risposta Lambda non è ancora disponibile in tutti. Regioni AWS Per informazioni sulla disponibilità delle funzionalità per regione, consulta AWS le funzionalità di Builder Center.

La velocità con cui Lambda trasmette le tue risposte dipende dalla dimensione della risposta. La velocità di streaming per i primi 6 MB di risposta della funzione è illimitata. Per le risposte superiori a 6 MB, il resto della risposta è soggetto a un limite di larghezza di banda. Per ulteriori informazioni sulla larghezza di banda dello streaming, consulta la sezione Limiti di larghezza di banda per lo streaming delle risposte.

Le risposte in streaming comportano costi e le risposte in streaming non vengono interrotte o interrotte quando la connessione del client che effettua la chiamata viene interrotta. Ai clienti verrà fatturata l'intera durata della funzione, quindi i clienti devono prestare attenzione quando configurano lunghi timeout di funzionamento.

Lambda supporta lo streaming delle risposte sui runtime gestiti di Node.js. Per altri linguaggi, incluso Python, puoi utilizzare un runtime personalizzato con un'integrazione API Runtime personalizzata per lo streaming delle risposte o utilizzare il Lambda Web Adapter.

> **Note**
>
> Quando testi la funzione tramite la console Lambda, vedrai sempre le risposte come memorizzate nel buffer.

Argomenti

- Limiti di larghezza di banda per lo streaming delle risposte
- Compatibilità VPC con streaming di risposta
- Scrittura di funzioni Lambda abilitate allo streaming delle risposte
- Richiamo di una funzione abilitata allo streaming di risposte utilizzando la funzione Lambda URLs
- Tutorial: creazione di una funzione Lambda di streaming delle risposte con un URL della funzione

Limiti di larghezza di banda per lo streaming delle risposte

I primi 6 MB del payload di risposta della funzione hanno una larghezza di banda illimitata. Dopo questa raffica iniziale, Lambda trasmette la tua risposta a una velocità massima di 2. MBps Se le risposte delle tue funzioni non superano mai i 6 MB, questo limite di larghezza di banda non verrà mai applicato.

> **Note**
>
> I limiti di larghezza di banda si applicano solo al payload di risposta della funzione e non all'accesso alla rete da parte della funzione.

La velocità della larghezza di banda illimitata varia in base a una serie di fattori, inclusa la velocità di elaborazione della funzione. Normalmente puoi aspettarti una frequenza superiore a 2 MBps per i primi 6 MB di risposta della funzione. Se la tua funzione trasmette in streaming una risposta a una destinazione esterna AWS, la velocità di streaming dipende anche dalla velocità della connessione Internet esterna.

Compatibilità VPC con streaming di risposta

Quando utilizzi le funzioni Lambda in un ambiente VPC, per lo streaming di risposta è importante tenere in considerazione quanto segue:

- La funzione Lambda URLs non supporta lo streaming di risposte all'interno di un ambiente VPC.
- Puoi utilizzare lo streaming di risposte all'interno di un VPC richiamando la funzione Lambda tramite l'SDK utilizzando l' AWS API. `InvokeWithResponseStream` Ciò richiede la configurazione degli endpoint VPC appropriati per Lambda.
- Per gli ambienti VPC, dovrai creare un endpoint VPC di interfaccia per Lambda per abilitare la comunicazione tra le tue risorse nel VPC e il servizio Lambda.

Un'architettura tipica per lo streaming di risposta in un VPC potrebbe includere:

```
Client in VPC -> Interface VPC endpoint for Lambda -> Lambda function -> Response
 streaming back through the same path
```

Scrittura di funzioni Lambda abilitate allo streaming delle risposte

La scrittura del gestore per le funzioni di streaming delle risposte è diversa dai modelli di gestore tipici. Quando scrivi funzioni di streaming, assicurati di completare le seguenti operazioni:

- Completa la funzione con il decoratore `awslambda.streamifyResponse()`. L'oggetto globale `awslambda` è fornito dall'ambiente di runtime Node.js di Lambda.
- Termina il flusso in modo corretto per assicurarti che tutta l'elaborazione dei dati sia completa.

Configurazione di un gestore delle funzioni per lo streaming delle risposte

Per indicare al runtime che Lambda deve trasmettere in streaming le risposte della funzione, è necessario racchiudere la funzione con il decoratore `streamifyResponse()`. Questo indica al runtime di utilizzare il percorso logico corretto per lo streaming delle risposte e consente alla funzione di trasmettere le risposte.

Il decoratore `streamifyResponse()` accetta una funzione che accetta i seguenti parametri:

- `event`: fornisce informazioni sull'evento di chiamata dell'URL della funzione, ad esempio il metodo HTTP, i parametri della query e il corpo della richiesta.
- `responseStream`: fornisce un flusso scrivibile.
- `context`: fornisce i metodi e le proprietà con informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

L'oggetto `responseStream` è un `writableStream` Node.js. Come con qualsiasi flusso di questo tipo, dovresti usare il metodo `pipeline()`.

> **Note**
>
> L'oggetto globale `awslambda` viene fornito automaticamente dal runtime Node.js di Lambda e non è richiesta alcuna importazione.

Example gestore abilitato allo streaming delle risposte

```
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);
});
```

Sebbene `responseStream` offra il metodo `write()` per scrivere sul flusso, ti consigliamo di utilizzare `pipeline()`laddove possibile. L'utilizzo di `pipeline()` garantisce che il flusso scrivibile non venga sopraffatto da un flusso leggibile più veloce.

Terminazione dello streaming

Assicurati di terminare correttamente il flusso prima che torni al gestore. Il metodo `pipeline()` gestisce questo aspetto automaticamente.

Per altri casi d'uso, chiama il metodo `responseStream.end()` per terminare correttamente un flusso. Questo metodo segnala che nel flusso non devono essere scritti altri dati. Questo metodo non è necessario se si scrive nel flusso con `pipeline()` o `pipe()`.

A partire da Node.js 24, Lambda non attende più che le promesse non risolte vengano completate dopo il ritorno del gestore o il termine del flusso di risposta. Se la tua funzione dipende da operazioni asincrone aggiuntive, come timer o recuperi, dovresti inserirle nel tuo gestore. `await`

Example Esempio di terminazione di un flusso con pipeline()

```
import { pipeline } from 'node:stream/promises';

export const handler = awslambda.streamifyResponse(async (event, responseStream,
 _context) => {
  await pipeline(requestStream, responseStream);
});
```

Example Esempio di terminazione di un flusso senza pipeline()

```
export const handler = awslambda.streamifyResponse(async (event, responseStream,
 _context) => {
  responseStream.write("Hello ");
  responseStream.write("world ");
  responseStream.write("from ");
  responseStream.write("Lambda!");
  responseStream.end();
});
```

Richiamo di una funzione abilitata allo streaming di risposte utilizzando la funzione Lambda URLs

> **Note**
>
> La tua funzione Lambda può ora trasmettere in streaming i payload di risposta tramite l'integrazione proxy di Amazon API Gateway.

Puoi richiamare le funzioni abilitate allo streaming delle risposte modificando la modalità di richiamo dell'URL della funzione. La modalità di richiamo determina quale operazione API Lambda utilizza per richiamare la funzione. Le modalità di richiamo disponibili sono:

- `BUFFERED`: questa è l'opzione predefinita. Lambda richiama la funzione utilizzando l'operazione API `Invoke`. I risultati delle chiamate sono disponibili quando il payload è completo. La dimensione massima del payload è pari a 6 MB.
- `RESPONSE_STREAM`: consente alla funzione di trasmettere in streaming i risultati del payload non appena diventano disponibili. Lambda richiama la funzione utilizzando l'operazione API `InvokeWithResponseStream`. La dimensione massima del payload di risposta è di 200 MB.

Puoi comunque richiamare la funzione senza lo streaming delle risposte chiamando direttamente l'operazione API `Invoke`. Tuttavia, Lambda trasmette in streaming tutti i payload di risposta per le chiamate che arrivano tramite l'URL della funzione fino a quando non si modifica la modalità di richiamo in `BUFFERED`.

Console

Creazione di un URL della funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione per la quale desideri impostare la modalità di richiamo.
3. Scegli la scheda Configurazione, quindi scegli URL della funzione.
4. Scegli Modifica, quindi scegli Impostazioni aggiuntive.
5. In Modalità di richiamo, scegli la modalità di richiamo desiderata.
6. Scegli Save (Salva).

AWS CLI

Per impostare la modalità di richiamo di un URL della funzione (AWS CLI)

```
aws lambda update-function-url-config \
  --function-name my-function \
  --invoke-mode RESPONSE_STREAM
```

CloudFormation

Per impostare la modalità di richiamo di un URL della funzione (CloudFormation)

```
MyFunctionUrl:
  Type: AWS::Lambda::Url
  Properties:
    AuthType: AWS_IAM
    InvokeMode: RESPONSE_STREAM
```

Per ulteriori informazioni sulla configurazione della funzione URLs, vedere Funzione URLsLambda.

Tutorial: creazione di una funzione Lambda di streaming delle risposte con un URL della funzione

In questo tutorial viene creata una funzione Lambda definita come archivio file .zip con un endpoint funzione URL che restituisce un flusso di risposte. Per ulteriori informazioni sulla configurazione della funzione URLs, vedereFunzione URLs.

Prerequisiti

Questo tutorial presuppone una certa conoscenza delle operazioni di base di Lambda e della console relativa. Se non lo si è già fatto, seguire le istruzioni riportate in Creare una funzione Lambda con la console per creare la prima funzione Lambda.

Per completare i passaggi seguenti, è necessaria la AWS CLI versione 2. I comandi e l'output previsto sono elencati in blocchi separati:

```
aws --version
```

Verrà visualizzato l'output seguente:

```
aws-cli/2.13.27 Python/3.11.6 Linux/4.14.328-248.540.amzn2.x86_64 exe/x86_64.amzn.2
```

Per i comandi lunghi viene utilizzato un carattere di escape (\) per dividere un comando su più righe.

In Linux e macOS utilizzare la propria shell e il proprio programma di gestione dei pacchetti preferiti.

> **Note**
>
> Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere una versione integrata su Windows di Ubuntu e Bash, installa il sottosistema Windows per Linux. I comandi della CLI di esempio in questa guida utilizzano la formattazione Linux. Se si utilizza la CLI di Windows, i comandi che includono documenti JSON in linea dovranno essere riformattati.

Creazione di un ruolo di esecuzione

Creare il ruolo di esecuzione che offre l'autorizzazione alla funzione Lambda per accedere alle risorse AWS .

Creazione di un ruolo di esecuzione

1. Aprire la pagina Roles (Ruoli) della console IAM AWS Identity and Access Management .
2. Scegli Crea ruolo.
3. Creare un ruolo con le seguenti proprietà:
   - Tipo di entità affidabile: servizio di AWS
   - Caso d'uso: Lambda
   - Autorizzazioni: AWSLambdaBasicExecutionRole
   - Nome ruolo – **`response-streaming-role`**

La AWSLambdaBasicExecutionRolepolicy dispone delle autorizzazioni necessarie alla funzione per scrivere log su Amazon CloudWatch Logs. Una volta creato il ruolo, prendi nota del relativo nome della risorsa Amazon (ARN). Questo valore servirà nella fase successiva.

Creazione di una funzione di streaming delle risposte (AWS CLI)

Crea una funzione Lambda di streaming delle risposte con un endpoint URL della funzione utilizzando l' AWS Command Line Interface (AWS CLI).

Creazione di una funzione in grado di trasmettere le risposte

1. Copiare il codice di esempio seguente in un file denominato `index.js`. Questa funzione trasmette tre risposte, separate da 1 secondo.

```
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. Crea un pacchetto di implementazione.

```
zip function.zip index.js
```

3. Creare una funzione Lambda con il comando `create-function`. Sostituisci il valore di `--role` con l'ARN del ruolo del passaggio precedente. Questo comando imposta il timeout della funzione su 10 secondi, il che consente alla funzione di trasmettere tre risposte.

```
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
```

Creazione di un URL della funzione

1. Aggiungi alla tua funzione una politica basata sulle risorse che concede e autorizzazioni. `lambda:InvokeFunctionUrl` `lambda:InvokeFunction` Ogni istruzione deve essere aggiunta in un comando separato. Sostituisci il valore di `--principal` con il tuo Account AWS ID.

```
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. Crea un endpoint URL per la funzione con il comando `create-function-url-config`.

```
aws lambda create-function-url-config \
  --function-name my-streaming-function \
  --auth-type AWS_IAM \
  --invoke-mode RESPONSE_STREAM
```

> **Note**
>
> Se ricevi un errore in merito`--invoke-mode`, potrebbe essere necessario eseguire l'aggiornamento a una versione più recente di AWS CLI.

Verifica l'endpoint URL della funzione

Testa l'integrazione richiamando la tua funzione. Puoi aprire l'URL della funzione in un browser oppure puoi usare 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
```

La nostra funzione URL utilizza il tipo di autenticazione `IAM_AUTH`. Ciò significa che è necessario firmare le richieste sia con la chiave di AWS accesso che con la chiave segreta. Nel comando precedente, sostituiscilo `AKIAIOSFODNN7EXAMPLE` con l'ID della chiave di AWS accesso. Inserisci

la tua chiave AWS segreta quando richiesto. Se non disponi della chiave AWS segreta, puoi invece utilizzare AWS credenziali temporanee.

Dovresti ottenere una risposta simile a questa:

```
Streaming with Helper
Hello 0
Hello 1
Hello 2
```

Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili a tuo carico. Account AWS

Come eliminare il ruolo di esecuzione

1. Aprire la pagina Ruoli della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

Per eliminare la funzione Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Digita **`confirm`** nel campo di immissione testo e scegli Delete (Elimina).

Comprensione dei metodi di invocazione della funzione Lambda

Dopo aver implementato la tua funzione Lambda, puoi richiamarla in diversi modi:

- La console Lambda: utilizza la console Lambda per creare rapidamente un evento di test per richiamare la tua funzione.
- L'AWS SDK: usa l'SDK per richiamare la AWS tua funzione a livello di codice.
- L'API Invoke: utilizza l'API Lambda Invoke per richiamare direttamente la tua funzione.
- The AWS Command Line Interface (AWS CLI): utilizzate il `aws lambda invoke` AWS CLI comando per richiamare direttamente la funzione dalla riga di comando.
- Un endpoint HTTP (S) URL di una funzione: utilizza la funzione URLs per creare un endpoint HTTP (S) dedicato che puoi utilizzare per richiamare la tua funzione.

Tutti questi metodi sono modi diretti per richiamare la tua funzione. In Lambda, un caso d'uso comune consiste nel richiamare la funzione in base a un evento che si verifica altrove nell'applicazione. Alcuni servizi possono richiamare una funzione Lambda con ogni nuovo evento. Si chiama trigger. Per i servizi basati su flussi e code, Lambda richiama la funzione con batch di record. Ciò è detto strumento di mappatura dell'origine degli eventi.

Quando si invoca una funzione, è possibile scegliere di invocarla in modo sincrono o asincrono. Con l'invocazione sincrona, è necessario attendere che la funzione elabori l'evento e restituisca una risposta. Con l'invocazione asincrona, Lambda accoda l'evento per l'elaborazione e restituisce una risposta immediatamente. Il parametro della richiesta `InvocationType` nell'API Invoke determina il modo in cui Lambda richiama la tua funzione. Un valore di `RequestResponse` indica una invocazione sincrona mentre un valore `Event` indica una invocazione asincrona.

Per richiamare la tua funzione IPv6, usa gli endpoint dual-stack pubblici di Lambda. Gli endpoint dual-stack supportano entrambi e. IPv4 IPv6 Gli endpoint dual-stack Lambda utilizzano la seguente sintassi:

```
protocol://lambda.us-east-1.api.aws
```

Puoi anche usare la funzione Lambda URLs per richiamare funzioni. IPv6 Gli endpoint URL della funzione hanno il formato seguente:

```
https://url-id.lambda-url.us-east-1.on.aws
```

Se l'invocazione della funzione genera un errore, per le invocazioni sincrone, visualizza il messaggio di errore nella risposta e riprova a richiamare manualmente. Per l'invocazione asincrona, Lambda gestisce i nuovi tentativi e può inviare i record di invocazione a una destinazione.

Richiamare una funzione Lambda in modo sincrono

Quando si invoca una funzione in modo sincrono, Lambda esegue la funzione e attende una risposta. Quando l'esecuzione di una funzione termina, Lambda restituisce la risposta dal codice della funzione con dati aggiuntivi, ad esempio la versione della funzione invocata. Per invocare una funzione in modo sincrono con l'AWS CLI, utilizzare il comando `invoke`.

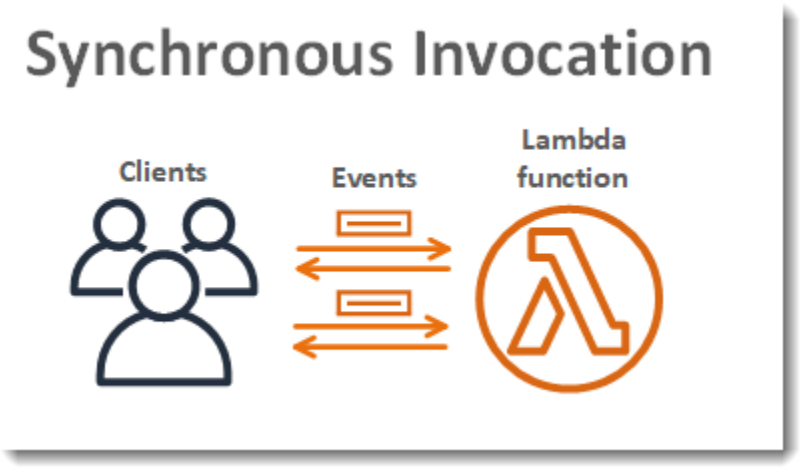
```
aws lambda invoke --function-name my-function \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "key": "value" }' response.json
```

L'opzione cli-binary-format è necessaria se si utilizza la versione 2 della AWS CLI. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
{
    "ExecutedVersion": "$LATEST",
    "StatusCode": 200
}
```

Il seguente diagramma mostra i client che richiamano una funzione Lambda in modo sincrono. Lambda invia gli eventi direttamente alla funzione e invia la risposta della funzione all'invoker.



Il `payload` è una stringa che contiene un evento in formato JSON. Il nome del file in cui l'AWS CLI scrive la risposta dalla funzione è `response.json`. Se la funzione restituisce un oggetto o un errore,

il corpo della risposta è l'oggetto o l'errore in formato JSON. Se la funzione termina senza errori, il corpo della risposta è `null`.

> **Note**
>
> Lambda non attende il completamento delle estensioni esterne prima di inviare la risposta. Le estensioni esterne vengono eseguite come processi indipendenti nell'ambiente di esecuzione e continuano l'esecuzione dopo che la chiamata della funzione è stata completata. Per ulteriori informazioni, consulta Aumentare le funzioni Lambda utilizzando le estensioni Lambda.

L'output del comando, che viene visualizzato nel terminale, include informazioni dalle intestazioni nella risposta da Lambda. Sono incluse la versione che ha elaborato l'evento (utile quando si utilizzano gli alias) e il codice di stato restituito da Lambda. Se Lambda è stato in grado di eseguire la funzione, il codice di stato è 200, anche se la funzione ha restituito un errore.

> **Note**
>
> Per le funzioni con un lungo timeout, il client potrebbe essere scollegato durante l'invocazione sincrona mentre è in attesa della risposta. Configurare il client HTTP, l'SDK, il firewall o il sistema operativo per consentire le connessioni lunghe con timeout o le impostazioni keep-alive.

Se Lambda non è in grado di eseguire la funzione, l'errore viene visualizzato nell'output.

```
aws lambda invoke --function-name my-function \
    --cli-binary-format raw-in-base64-out \
    --payload value response.json
```

Verrà visualizzato l'output seguente:

```
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 è uno strumento open source che consente di interagire con i servizi AWS tramite i comandi nella shell a riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della AWS CLI versione 2.

È possibile utilizzare AWS CLI per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'`base64` utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `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
```

L'opzione cli-binary-format è necessaria se si utilizza la versione 2 della AWS CLI. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
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
```

L'utilità `base64` è disponibile su Linux, macOS e Ubuntu su Windows. Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

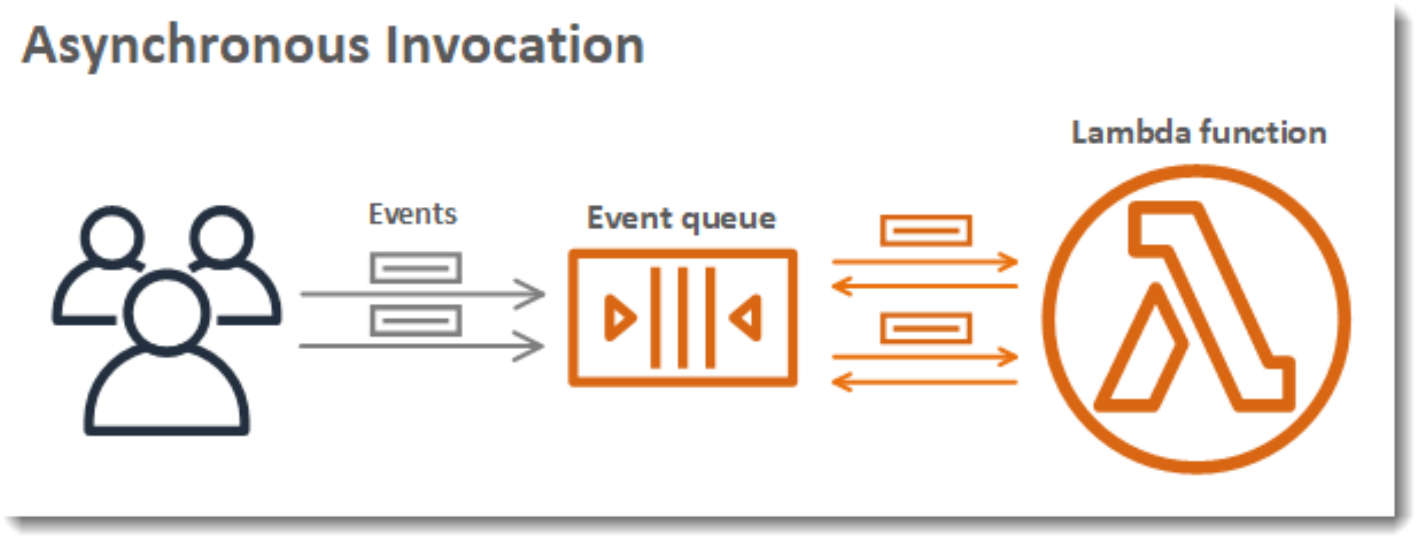
Per ulteriori informazioni sulle API `Invoke`, incluso un elenco completo di parametri, intestazioni ed errori, consulta Invoke (Invoca).

Quando si invoca una funzione direttamente, è possibile controllare la risposta agli errori e riprovare. L'AWS CLI e l'SDK AWS, inoltre, riprovano automaticamente sugli errori di timeout del client, di throttling e di servizio. Per ulteriori informazioni, consulta Informazioni sul comportamento relativo ai nuovi tentativi in Lambda.

Richiamo di una funzione Lambda in modo sincrono

Diversi Servizi AWS, come Amazon Simple Storage Service (Amazon S3) e Amazon Simple Notification Service (Amazon SNS), chiamano le funzioni in modo asincrono per elaborare gli eventi. Puoi anche richiamare una funzione Lambda in modo asincrono utilizzando la AWS Command Line Interface (AWS CLI) o uno degli SDK AWS. Quando si richiama una funzione in modo asincrono, non si attende una risposta dal codice della funzione. Si passa l'evento a Lambda e Lambda si occupa del resto. È possibile configurare il modo in cui Lambda gestisce gli errori e invia i record di invocazione a una risorsa a valle come Amazon Simple Queue Service (Amazon SQS) o Amazon EventBridge (EventBridge) per concatenare i componenti dell'applicazione.

Il seguente diagramma mostra i client che richiamano una funzione Lambda in modo asincrono. Lambda accoda gli eventi prima di inviarli alla funzione.



Per l'invocazione asincrona, Lambda inserisce l'evento in una coda e restituisce una risposta di esito positivo senza ulteriori informazioni. Un processo separato legge gli eventi dalla coda ed esegue la funzione.

Per richiamare una funzione Lambda in modo asincrono utilizzando la AWS Command Line Interface (AWS CLI) o uno degli SDK AWS, imposta il parametro InvocationType su `Event`. L'esempio di codice seguente mostra un comando AWS CLI per invocare una funzione.

```
aws lambda invoke \
  --function-name my-function  \
  --invocation-type Event \
  --cli-binary-format raw-in-base64-out \
```

```
  --payload '{ "key": "value" }' response.json
```

Verrà visualizzato l'output seguente:

```
{
    "StatusCode": 202
}
```

L'opzione cli-binary-format è necessaria se si utilizza la versione 2 della AWS CLI. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Il file di output (`response.json`) non contiene informazioni, ma è comunque creato quando si esegue il comando. Se Lambda non è in grado di aggiungere l'evento alla coda, il messaggio di errore viene visualizzato nell'output del comando.

In che modo Lambda gestisce gli errori e i nuovi tentativi con una invocazione asincrona

Lambda gestisce la coda di eventi asincroni della funzione ed effettua nuovi tentativi in caso di errori. Se la funzione restituisce un errore, per impostazione predefinita Lambda prova a eseguirla altre due volte, con un minuto di attesa tra i primi due tentativi e due minuti tra il secondo e il terzo. Gli errori di funzione includono gli errori restituiti dal codice della funzione e gli errori restituiti dal runtime della funzione, ad esempio timeout.

Se la funzione non dispone di sufficiente concorrenza per elaborare tutti gli eventi, ulteriori richieste saranno sottoposte a throttling. Per gli errori di limitazione (429) e gli errori di sistema (serie 500), Lambda restituisce l'evento alla coda e prova a eseguire nuovamente la funzione per un massimo di 6 ore. L'intervallo tra i tentativi aumenta esponenzialmente da 1 secondo dopo il primo tentativo a un massimo di 5 minuti. Se la coda contiene molte voci, Lambda aumenta l'intervallo dei tentativi e riduce la velocità con cui legge gli eventi dalla coda.

Anche se la funzione non restituisce un errore, è possibile che riceva lo stesso evento da Lambda più volte perché la coda stessa alla fine è coerente. Se la funzione non è in grado di seguire gli eventi in entrata, gli eventi possono anche essere eliminati dalla coda senza essere inviati alla funzione. Verificare che il codice della funzione gestisca normalmente gli eventi duplicati e che si disponga di sufficiente concorrenza per gestire tutte le invocazioni.

Quando la coda è molto lunga, i nuovi eventi potrebbero diventare datati prima che Lambda abbia la possibilità di inviarli alla funzione. Quando un evento scade o fallisce tutti i tentativi di elaborazione, Lambda lo scarta. È possibile configurare la gestione degli errori per una funzione per ridurre il numero di tentativi eseguiti da Lambda o per eliminare più rapidamente gli eventi non elaborati. Configurare una coda DLQ per conservare gli eventi scartati. Per registrare le registrazioni delle chiamate non riuscite (ad esempio timeout o errori di runtime), create una destinazione in caso di errore.

Configurazione delle impostazioni di gestione degli errori per le invocazioni asincrone Lambda

Usa le seguenti impostazioni per configurare il modo in cui Lambda gestisce gli errori e i nuovi tentativi per le invocazioni di funzioni asincrone:

- MaximumEventageInSeconds: il tempo massimo in secondi per cui Lambda conserva un evento nella coda degli eventi asincroni prima di eliminarlo.
- MaximumRetryAttempts: il numero massimo di tentativi che Lambda effettua quando la funzione restituisce un errore.

Utilizza la console Lambda o AWS CLI per configurare le impostazioni di gestione degli errori per una funzione, una versione o un alias.

Console

Per configurare la gestione degli errori

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione), quindi scegli Asynchronous invocation (Chiamata asincrona).
4. In Asynchronous invocation (Chiamata asincrona), scegliere Edit (Modifica).
5. Configura le impostazioni seguenti.
   - Maximum age of event (Età massima dell'evento): il tempo massimo per cui Lambda conserva un evento nella coda degli eventi asincroni, fino a 6 ore.
   - Retry attempts (Nuovi tentativi): il numero di tentativi che Lambda effettua quando la funzione restituisce un errore, tra 0 e 2.

6. Selezionare Salva.

AWS CLI

Per configurare l'invocazione asincrona con la AWS CLI, utilizza il comando put-function-event-invoke-config. Nell'esempio seguente viene configurata una funzione con una durata massima dell'evento di 1 ora e senza ulteriori tentativi.

```
aws lambda put-function-event-invoke-config \
  --function-name error \
  --maximum-event-age-in-seconds 3600 \
  --maximum-retry-attempts 0
```

Il comando `put-function-event-invoke-config` sovrascrive qualsiasi configurazione esistente sulla funzione, versione o alias. Per configurare un'opzione senza reimpostarne altre, utilizza update-function-event-invoke-config. Nell'esempio seguente Lambda viene configurato per inviare un record a una coda SQS standard denominata `destination` quando non è possibile elaborare un evento.

```
aws lambda update-function-event-invoke-config \
  --function-name my-function \
  --destination-config '{"OnFailure":{"Destination": "arn:aws:sqs:us-
east-1:123456789012:destination"}}'
```

Verrà visualizzato l'output seguente:

```
{
    "LastModified": 1573686021.479,
    "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function:
$LATEST",
    "MaximumRetryAttempts": 0,
    "MaximumEventAgeInSeconds": 3600,
    "DestinationConfig": {
        "OnSuccess": {},
        "OnFailure": {}
    }
}
```

Quando un evento di invocazione supera l'età massima o fallisce tutti i nuovi tentativi, Lambda lo scarta. Per conservare una copia degli eventi eliminati, configurare una destinazione dell'evento non riuscito.

Acquisizione dei record delle invocazioni asincrone Lambda

Lambda può inviare record di chiamate asincrone a uno dei seguenti. Servizi AWS

- Amazon SQS: una coda SQS standard
- Amazon SNS: un argomento SNS standard
- Amazon S3: un bucket Amazon S3 (solo in caso di errore)
- AWS Lambda: una funzione Lambda
- Amazon EventBridge: un bus per EventBridge eventi

Il record di invocazione contiene dettagli sulla richiesta e la risposta in formato JSON. È possibile configurare destinazioni separate per gli eventi che vengono elaborati correttamente e per quelli che restituiscono un errore a ogni tentativo di elaborazione. In alternativa, è possibile configurare una coda standard di Amazon SQS o un argomento standard di Amazon SNS come coda DLQ per gli eventi scartati. Per le code DLQ, Lambda invia solo il contenuto dell'evento, senza dettagli sulla risposta.

Se Lambda non è in grado di inviare un record a una destinazione che hai configurato, invia una `DestinationDeliveryFailures` metrica ad Amazon. CloudWatch Ciò può verificarsi se la configurazione include un tipo di destinazione non supportato, ad esempio una coda FIFO di Amazon SQS o un argomento FIFO di Amazon SNS. Gli errori di recapito possono verificarsi anche a causa di errori di autorizzazioni e limiti di dimensione. Per ulteriori informazioni sui parametri di invocazione Lambda, consulta the section called “Parametri di invocazione”.

> **Note**
>
> Per impedire l'attivazione di una funzione, è possibile impostare la simultaneità riservata della funzione su zero. Quando si imposta la simultaneità riservata su zero per una funzione chiamata in modo asincrono, Lambda inizia a inviare i nuovi eventi alla coda DLQ configurata o alla destinazione degli eventi in caso di errore, senza nuovi tentativi. Per elaborare gli eventi inviati mentre la simultaneità riservata era impostata su zero, è necessario utilizzare gli eventi dalla coda DLQ o dalla destinazione degli eventi in caso di errore.

Aggiunta di una destinazione

Per mantenere i record delle chiamate asincrone, aggiungi una destinazione alla funzione. È possibile scegliere di inviare a una destinazione le chiamate riuscite o non riuscite. Ogni funzione può avere più destinazioni, quindi è possibile configurare destinazioni separate per eventi riusciti e non riusciti. Ogni record inviato alla destinazione è un documento JSON con i dettagli relativi alla chiamata. Come per le impostazioni di gestione degli errori, è possibile impostare le destinazioni su una funzione, una versione della funzione o un alias.

> **Tip**
>
> Puoi anche conservare i record delle chiamate non riuscite per i seguenti tipi di mappatura delle sorgenti degli eventi: Amazon Kinesis, Amazon DynamoDB e Apache Kafka (Amazon MSK e Apache Kafka autogestito).

La tabella seguente elenca le destinazioni supportate per i record di chiamata asincrona. Affinché Lambda invii correttamente i record alla destinazione prescelta, assicurati che il ruolo di esecuzione della funzione disponga anche delle autorizzazioni pertinenti. La tabella descrive anche il modo in cui ogni tipo di destinazione riceve il record di chiamata JSON.

Tipo di destinazione	Autorizzazione richiesta	Formato JSON specifico della destinazione
Coda Amazon SQS	come: SendMessage	Lambda passa il record di chiamata come `Message` alla destinazione.
Argomento Amazon SNS	sns:Publish	Lambda passa il record di chiamata come `Message` alla destinazione.
Bucket Amazon S3 (solo in caso di errore)	s3: PutObject s3: ListBucket	• Lambda archivia il record di invocazione come oggetto JSON nel bucket di destinazione.

Tipo di destinazione	Autorizzazione richiesta	Formato JSON specifico della destinazione
		• Il nome dell'oggetto S3 utilizza la seguente convenzione di denominazione: `aws/lambda/async/<function-name>/YYYY/MM/DD/YYYY-MM-DDTHH.MM.SS-<Random UUID>`
funzione Lambda	lambda: InvokeFunction	Lambda passa il record di chiamata come payload alla funzione.

Tipo di destinazione	Autorizzazione richiesta	Formato JSON specifico della destinazione
EventBridge	eventi: PutEvents	• Lambda passa il record di invocazione come nella `detail` chiamata. PutEvents • Il valore per il campo di eventi `source` è `aws.lambda` . • Il valore per il campo dell'evento `detail-type` è "Risultato della chiamata della funzione Lambda - Successo" o "Risultato della chiamata della funzione Lambda - Errore". • Il campo `resource` dell'evento contiene la funzione e la destinazione Amazon Resource Names (ARNs). • Per altri campi relativi agli eventi, consulta Amazon EventBridge events.

> **Note**
>
> Per le destinazioni Amazon S3, se hai abilitato la crittografia sul bucket utilizzando una chiave KMS, la tua funzione necessita anche dell'autorizzazione kms:. GenerateDataKey

> **Important**
>
> Quando usi Amazon SNS come destinazione, tieni presente che Amazon SNS ha un limite massimo di dimensione dei messaggi di 256 KB. Se il payload di chiamata asincrona si avvicina a 1 MB, il record di chiamata (che include il payload originale più metadati aggiuntivi) potrebbe superare il limite di Amazon SNS e causare errori di consegna. Prendi in considerazione l'utilizzo di destinazioni Amazon SQS o Amazon S3 per payload più grandi.

La procedura seguente descrive come configurare una destinazione per una funzione che utilizza la console Lambda e la AWS CLI.

Console

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. In Function overview (Panoramica delle funzioni), scegliere Add destination (Aggiungi destinazione).
4. Per Source (Origine), scegliere Asynchronous invocation (Chiamata asincrona).
5. Per Condition (Condizione) scegliere tra le seguenti opzioni:
   - In caso di errore: viene inviato un record quando l'evento fallisce tutti i tentativi di elaborazione o supera l'età massima.
   - On success (In caso di esito positivo): viene inviato un record quando la funzione elabora correttamente un'invocazione asincrona.
6. Per Destination type (Tipo di destinazione), scegliere il tipo di risorsa che riceve il record di invocazione.
7. Per Destination (Destinazione), scegliere una risorsa.
8. Scegli Save (Salva).

AWS CLI

Per configurare una destinazione utilizzando AWS CLI, esegui il comando -config. update-function-event-invoke Nell'esempio seguente Lambda viene configurato per inviare un record a una coda SQS standard denominata `destination` quando non è possibile elaborare un evento.

```
aws lambda update-function-event-invoke-config \
  --function-name my-function \
  --destination-config '{"OnFailure":{"Destination": "arn:aws:sqs:us-
east-1:123456789012:destination"}}'
```

Best practice di sicurezza per destinazioni Amazon S3

L'eliminazione di un bucket S3 configurato come destinazione senza rimuovere la destinazione dalla configurazione della funzione può creare un rischio per la sicurezza. Se un altro utente conosce il nome del bucket di destinazione, può ricreare il bucket nel proprio Account AWS. I record delle invocazioni non riuscite verranno inviati al relativo bucket, esponendo potenzialmente i dati della tua funzione.

> **Warning**
>
> Per assicurarti che i record di invocazione della tua funzione non possano essere inviati a un bucket S3 in un altro Account AWS, aggiungi una condizione al ruolo di esecuzione della funzione che limiti le `s3:PutObject` autorizzazioni ai bucket del tuo account.

Di seguito viene illustrato un esempio di policy IAM che limita le autorizzazioni `s3:PutObject` della funzione ai bucket presenti nell'account. Questa policy fornisce inoltre a Lambda l'autorizzazione `s3:ListBucket` necessaria per utilizzare un bucket S3 come destinazione.

```
{
    "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"
                }
            }
        }
    ]
}
```

Per aggiungere una politica di autorizzazioni al ruolo di esecuzione della tua funzione utilizzando Console di gestione AWS o AWS CLI, consulta le istruzioni nelle seguenti procedure:

Console

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Seleziona la funzione Lambda di cui si desidera modificare il ruolo di esecuzione.
3. Nella scheda Configurazione scegli Autorizzazioni.
4. Nella scheda Ruolo di esecuzione, seleziona il nome del ruolo della funzione per aprire la pagina della console IAM del ruolo.
5. Aggiungi una policy di autorizzazioni di base al ruolo completando le seguenti operazioni:
   a. Nel riquadro Policy di autorizzazioni, scegli Aggiungi autorizzazioni, poi Crea policy in linea.
   b. Nell'editor delle policy, seleziona JSON.
   c. Incolla la policy che desideri aggiungere nell'editor (sostituendo il codice JSON esistente), quindi scegli Avanti.
   d. In Dettagli della policy, specifica un nome per la policy.
   e. Scegli Crea policy.

AWS CLI

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (CLI)

1. Crea un documento di policy JSON con le autorizzazioni richieste e salvalo in una directory locale.
2. Utilizza il comando della CLI `put-role-policy` di IAM per aggiungere le autorizzazioni per il ruolo di esecuzione di una funzione. Esegui il comando seguente dalla directory in cui hai

salvato il documento di policy JSON e sostituisci il nome del ruolo, il nome della policy e il documento di policy con i tuoi valori.

```
aws iam put-role-policy \
--role-name my_lambda_role \
--policy-name LambdaS3DestinationPolicy \
--policy-document file://my_policy.json
```

Record di invocazione di esempio

Quando un'invocazione corrisponde alla condizione, Lambda invia un documento JSON con i dettagli sull'invocazione alla destinazione. L'esempio seguente mostra un record di invocazione per un evento che non è stato possibile elaborare per tre volte a causa di un errore di funzione.

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"
    }
```

```
}
```

Il record di invocazione contiene dettagli sull'evento, la risposta e il motivo per cui il record è stato inviato.

Tracciamento delle richieste verso le destinazioni

È possibile utilizzare AWS X-Ray per visualizzare una vista connessa di ogni richiesta mentre viene messa in coda, elaborata da una funzione Lambda e inviata al servizio di destinazione. Quando si attiva il tracciamento X-Ray per una funzione o un servizio che richiama una funzione, Lambda aggiunge un'intestazione X-Ray alla richiesta e passa l'intestazione al servizio di destinazione. Le tracce dei servizi upstream vengono collegate automaticamente alle tracce delle funzioni Lambda a valle e dei servizi di destinazione, creando una end-to-end visualizzazione dell'intera applicazione. Per ulteriori informazioni sul tracciamento, consulta Visualizza le chiamate alla funzione Lambda utilizzando AWS X-Ray.

Aggiunta di una coda DLQ

In alternativa a una destinazione in caso di errore, è possibile configurare la funzione con una coda DLQ per salvare gli eventi eliminati per ulteriori elaborazioni. Una coda DLQ agisce allo stesso modo di una destinazione in caso di errore in quanto viene utilizzata quando un evento non riesce a tutti i tentativi di elaborazione o scade senza essere elaborato. Tuttavia, è possibile aggiungere o rimuovere una coda DLQ solo a livello di funzione. Le versioni della funzione utilizzano le stesse impostazioni della coda DLQ della versione non pubblicata ($LATEST). Le destinazioni in caso di errore supportano anche destinazioni aggiuntive e includono dettagli sulla risposta della funzione nel record di invocazione.

Per rielaborare gli eventi in una coda DLQ, è possibile impostarla come origine eventi per la funzione Lambda. In alternativa, è possibile recuperare gli eventi manualmente.

È possibile scegliere una coda standard di Amazon SQS o un argomento standard di Amazon SNS per la coda DLQ. Le code FIFO e gli argomenti FIFO di Amazon SNS non sono supportati.

- Coda Amazon SQS: una coda conserva gli eventi non riusciti finché non vengono richiamati. Scegli una coda standard Amazon SQS se prevedi che una singola entità, come una funzione Lambda o un CloudWatch allarme, elabori l'evento non riuscito. Per ulteriori informazioni, consulta Utilizzo di Lambda con Amazon SQS.
- Argomento Amazon SNS – Un argomento invia gli eventi non riusciti a una o più destinazioni. Scegli un argomento standard di Amazon SNS se ti aspetti che più entità agiscano su un evento

non riuscito. Ad esempio, puoi configurare un argomento per inviare eventi a un indirizzo e-mail, una funzione Lambda, and/or un endpoint HTTP. Per ulteriori informazioni, consulta Richiamo di funzioni Lambda mediante notifiche Amazon SNS.

Per inviare gli eventi a una coda o argomento, la funzione necessita di autorizzazioni aggiuntive. Aggiungi una policy con le autorizzazioni necessarie per il ruolo di esecuzione di una funzione. Se la coda o l'argomento di destinazione è crittografato con una AWS KMS chiave gestita dal cliente, assicurati che sia il ruolo di esecuzione della funzione che la politica basata sulle risorse della chiave contengano le autorizzazioni pertinenti.

Dopo aver creato il target e l'aggiornamento del ruolo di esecuzione della funzione, aggiungere la coda DLQ alla funzione. È possibile configurare più funzioni per l'invio di eventi allo stesso oggetto.

Console

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione), quindi scegli Asynchronous invocation (Chiamata asincrona).
4. In Asynchronous invocation (Chiamata asincrona), scegliere Edit (Modifica).
5. Imposta il servizio coda DLQ su Amazon SQS o Amazon SNS.
6. Scegliere l'argomento o la coda target.
7. Scegli Save (Salva).

AWS CLI

Per configurare una coda di lettere morte con, usa il comando. AWS CLIupdate-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 invia l'evento alla coda DLQ così com'è, con ulteriori informazioni negli attributi. Queste informazioni possono essere utilizzate per identificare l'errore restituito dalla funzione o correlare l'evento ai log o a una traccia AWS X-Ray .

Attributi dei messaggi della coda DLQ

- RequestID (String) – L'ID della richiesta di invocazione. La richiesta IDs viene visualizzata nei registri delle funzioni. È inoltre possibile utilizzare l'SDK X-Ray per registrare l'ID di richiesta su un attributo nella traccia. Si possono quindi cercare le tracce in base all'ID richiesta nella console X-Ray.
- ErrorCode(Numero): il codice di stato HTTP.
- ErrorMessage(String) — Il primo 1 KB del messaggio di errore.

Se Lambda non riesce a inviare un messaggio alla coda delle lettere non scritte, elimina l'evento ed emette la metrica. DeadLetterErrors Questo può accadere a causa di mancanza di autorizzazioni oppure se le dimensioni totali del messaggio superano il limite per la coda o l'argomento target. Ad esempio, supponiamo che una notifica Amazon SNS con un corpo di dimensioni prossime a 1 MB attivi una funzione che genera un errore. In tal caso, i dati relativi a eventi aggiunti da Amazon SNS, combinati con gli attributi aggiunti da Lambda, possono far sì che il messaggio superi le dimensioni massime consentite nella coda DLQ.

Se si utilizza Amazon SQS come origine eventi, configurare una coda DLQ sulla coda Amazon SQS stessa e non sulla funzione Lambda. Per ulteriori informazioni, consulta Utilizzo di Lambda con Amazon SQS.

Richiamo di funzioni Lambda durevoli

Le funzioni Lambda durevoli possono essere richiamate utilizzando gli stessi metodi delle funzioni Lambda predefinite, ma con importanti considerazioni per le esecuzioni di lunga durata. Questa sezione descrive i modelli di invocazione, la gestione dell'esecuzione e le migliori pratiche per funzioni durevoli.

Limiti di invocazione sincrona

Le chiamate sincrone di funzioni Lambda durevoli sono limitate a 15 minuti, come le funzioni Lambda predefinite. Se la funzione duratura deve essere eseguita per più di 15 minuti, deve essere richiamata in modo asincrono.

Quando utilizzare la chiamata sincrona: utilizzala per funzioni durevoli che vengono completate entro 15 minuti e quando sono necessari risultati immediati, come flussi di lavoro di approvazione rapidi o brevi attività di elaborazione dei dati.

Richiamata asincrona per flussi di lavoro di lunga durata

Per funzioni durevoli che possono durare più di 15 minuti, utilizzate la chiamata asincrona. Ciò consente alla funzione di continuare a funzionare mentre il client riceve un riconoscimento immediato.

TypeScript

```
import { LambdaClient, InvokeCommand } from "@aws-sdk/client-lambda";

const client = new LambdaClient({});

// Asynchronous invocation
const command = new InvokeCommand({
  FunctionName: "my-durable-function",
  InvocationType: "Event", // Asynchronous
  Payload: JSON.stringify({ orderId: "12345" })
});

await client.send(command);
```

Python

```
import boto3
import json

client = boto3.client('lambda')

# Asynchronous invocation
response = client.invoke(
    FunctionName='my-durable-function',
    InvocationType='Event',  # Asynchronous
    Payload=json.dumps({'order_id': '12345'})
)
```

Gestione dell'esecuzione APIs

Lambda consente APIs di gestire e monitorare le esecuzioni di funzioni durevoli, tra cui elencare le esecuzioni, ottenere lo stato dell'esecuzione e interrompere le esecuzioni in esecuzione.

TypeScript

```
// Get execution status
const statusCommand = new InvokeCommand({
  FunctionName: "my-durable-function",
  InvocationType: "RequestResponse",
  Payload: JSON.stringify({
    action: "getStatus",
    executionId: "exec-123"
  })
});

const result = await client.send(statusCommand);
```

Python

```
# Get execution status
```

```
response = client.invoke(
    FunctionName='my-durable-function',
    InvocationType='RequestResponse',
    Payload=json.dumps({
        'action': 'get_status',
        'execution_id': 'exec-123'
    })
)
```

In che modo Lambda elabora i record provenienti da origini eventi basate su flussi e code

Uno strumento di mappatura dell'origine degli eventi è una risorsa Lambda che legge gli elementi da servizi basati su flussi o code e richiama una funzione con batch di record. All'interno di uno strumento di mappatura dell'origine degli eventi, risorse chiamate poller di eventi cercano attivamente nuovi messaggi e richiamano funzioni. Per impostazione predefinita, Lambda scala automaticamente i poller di eventi, ma per determinati tipi di origini eventi, puoi utilizzare la modalità provisioning per controllare il numero minimo e massimo di poller di eventi dedicati allo strumento di mappatura dell'origine degli eventi.

I seguenti servizi utilizzano gli strumenti di mappatura dell'origine degli eventi per richiamare le funzioni Lambda:

- Amazon DocumentDB (compatibile con MongoDB) (Amazon DocumentDB)
- Amazon DynamoDB
- Amazon Kinesis
- Amazon MQ
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)
- Apache Kafka gestito dal cliente
- Amazon Simple Queue Service (Amazon SQS)

> **Warning**
>
> Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center. AWS

In che modo gli strumenti di mappatura dell'origine degli eventi differiscono dai trigger diretti

Alcuni Servizi AWS possono richiamare direttamente le funzioni Lambda utilizzando i trigger. Questi servizi inviano eventi a Lambda e la funzione viene richiamata immediatamente quando si verifica l'evento specificato. I trigger sono adatti per eventi discreti ed elaborazione in tempo reale. Quando crei un trigger utilizzando la console Lambda, la console interagisce con il AWS servizio corrispondente per configurare la notifica degli eventi su quel servizio. Il trigger viene effettivamente archiviato e gestito dal servizio che genera gli eventi, non da Lambda. Ecco alcuni esempi di servizi che utilizzano i trigger per richiamare le funzioni Lambda:

- Amazon Simple Storage Service (Amazon S3): richiama una funzione quando un oggetto viene creato, eliminato o modificato in un bucket. Per ulteriori informazioni, consulta Tutorial: uso di un trigger Amazon S3 per richiamare una funzione Lambda.
- Amazon Simple Notification Service (Amazon SNS): richiama una funzione quando un messaggio è pubblicato in un argomento SNS. Per ulteriori informazioni, consulta Tutorial: Utilizzo AWS Lambda con Amazon Simple Notification Service.
- Gateway Amazon API: richiama una funzione quando viene effettuata una richiesta API a un endpoint specifico. Per ulteriori informazioni, consulta Richiamo di funzione Lambda utilizzando un endpoint Gateway Amazon API.

Gli strumenti di mappatura dell'origine degli eventi sono risorse Lambda create e gestite all'interno del servizio Lambda. Gli strumenti di mappatura dell'origine degli eventi sono progettate per l'elaborazione di dati o messaggi in streaming ad alto volume dalle code. L'elaborazione dei record da un flusso o da una coda in batch è più efficiente rispetto all'elaborazione dei record singolarmente.

Comportamento di batching

Per impostazione predefinita, una mappatura delle origini eventi raggruppa i registri in un unico payload che Lambda invia alla funzione. Per ottimizzare il comportamento del batch, è possibile configurare una finestra di batch (MaximumBatchingWindowInSeconds) e una dimensione del batch (BatchSize). Una finestra di batch è il tempo massimo per la raccolta dei registri in un singolo payload. La dimensione del batch è il numero massimo di registri in un singolo batch. Lambda richiama la funzione in presenza dei tre criteri seguenti:

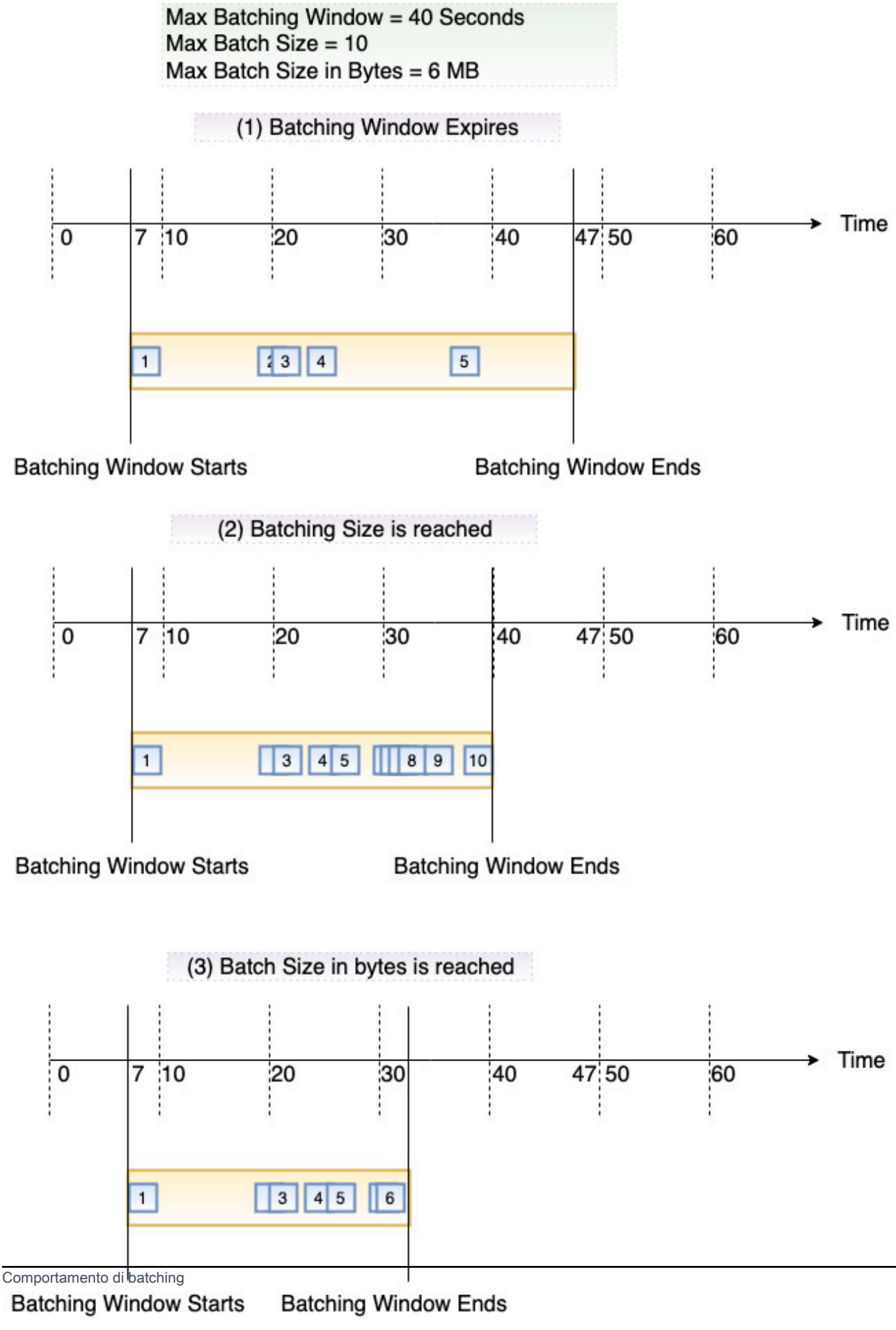
- La finestra di dosaggio raggiunge il valore massimo. Il comportamento predefinito della finestra di batch varia in base alla specifica origine eventi.

- Per le origini eventi Kinesis, DynamoDB e Amazon SQS: la finestra di batch di default è 0 secondi. Ciò significa che Lambda richiama la funzione non appena i record sono disponibili. Per impostare una finestra di batch, configura `MaximumBatchingWindowInSeconds`. È possibile impostare questo parametro su qualsiasi valore da compreso tra 0 e 300 secondi con incrementi di 1 secondo. Se si configura una la finestra di batch, la finestra successiva inizia non appena viene completato il precedente richiamo della funzione.
- Per le origini degli eventi di Amazon MSK, Apache Kafka autogestito, Amazon MQ e Amazon DocumentDB la finestra di batch predefinita è 500 ms. È possibile configurare `MaximumBatchingWindowInSeconds` su qualsiasi valore da 0 secondi a 300 secondi con incrementi di secondi. Nella modalità preimpostata per le mappature delle sorgenti degli eventi Kafka, quando si configura una finestra di batch, la finestra successiva inizia non appena viene completato il batch precedente. Nelle mappature delle sorgenti di eventi Kafka senza provisioning, quando si configura una finestra di batch, la finestra successiva inizia non appena viene completata la precedente invocazione della funzione. Per ridurre al minimo la latenza quando si utilizzano le mappature delle sorgenti degli eventi Kafka in modalità provisioning, impostate su 0. `MaximumBatchingWindowInSeconds` Questa impostazione assicura che Lambda inizi l'elaborazione del batch successivo immediatamente dopo aver completato la chiamata della funzione corrente. Per ulteriori informazioni sull'elaborazione a bassa latenza, consulta. Apache Kafka a bassa latenza
- Per le sorgenti di eventi Amazon MQ e Amazon DocumentDB: la finestra di batch predefinita è di 500 ms. È possibile configurare `MaximumBatchingWindowInSeconds` su qualsiasi valore da 0 secondi a 300 secondi con incrementi di secondi. Una finestra di batch inizia non appena arriva il primo registro.

  > **Note**
  >
  > Perché è possibile modificare `MaximumBatchingWindowInSeconds` solo in incrementi di secondi, non puoi tornare alla finestra di batch predefinita di 500 ms dopo averla modificata. Per ripristinare la finestra di batch predefinita, è necessario creare una nuova mappatura dell'origine eventi.

- Le dimensioni del batch sono soddisfatte. La dimensione minima del batch è 1. La dimensione predefinita e massima del batch dipendono dall'origine eventi. Per ulteriori informazioni su questi valori, consulta le specifiche di BatchSize per l'operazione API `CreateEventSourceMapping`.
- La dimensione del payload raggiunge 6 MB. Tale limite non è modificabile.

Il diagramma seguente illustra queste tre configurazioni. Supponiamo che una finestra di batch inizi a `t = 7` secondi. Nel primo scenario, la finestra di batch raggiunge il suo massimo di 40 secondi a `t = 47` secondi dopo aver accumulato 5 registri. Nel secondo scenario, la dimensione del batch raggiunge 10 prima della scadenza della finestra di batch, quindi la finestra di batch termina in anticipo. Nel secondo scenario, il valore massimo del payload viene raggiunta prima della scadenza della finestra di batch, quindi la finestra di batch termina in anticipo.


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

Consigliamo di eseguire il test con dimensioni di batch e record diverse in modo che la frequenza di polling di ogni origine eventi sia regolata in base alla velocità con cui la funzione è in grado di completare l'attività. Il CreateEventSourceMapping`BatchSize`parametro controlla il numero massimo di record che possono essere inviati alla funzione con ogni chiamata. Una dimensione di batch maggiore può spesso assorbire in modo più efficiente il costo associato all'invocazione in un set di record più grande, aumentando in tal modo il throughput.

Lambda non attende il completamento di alcuna estensione prima di inviare il batch successivo per l'elaborazione. In altre parole, le estensioni possono continuare a funzionare mentre Lambda elabora il successivo batch di record. Ciò può causare problemi di limitazione in caso di violazione delle impostazioni o dei limiti di simultaneità. Per rilevare se si tratta di un potenziale problema, monitora le tue funzioni e verifica se i parametri di simultaneità per lo strumento di mappatura dell'origine degli eventi sono superiori al previsto. A causa degli intervalli ridotti tra le invocazioni, Lambda potrebbe segnalare brevemente un utilizzo della simultaneità maggiore rispetto al numero di partizioni. Ciò può essere vero anche per le funzioni Lambda senza estensioni.

Per impostazione predefinita, se la funzione restituisce un errore, l'intera mappatura del batch viene rielaborato fino a quando la funzione non restituisce esito positivo o gli elementi nel batch scadono. Per garantire l'ordine di elaborazione, l'elaborazione della mappatura delle fonti eventi per gli shard interessati viene messa in pausa finché l'errore non viene risolto. Per le origini di flusso (DynamoDB e Kinesis), è possibile configurare il numero massimo di tentativi che Lambda effettua quando la funzione restituisce un errore. Gli errori di servizio o le limitazioni per cui il batch non raggiunge la funzione non vengono conteggiati ai fini dei tentativi ripetuti. È inoltre possibile configurare lo strumento di mappatura dell'origine degli eventi per inviare un record di invocazione a una destinazione quando elimina un batch di eventi.

Modalità provisioning

Gli strumenti di mappatura dell'origine degli eventi Lambda utilizzano i poller di eventi per interrogare l'origine eventi alla ricerca di nuovi messaggi. Per impostazione predefinita, Lambda gestisce la scalabilità automatica di questi poller in base al volume dei messaggi. Quando il traffico di messaggi aumenta, Lambda aumenta automaticamente il numero di poller di eventi per gestire il carico e li riduce quando il traffico diminuisce.

In modalità provisioned, puoi ottimizzare la velocità effettiva della mappatura delle sorgenti degli eventi definendo limiti minimi e massimi per le risorse di polling dedicate che rimangono pronte a gestire i modelli di traffico previsti. Queste risorse si scalano automaticamente 3 volte più velocemente per gestire picchi improvvisi nel traffico degli eventi e forniscono una capacità 16 volte

superiore per elaborare milioni di eventi. Questo ti aiuta a creare carichi di lavoro basati sugli eventi altamente reattivi con requisiti prestazionali rigorosi.

In Lambda, un event poller è un'unità di calcolo con funzionalità di throughput che variano in base al tipo di origine dell'evento. Per Amazon MSK e Apache Kafka autogestito, ogni event poller può gestire fino al 5% MB/sec del throughput o fino a 5 chiamate simultanee. Ad esempio, se la fonte dell'evento produce un payload medio di 1 MB e la durata media della funzione è di 1 secondo, un singolo event poller di eventi Kafka può supportare 5 MB/sec velocità effettiva e 5 chiamate Lambda simultanee (presupponendo che non vi sia alcuna trasformazione del payload). Per Amazon SQS, ogni event poller può gestire fino all'1% MB/sec del throughput o fino a 10 chiamate simultanee. L'utilizzo della modalità provisioned comporta costi aggiuntivi in base all'utilizzo dell'event poller. Per i dettagli sui prezzi, vedere Prezzi di AWS Lambda.

La modalità provisioned è disponibile per Amazon MSK, Apache Kafka autogestite e sorgenti di eventi Amazon SQS. Mentre le impostazioni di simultaneità consentono di controllare la scalabilità della funzione, la modalità provisioning consente di controllare il throughput dello strumento di mappatura dell'origine degli eventi. Per garantire le massime prestazioni, potrebbe essere necessario regolare entrambe le impostazioni in modo indipendente.

La modalità Provisioned è ideale per applicazioni in tempo reale che richiedono una latenza di elaborazione degli eventi costante, come le società di servizi finanziari che elaborano i feed di dati di mercato, le piattaforme di e-commerce che forniscono consigli personalizzati in tempo reale e le società di gioco che gestiscono le interazioni tra giocatori dal vivo.

Ogni event poller supporta una capacità di throughput diversa:

- Per Amazon MSK e Apache Kafka autogestito: fino al 5% MB/sec della velocità effettiva o fino a 5 richiami simultanei
- Per Amazon SQS: fino all'1% MB/sec del throughput o fino a 10 richiami simultanei o fino a 10 chiamate API di polling SQS al secondo.

Per le mappature delle sorgenti di eventi Amazon SQS, puoi impostare il numero minimo di poller tra 2 e 200 con un valore predefinito di 2 e il numero massimo tra 2 e 2.000 con un valore predefinito di 200. Lambda ridimensiona il numero di sondaggi di eventi tra il minimo e il massimo configurati, aggiungendo rapidamente fino a 1.000 simultanee al minuto per fornire un'elaborazione coerente e a bassa latenza degli eventi.

Per le mappature delle sorgenti degli eventi Kafka, puoi impostare il numero minimo di poller tra 1 e 200 con un valore predefinito di 1 e il numero massimo tra 1 e 2.000 con un valore predefinito di 200.

Lambda ridimensiona il numero di sondaggi degli eventi tra il minimo e il massimo configurati in base al backlog degli eventi nell'argomento per fornire un'elaborazione a bassa latenza degli eventi.

Tieni presente che per le sorgenti di eventi Amazon SQS, l'impostazione di concorrenza massima non può essere utilizzata con la modalità provisioned. Quando si utilizza la modalità provisioned, è possibile controllare la concorrenza tramite l'impostazione del numero massimo di poller di eventi.

Per informazioni dettagliate sulla configurazione della modalità provisioning, consulta le sezioni seguenti:

- Configurazione della modalità provisioning per gli strumenti di mappatura dell'origine degli eventi Amazon MSK
- Configurazione della modalità provisioning per lo strumento di mappatura dell'origine degli eventi di Apache Kafka autogestito
- Utilizzo della modalità provisioning con le mappature delle sorgenti degli eventi di Amazon SQS

Per ridurre al minimo la latenza in modalità provisioned, impostate su 0. `MaximumBatchingWindowInSeconds` Questa impostazione assicura che Lambda inizi l'elaborazione del batch successivo immediatamente dopo aver completato la chiamata della funzione corrente. Per ulteriori informazioni sull'elaborazione a bassa latenza, consulta. Apache Kafka a bassa latenza

Dopo aver configurato la modalità provisioning, puoi osservare l'utilizzo dei poller di eventi per il tuo carico di lavoro monitorando il parametro `ProvisionedPollers`. Per ulteriori informazioni, consulta the section called “Parametri dello strumento di mappatura dell'origine degli eventi”.

API della mappatura dell'origine eventi

Per gestire un'origine eventi con la AWS Command Line Interface (AWS CLI) o un SDK AWS, è possibile utilizzare le seguenti operazioni API:

- CreateEventSourceMapping
- ListEventSourceMappings
- GetEventSourceMapping
- UpdateEventSourceMapping
- DeleteEventSourceMapping

Utilizzo di tag negli strumenti di mappatura dell'origine degli eventi

Puoi taggare gli strumenti di mappatura dell'origine degli eventi per organizzare e gestire le risorse. I tag sono coppie chiave-valore a forma libera associate alle risorse supportate su Servizi AWS. Per ulteriori informazioni sui casi d'uso dei tag, consulta Strategie di tagging comuni nella Guida AWS alle risorse di etichettatura e all'editor di tag.

Gli strumenti di mappatura dell'origine degli eventi sono associati a funzioni che possono avere i propri tag. Gli strumenti di mappatura dell'origine degli eventi non ereditano automaticamente i tag dalle funzioni. Puoi utilizzare l' AWS Lambda API per visualizzare e aggiornare i tag. Puoi anche visualizzare e aggiornare i tag mentre gestisci una mappatura specifica dell'origine degli eventi nella console Lambda, compresi quelli che utilizzano la modalità Provisioned per Amazon SQS.

Autorizzazioni necessarie per lavorare con i tag

Per consentire a un'identità AWS Identity and Access Management (IAM) (utente, gruppo o ruolo) di leggere o impostare tag su una risorsa, concedile le autorizzazioni corrispondenti:

- lambda: ListTags —Quando una risorsa contiene dei tag, concedi questa autorizzazione a chiunque debba richiamarla. `ListTags` Per le funzioni con tag, questa autorizzazione è necessaria anche per `GetFunction`.
- lambda: TagResource —Concedi questa autorizzazione a chiunque abbia bisogno di chiamare `TagResource` o eseguire un tag durante la creazione.

Facoltativamente, valuta la possibilità di concedere anche l'UntagResourceautorizzazione lambda: per consentire `UntagResource` le chiamate alla risorsa.

Per ulteriori informazioni, consulta Policy IAM basate sull'identità per Lambda.

Utilizzo di tag con la console Lambda

Puoi utilizzare la console Lambda per creare mappature delle sorgenti di eventi con tag, aggiungere tag alle mappature delle sorgenti di eventi esistenti e filtrare le mappature delle sorgenti degli eventi per tag, comprese quelle configurate in modalità Provisioned per Amazon SQS.

Quando aggiungi un trigger per i servizi basati su flussi e code supportati tramite la console Lambda, Lambda crea automaticamente uno strumento di mappatura dell'origine degli eventi. Per ulteriori informazioni su queste origini eventi, consulta the section called “Strumento di mappatura dell'origine

degli eventi". Per creare uno strumento di mappatura dell'origine degli eventi nella console, sono necessari i prerequisiti seguenti:

- Una funzione .
- Un'origine eventi proveniente da un servizio interessato.

È possibile aggiungere i tag come parte della stessa interfaccia utente utilizzata per creare o aggiornare i trigger.

Per aggiungere un tag durante la creazione di uno strumento di mappatura dell'origine degli eventi

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione .
3. In Panoramica delle funzioni, scegliere Aggiungi trigger.
4. In Configurazione trigger, nell'elenco a discesa, scegli il nome del servizio da cui proviene l'origine eventi.
5. Fornisci la configurazione di base per la tua origine eventi. Per ulteriori informazioni sulla configurazione dell'origine eventi, consulta la sezione per il servizio correlato in Integrazione con altri servizi.
6. In Configurazione dello strumento di mappatura dell'origine degli eventi, scegli Impostazioni aggiuntive.
7. In Tag, seleziona Aggiungi nuovo tag.
8. Nel campo Chiave, inserisci la chiave del tag. Per informazioni sulle restrizioni relative ai tag, consulta Limiti e requisiti di denominazione dei tag nella Guida alle risorse di etichettatura e all'editor di tag. AWS
9. Scegliere Aggiungi.

Per aggiungere tag a uno strumento di mappatura dell'origine degli eventi esistente

1. Apri Strumenti di mappatura dell'origine degli eventi nella console Lambda.
2. Dall'elenco delle risorse, scegli l'UUID per lo strumento di mappatura dell'origine degli eventi corrispondente alla funzione e all'ARN dell'origine eventi.
3. Dall'elenco delle schede sotto il riquadro di configurazione generale, scegli Tag.
4. Scegliere Gestisci tag.

5. Scegliere Aggiungi nuovo tag.
6. Nel campo Chiave, inserisci la chiave del tag. Per informazioni sulle restrizioni relative ai tag, consulta Limiti e requisiti di denominazione dei tag nella Guida alle risorse di etichettatura e all'editor di tag AWS .
7. Scegli Save (Salva).

Per filtrare gli strumenti di mappatura dell'origine degli eventi per tag

1. Apri Strumenti di mappatura dell'origine degli eventi nella console Lambda.
2. Scegli la barra di ricerca.
3. Dall'elenco a discesa, seleziona la chiave di tag sotto Tag.
4. Seleziona Usa: "nome-tag" per vedere tutti gli strumenti di mappatura dell'origine degli eventi etichettati con questa chiave, oppure scegli un operatore per filtrare ulteriormente in base al valore.
5. Seleziona il valore del tag da filtrare in base a una combinazione di chiave e valore del tag.

La barra di ricerca supporta anche la ricerca di chiavi di tag. Immetti il nome di una chiave per trovarla nell'elenco.

Usare i tag con il AWS CLI

Puoi aggiungere e rimuovere tag sulle risorse Lambda esistenti, inclusi gli strumenti di mappatura dell'origine degli eventi, con l'API Lambda. Puoi aggiungere i tag anche quando crei uno strumento di mappatura dell'origine degli eventi, che ti consente di mantenere etichettata una risorsa per tutto il suo ciclo di vita.

Aggiornamento dei tag con il tag Lambda APIs

Puoi aggiungere e rimuovere tag per le risorse Lambda supportate tramite le operazioni TagResourcee UntagResourceAPI.

Puoi chiamare queste operazioni tramite la AWS CLI. Per aggiungere i tag a una risorsa esistente, utilizza il comando `tag-resource`. Questo esempio aggiunge due tag, uno con la chiave *`Department`* e uno con la chiave*`CostCenter`*.

```
aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:resource-type:my-resource \
```

```
--tags Department=Marketing,CostCenter=1234ABCD
```

Pr rimuovere i tag, utilizza il comando `untag-resource`. Questo esempio rimuove il tag con la chiave*`Department`*.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:resource-type:resource-identifier \
--tag-keys Department
```

Aggiunta di tag durante la creazione di uno strumento di mappatura dell'origine degli eventi

Per creare una nuova mappatura delle sorgenti di eventi Lambda con tag, utilizza l'CreateEventSourceMappingoperazione API. Specifica il parametro `Tags`. È possibile richiamare questa operazione con il `create-event-source-mapping` AWS CLI comando e l'`--tags`opzione. Per ulteriori informazioni sul comando CLI, vedere create-event-source-mappingnel Command Reference AWS CLI .

Prima di utilizzare il parametro `Tags` con `CreateEventSourceMapping`, assicurati che il tuo ruolo disponga dell'autorizzazione per etichettare le risorse oltre alle normali autorizzazioni necessarie per questa operazione. Per ulteriori informazioni sulle autorizzazioni per il tagging, consulta the section called “Autorizzazioni necessarie per lavorare con i tag”.

Visualizzazione dei tag con il tag Lambda APIs

Per visualizzare i tag applicati a una risorsa Lambda specifica, utilizza l'operazione API `ListTags`. Per ulteriori informazioni, consulta ListTags.

Puoi richiamare questa operazione con il `list-tags` AWS CLI comando fornendo un ARN (Amazon Resource Name).

```
aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:resource-type:resource-identifier
```

Filtro delle risorse per tag

Puoi utilizzare l'operazione AWS Resource Groups Tagging API GetResourcesAPI per filtrare le tue risorse in base ai tag. L'operazione `GetResources` riceve fino a 10 filtri, ognuno dei quali contenente una chiave di tag e un massimo di 10 valori di tag. Fornisci `GetResources` con un `ResourceType` per filtrare in base a tipi di risorse specifiche.

È possibile richiamare questa operazione utilizzando il `get-resources` AWS CLI comando. Per esempi di utilizzo di `get-resources`, consulta get-resources nella Riferimento ai comandi CLI di AWS .

Controllare gli eventi che Lambda invia alla funzione

Puoi utilizzare il filtraggio degli eventi per controllare quali record di un flusso o di una coda Lambda invia alla funzione. Ad esempio, puoi aggiungere un filtro in modo che la tua funzione elabori solo i messaggi Amazon SQS contenenti determinati parametri di dati. Il filtraggio degli eventi funziona solo con determinati strumenti di mappatura dell'origine degli eventi. È possibile aggiungere filtri alle mappature delle sorgenti degli eventi per quanto segue: Servizi AWS

- Amazon DynamoDB
- Flusso di dati Amazon Kinesis
- Amazon MQ
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)
- Apache Kafka gestito dal cliente
- Amazon Simple Queue Service (Amazon SQS)

Per informazioni specifiche sul filtraggio con origini eventi specifiche, consulta the section called “Utilizzo di filtri con diversi Servizi AWS”. Lambda non supporta il filtraggio degli eventi per Amazon DocumentDB.

Per impostazione predefinita, è possibile definire fino a cinque filtri diversi per una singola mappatura dell'origine degli eventi. I tuoi filtri sono collegati ORed logicamente. Se un record proveniente dall'origine dell'evento soddisfa uno o più filtri, Lambda include il record nell'evento successivo che invia alla funzione. Se nessuno dei filtri è soddisfatto, Lambda scarta il record.

> **Note**
>
> Se devi definire più di cinque filtri per un'origine eventi, puoi richiedere un aumento della quota fino a 10 filtri per ciascuna origine eventi. Se provi ad aggiungere più filtri di quelli consentiti dalla quota corrente, Lambda restituisce un errore quando provi a creare l'origine degli eventi.

Argomenti

- Nozioni di base sul filtraggio di eventi
- Gestione dei record che non soddisfano i criteri di filtraggio

- Sintassi delle regole di filtro
- Collegamento dei criteri di filtro a una mappatura dell'origine evento (console)
- Collegamento dei criteri di filtro a una mappatura dell'origine evento (AWS CLI)
- Collegamento dei criteri di filtro a una mappatura dell'origine evento (AWS SAM)
- Crittografia dei criteri di filtro
- Utilizzo di filtri con diversi Servizi AWS

Nozioni di base sul filtraggio di eventi

Un oggetto criterio di filtro (`FilterCriteria`) è una struttura costituita da un elenco di filtri (`Filters`). Ogni filtro è una struttura che definisce un modello di filtraggio degli eventi (`Pattern`). Un modello è una rappresentazione di stringa di una regola di filtraggio JSON. La struttura di un oggetto `FilterCriteria` è come segue.

```
{
   "Filters": [
        {
            "Pattern": "{ \"Metadata1\": [ rule1 ], \"data\": { \"Data1\":
 [ rule2 ] }}"
        }
    ]
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
    "Metadata1": [ rule1 ],
    "data": {
        "Data1": [ rule2 ]
    }
}
```

Il modello di filtraggio può includere proprietà dei metadati, proprietà dei dati o entrambe. I parametri dei metadati disponibili e il formato dei parametri dei dati variano a seconda del Servizio AWS che funge da origine dell'evento. Ad esempio, supponiamo che la mappatura dell'origine degli eventi riceva il seguente record da una coda 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"
}
```

- Le Proprietà dei metadati sono i campi contenenti informazioni sull'evento che ha creato il record. Nel record Amazon SQS di esempio, le proprietà dei metadati includono campi come `messageIDeventSourceArn` e `awsRegion`.
- Le Proprietà dei dati sono i campi del record contenenti i dati del flusso o della coda. Nell'esempio di evento Amazon SQS, la chiave per il campo dati è `body` e le proprietà dei dati sono i campi `City`, `State` e `Temperature`.

Diversi tipi di origine degli eventi utilizzano valori di chiave differenti per i rispettivi campi di dati. Per filtrare le proprietà dei dati, assicurati di utilizzare la chiave corretta nel modello del filtraggio. Per un elenco delle chiavi di filtraggio dei dati e per vedere esempi di modelli di filtro per ciascuna delle chiavi supportate Servizio AWS, consulta. Utilizzo di filtri con diversi Servizi AWS

Il filtraggio degli eventi è in grado di gestire il filtraggio JSON multi-livello. Considera, ad esempio, il seguente frammento di un record da un flusso DynamoDB:

```
"dynamodb": {
    "Keys": {
        "ID": {
            "S": "ABCD"
        }
        "Number": {
            "N": "1234"
    },
    ...
```

```
}
```

Supponi di voler elaborare solo i record in cui il valore della chiave di ordinamento `Number` è 4567. In questo caso, l'oggetto `FilterCriteria` appare così:

```
{
    "Filters": [
        {
            "Pattern": "{ \"dynamodb\": { \"Keys\": { \"Number\": { \"N\":
 [ "4567" ] } } } }"
        }
    ]
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
    "dynamodb": {
        "Keys": {
            "Number": {
                "N": [ "4567" ]
                }
            }
        }
}
```

Gestione dei record che non soddisfano i criteri di filtraggio

Il modo in cui Lambda gestisce i record che non soddisfano i criteri di filtro dipende dall'origine eventi.

- Per Amazon SQS, se un messaggio non soddisfa i criteri di filtraggio, Lambda rimuove automaticamente il messaggio dalla coda. Non è necessario eliminare manualmente questi messaggi in Amazon SQS.
- Per Kinesis e DynamoDB, una volta che i criteri di filtraggio valutano un record, l'iteratore di flussi passa oltre questo record. Se il registro non soddisfa i criteri di filtro, non è necessario eliminare manualmente il registro dall'origine evento. Dopo il periodo di conservazione, Kinesis e DynamoDB eliminano automaticamente questi vecchi record. Se vuoi che i record vengano eliminati prima, consulta Modifica del periodo di conservazione dei dati.
- Per i messaggi Amazon MSK, Apache Kafka autogestiti e Amazon MQ, Lambda elimina i messaggi che non corrispondono a tutti i campi inclusi nel filtro. Per Amazon MSK e Apache

Kafka autogestito, Lambda esegue il commit degli offset per i messaggi corrispondenti e non corrispondenti dopo aver richiamato correttamente la funzione. Per Amazon MQ, Lambda riconosce i messaggi corrispondenti dopo aver richiamato con successo la funzione e riconosce i messaggi non corrispondenti quando li filtra.

Sintassi delle regole di filtro

Per le regole di filtro, Lambda supporta EventBridge le regole di Amazon e utilizza la stessa sintassi di. EventBridge Per ulteriori informazioni, consulta i modelli di EventBridge eventi di Amazon nella Amazon EventBridge User Guide.

Di seguito è riportato un riepilogo di tutti gli operatori di confronto disponibili per il filtro eventi Lambda.

Operatore di confronto	Esempio	Sintassi delle regole
Null	UserID è nullo	"UserID": [nullo]
Empty	LastName è vuoto	"LastName": [""]
Equals	Il nome è "Alice"	"Nome": ["Alice"]
Uguale a (ignora maiuscole e minuscole)	Il nome è "Alice"	«Nome»: [{"equals-ignore-case«: «alice»}]
And	La posizione è "New York" e il giorno è "lunedì"	"Luogo": ["New York"], "Giorno": ["lunedì"]
Or	PaymentType è «Credito» o «Debito»	"PaymentType«: [«Credito», «Debito"]
Or (campi multipli)	La posizione è "New York" o il giorno è "lunedì".	"$or": [{ "Location": ["New York"] }, { "Day": ["Monday"] }]
Not	Il tempo è qualsiasi tranne "piovoso"	"Meteo": [{ "anything-but": ["Piove"] }]
Numerico (uguale)	Il prezzo è 100	"Prezzo": [{ "numerico": ["=", 100] }]

Operatore di confronto	Esempio	Sintassi delle regole
Numerico (intervallo)	Il prezzo è superiore a 10 e inferiore o uguale a 20	"Prezzo": [{ "numerico": [">", 10, "<=", 20] }]
Exists	ProductName esiste	"ProductName«: [{«exists»: vero}]
Does not exist	ProductName non esiste	"ProductName«: [{«exists»: false}]
Begins with	La Regione è negli Stati Uniti	"Regione": [{"prefisso": "us-" }]
Ends with	FileName termina con un'estensione.png.	"FileName«: [{«suffix»: «.png»}]

> **Note**
>
> Ad esempio EventBridge, per le stringhe, Lambda utilizza la corrispondenza character-by-character esatta senza ripiegamento tra maiuscole e minuscole o qualsiasi altra normalizzazione delle stringhe. Per i numeri Lambda utilizza anche la rappresentazione di stringhe. Ad esempio, 300, 300.0 e 3.0e2 non sono considerati uguali.

Tieni presente che l'operatore Exists funziona solo sui nodi foglia nell'origine eventi JSON. Non corrisponde ai nodi intermedi. Ad esempio, con il seguente codice JSON, lo schema di filtro `{ "person": { "address": [ { "exists": true } ] } }` non troverebbe una corrispondenza perché `"address"` è un nodo intermedio.

```
{
  "person": {
    "name": "John Doe",
    "age": 30,
    "address": {
      "street": "123 Main St",
      "city": "Anytown",
      "country": "USA"
    }
```

```
  }
}
```

Collegamento dei criteri di filtro a una mappatura dell'origine evento (console)

Seguire questi passaggi per creare una nuova mappatura delle origini eventi con criteri di filtro utilizzando la console Lambda.

Per creare una nuova mappatura dell'origine evento con criteri di filtro (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere il nome di una funzione per la creazione di una mappatura dell'origine evento.
3. In Panoramica delle funzioni, scegliere Aggiungi trigger.
4. Per Trigger configuration (Configurazione trigger), scegliere un tipo di attivazione che supporta il filtraggio degli eventi. Per un elenco dei servizi supportati, consulta l'elenco all'inizio di questa pagina.
5. Espandere Additional settings (Impostazioni aggiuntive).
6. Alla voce Filter criteria (Criteri di filtro), seleziona Add (Aggiungi), quindi definisci e immetti i filtri. Ad esempio, è possibile inserire i seguenti valori.

   ```
   { "Metadata" : [ 1, 2 ] }
   ```

   Ciò indica a Lambda di elaborare solo i registri in cui il campo `Metadata` è uguale a 1 o 2. Puoi continuare a selezionare Aggiungi per aggiungere altri filtri fino al numero massimo consentito.
7. Una volta completata l'aggiunta di filtri, scegli Salva.

Quando si inseriscono i criteri di filtraggio utilizzando la console, si inserisce solo il modello di filtraggio e non è necessario fornire la chiave `Pattern` o le virgolette di escape. Nel passaggio 6 delle istruzioni precedenti, `{ "Metadata" : [ 1, 2 ] }` corrisponde ai `FilterCriteria` seguenti.

```
{
   "Filters": [
      {
          "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
```

```
      }
   ]
}
```

Dopo aver creato la mappatura dell'origine eventi nella console, è possibile visualizzare il `FilterCriteria` formattato nei dettagli del trigger. Per altri esempi di creazione di filtri per gli eventi utilizzando la console, consulta la pagina Utilizzo di filtri con diversi Servizi AWS.

Collegamento dei criteri di filtro a una mappatura dell'origine evento (AWS CLI)

Supponiamo che si desideri che una mappatura dell'origine eventi abbia i seguenti `FilterCriteria`:

```
{
   "Filters": [
      {
          "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
      }
   ]
}
```

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtro utilizzando AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
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 ]}"}]}'
```

Questo create-event-source-mappingcomando crea una nuova mappatura dell'origine degli eventi Amazon SQS per una funzione `my-function` con quanto specificato. `FilterCriteria`

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"Metadata\" : [ 1, 2 ]}"}]}'
```

Si noti che per aggiornare una mappatura di origine eventi, è necessario il suo UUID. È possibile ottenere l'UUID da una chiamata. list-event-source-mappings Lambda restituisce anche l'UUID nella risposta CLI. create-event-source-mapping

Per rimuovere i criteri di filtro da un'origine di eventi, puoi eseguire il update-event-source-mappingcomando seguente con un oggetto vuoto. `FilterCriteria`

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria "{}"
```

Per ulteriori esempi di creazione di filtri di eventi utilizzando il AWS CLI, vedereUtilizzo di filtri con diversi Servizi AWS.

Collegamento dei criteri di filtro a una mappatura dell'origine evento (AWS SAM)

Supponiamo di voler configurare un'origine di eventi in modo AWS SAM da utilizzare i seguenti criteri di filtro:

```
{
   "Filters": [
      {
          "Pattern": "{ \"Metadata\" : [ 1, 2 ] }"
      }
   ]
}
```

Per aggiungere questi criteri di filtro allo strumento di mappatura dell'origine degli eventi, inserisci il seguente frammento nel modello YAML delle origini eventi.

```
FilterCriteria:
  Filters:
    - Pattern: '{"Metadata": [1, 2]}'
```

Per ulteriori informazioni sulla creazione e la configurazione di un AWS SAM modello per la mappatura delle sorgenti di un evento, consultate la EventSourcesezione della Guida per gli AWS SAM sviluppatori. Per altri esempi di creazione di filtri di eventi utilizzando AWS SAM modelli, consulta. Utilizzo di filtri con diversi Servizi AWS

Crittografia dei criteri di filtro

Per impostazione predefinita, Lambda non crittografa l'oggetto dei criteri di filtro. Nei casi d'uso in cui è possibile includere informazioni riservate nell'oggetto dei criteri di filtro, è possibile utilizzare la propria chiave KMS per crittografarlo.

Dopo aver crittografato l'oggetto dei criteri di filtro, puoi visualizzarne la versione in testo semplice utilizzando una GetEventSourceMappingchiamata API. È necessario disporre delle autorizzazioni `kms:Decrypt` per poter visualizzare correttamente i criteri di filtro in testo non crittografato.

> **Note**
>
> Se l'oggetto dei criteri di filtro è crittografato, Lambda oscura il valore del `FilterCriteria` campo nella risposta alle chiamate. ListEventSourceMappings Invece, questo campo viene visualizzato come `null`. Per vedere il vero valore di`FilterCriteria`, usa l'GetEventSourceMappingAPI.
> Per visualizzare il valore decrittografato di `FilterCriteria` nella console, assicurati che il tuo ruolo IAM contenga le autorizzazioni per. GetEventSourceMapping

Puoi specificare la tua chiave KMS tramite la console, l'API, la CLI o CloudFormation.

Per crittografare i criteri di filtro con una chiave KMS di proprietà del cliente (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Selezionare Add trigger (Aggiungi trigger). Se hai già un trigger esistente, scegli la scheda Configurazione, quindi scegli Trigger. Seleziona il trigger esistente e scegli Modifica.
3. Seleziona la casella di controllo accanto a Crittografa con chiave KMS gestita dal cliente.
4. Per Scegli una chiave di crittografia KMS gestita dal cliente, seleziona una chiave abilitata esistente o crea una nuova chiave. A seconda dell'operazione, sono necessarie alcune o tutte le seguenti autorizzazioni: `kms:DescribeKey`, `kms:GenerateDataKey` e `kms:Decrypt`. Utilizza la policy delle chiavi KMS per concedere queste autorizzazioni.

Se utilizzi la tua chiave KMS, le seguenti operazioni API devono essere permesse nella policy della chiave:

- `kms:Decrypt`: deve essere concesso al principale del servizio Lambda regionale (`lambda.AWS_region.amazonaws.com`). Ciò permette a Lambda per decrittografare i dati con questa chiave KMS.
  - Per evitare un problema di confused deputy tra servizi, la policy della chiave utilizza la chiave di condizione globale `aws:SourceArn`. Il valore corretto della chiave `aws:SourceArn` è l'ARN della risorsa dello strumento di mappatura dell'origine degli eventi, quindi puoi aggiungerlo alla tua policy solo dopo averne rilevato l'ARN. Lambda inoltra inoltre le chiavi `aws:lambda:FunctionArn` e `aws:lambda:EventSourceArn` e i rispettivi valori nel contesto di crittografia quando effettua una richiesta di decrittografia a KMS. Affinché la richiesta di decrittografia abbia successo, questi valori devono corrispondere alle condizioni specificate nella policy della chiave. Non è necessario includere EventSourceArn fonti di eventi Kafka gestite in modo autonomo poiché non dispongono di un. EventSourceArn
- `kms:Decrypt`— Deve essere concesso anche al preside che intende utilizzare la chiave per visualizzare i criteri di filtro in chiaro nelle nostre chiamate API. GetEventSourceMappingDeleteEventSourceMapping
- `kms:DescribeKey`: fornisce i dettagli della chiave gestita dal cliente per consentire al principale specificato di utilizzare la chiave.
- `kms:GenerateDataKey`: fornisce le autorizzazioni a Lambda per generare una chiave di dati per crittografare i criteri di filtro per conto del principale specificato (crittografia a busta).

Puoi utilizzarlo AWS CloudTrail per tenere traccia AWS KMS delle richieste che Lambda effettua per tuo conto. Per esempi di CloudTrail eventi, consulta???.

Consigliamo inoltre di utilizzare la chiave di condizione `kms:ViaService` per limitare l'uso della chiave KMS solo alle richieste provenienti da Lambda. Il valore di questa chiave è il principale del servizio Lambda regionale (`lambda.AWS_region.amazonaws.com`). Di seguito è riportata una policy della chiave di esempio che concede tutte le autorizzazioni pertinenti:

Example AWS KMS politica chiave

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"
                }
            }
        }
    ]
}
```

Per utilizzare la propria chiave KMS per crittografare i criteri di filtro, è possibile utilizzare anche il seguente CreateEventSourceMapping AWS CLI comando. Specifica l'ARN della chiave KMS con il flag `--kms-key-arn`.

```
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
```

Se disponi di una mappatura della fonte degli eventi esistente, usa invece il UpdateEventSourceMapping AWS CLI comando. Specifica l'ARN della chiave KMS con il flag `--kms-key-arn`.

```
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
```

Questa operazione sovrascrive qualsiasi chiave KMS specificata in precedenza. Se specifichi il flag `--kms-key-arn` insieme a un argomento vuoto, Lambda smette di usare la chiave KMS per crittografare i criteri di filtro. Torna invece a utilizzare per impostazione predefinita una chiave di proprietà di Amazon.

Per specificare la tua chiave KMS in un CloudFormation modello, usa la `KMSKeyArn` proprietà del tipo di `AWS::Lambda::EventSourceMapping` risorsa. Ad esempio, puoi inserire il seguente frammento di codice nel modello YAML dell'origine eventi.

```
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"
    ...
```

Per poter visualizzare i criteri di filtro crittografati in testo semplice in una chiamata GetEventSourceMappingo in una chiamata DeleteEventSourceMappingAPI, devi disporre delle autorizzazioni. `kms:Decrypt`

A partire dal 6 agosto 2024, il `FilterCriteria` campo non viene più visualizzato nei AWS CloudTrail log di e nelle chiamate DeleteEventSourceMappingAPI se la funzione non utilizza il filtro degli eventi. CreateEventSourceMappingUpdateEventSourceMapping Se la tua funzione utilizza il filtro degli eventi, il campo `FilterCriteria` viene visualizzato come vuoto (`{}`). Puoi comunque visualizzare i criteri di filtro in testo semplice nella risposta alle chiamate GetEventSourceMappingAPI se disponi delle `kms:Decrypt` autorizzazioni per la chiave KMS corretta.

Esempio di voce di registro per CloudTrail le chiamate Create/Update/DeleteEventSourceMapping

Nel seguente AWS CloudTrail esempio di voce di registro per una CreateEventSourceMapping chiamata, `FilterCriteria` viene visualizzata come empty (`{}`) perché la funzione utilizza il filtro degli eventi. Questo è il caso anche se l'oggetto `FilterCriteria` contiene criteri di filtro validi che la funzione utilizza attivamente. Se la funzione non utilizza il filtro degli eventi, CloudTrail non visualizzerà affatto il `FilterCriteria` campo nelle voci di registro.

```
{
    "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"
}
```

Utilizzo di filtri con diversi Servizi AWS

Diversi tipi di origine degli eventi utilizzano valori di chiave differenti per i rispettivi campi di dati. Per filtrare le proprietà dei dati, assicurati di utilizzare la chiave corretta nel modello del filtraggio. La tabella seguente fornisce le chiavi di filtro per ogni supporto Servizio AWS.

Servizio AWS	Chiave di filtraggio
DynamoDB	`dynamodb`
Kinesis	`data`
Amazon MQ	`data`
MSK Amazon	`value`
Apache Kafka gestito dal cliente	`value`
Amazon SQS	`body`

Le sezioni seguenti forniscono esempi di modelli di filtraggio per diversi tipi di origini eventi. Forniscono inoltre definizioni dei formati di dati in entrata supportati e dei formati del corpo dei modelli di filtraggio per ciascun servizio supportato.

- the section called “Filtro eventi”
- the section called “Filtro eventi”
- the section called “Filtro eventi”
- the section called “Filtro eventi”
- the section called “Filtro eventi”

Test delle funzioni Lambda nella console

È possibile testare la funzione Lambda nella console richiamando la funzione con un evento di test. Un evento di test è un input JSON per la funzione. Se la funzione non richiede input, l'evento può essere un documento (`{}`) vuoto.

Quando esegui un test nella console, Lambda invoca la funzione in modo sincrono con l'evento di test. Il runtime della funzione converte il JSON di evento in un oggetto e lo passa al metodo del gestore del codice per l'elaborazione.

> **Creare un evento di test**
>
> Prima di poter eseguire il test nella console, devi creare un evento di test privato o condivisibile.

Invocare funzioni con eventi di test

Testare una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione che desideri testare.
3. Seleziona la scheda Test.
4. Sotto a Evento di test, scegli Crea nuovo evento o Modifica Evento salvato, quindi seleziona l'evento salvato da utilizzare.
5. Facoltativamente, scegli un Modello per l'evento JSON.
6. Scegli Test (Esegui test).
7. Per esaminare i risultati del test, in Execution result (Risultato esecuzione), espandi Details (Dettagli).

Per invocare la funzione senza salvare l'evento di test scegli Test prima di salvare. Questo crea un evento di test non salvato che Lambda conserverà solo per l'intera durata della sessione.

Per i runtime Node.js, Python e Ruby, puoi anche accedere agli eventi di test salvati e non salvati esistenti nella scheda Codice. Utilizza la sezione EVENTI DI TEST per creare, modificare ed eseguire test.

Creazione di eventi di test privati

Gli eventi di test privati sono disponibili solo per il creatore dell'evento e non richiedono autorizzazioni aggiuntive per l'uso. Puoi creare fino a 10 eventi di test per ogni funzione.

Creare un evento di test

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione che desideri testare.
3. Seleziona la scheda Test.
4. Sotto a Test event (evento di test), procedi come segue:
    a. Seleziona un Template (modello).
    b. Inserisci un Nome per l'evento di test.
    c. Nella casella di immissione testo, inserire l'evento di test JSON.
    d. Sotto Event sharing settings (impostazioni di condivisione degli eventi), scegli Private (privato).
5. Scegli Save changes (Salva modifiche).

Per i runtime Node.js, Python e Ruby, puoi anche creare eventi di test nella scheda Codice. Utilizza la sezione EVENTI DI TEST per creare, modificare ed eseguire test.

Creazione di eventi di test condivisibili

Gli eventi di test condivisibili sono eventi di test che puoi condividere con altri utenti nello stesso AWS account. Puoi modificare gli eventi di test condivisibili di altri utenti e richiamare la funzione con essi.

Lambda salva gli eventi di test condivisibili come schemi in un registro di schemi Amazon EventBridge (CloudWatch Events) denominato. `lambda-testevent-schemas` Poiché Lambda utilizza questo registro di sistema per memorizzare e chiamare gli eventi di test condivisibili creati, è consigliabile non modificare tale registro o creare un registro utilizzando il nome `lambda-testevent-schemas`.

Per visualizzare, condividere e modificare gli eventi di test condivisibili, devi disporre delle autorizzazioni per tutte le seguenti operazioni dell'API del registro degli schemi EventBridge (CloudWatch Events):

- `schemas.CreateRegistry`

- schemas.CreateSchema
- schemas.DeleteSchema
- schemas.DeleteSchemaVersion
- schemas.DescribeRegistry
- schemas.DescribeSchema
- schemas.GetDiscoveredSchema
- schemas.ListSchemaVersions
- schemas.UpdateSchema

Ricorda che il salvataggio delle modifiche apportate a un evento di test condivisibile sovrascrive tale evento.

Se non riesci a creare, modificare o visualizzare eventi di test condivisibili, verifica che il tuo account disponga delle autorizzazioni necessarie per queste operazioni. Se disponi delle autorizzazioni necessarie ma non riesci ancora ad accedere agli eventi di test condivisibili, verifica eventuali politiche basate sulle risorse che potrebbero limitare l'accesso al registro (Events). EventBridge CloudWatch

Per creare un evento di test

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione che desideri testare.
3. Seleziona la scheda Test.
4. Sotto a Test event (evento di test), procedi come segue:
    a. Seleziona un Template (modello).
    b. Inserisci un Nome per l'evento di test.
    c. Nella casella di immissione testo, inserire l'evento di test JSON.
    d. Sotto Event sharing settings (impostazioni di condivisione degli eventi), scegli Shareable (condivisibile).
5. Scegli Save changes (Salva modifiche).

Usa eventi di test condivisibili con. AWS Serverless Application Model

Puoi usare AWS SAM per richiamare eventi di test condivisibili. Consultare `sam remote test-event` nella Guida per gli sviluppatori di AWS Serverless Application Model

Eliminare schemi di eventi di test condivisibili

Quando si eliminano eventi di test condivisibili, Lambda li rimuove dal registro `lambda-testevent-schemas`. Se si rimuove l'ultimo evento di test condivisibile dal Registro di sistema, Lambda elimina il Registro di sistema.

Se si elimina la funzione, Lambda non elimina gli schemi di eventi di test condivisibili associati. È necessario pulire queste risorse manualmente dalla console EventBridge (CloudWatch Events).

Stati funzione Lambda

Lambda include un campo State nella configurazione della funzione per tutte le funzioni per indicare quando la funzione è pronta per essere richiamata. `State`fornisce informazioni sullo stato corrente della funzione, incluso se è possibile richiamarla correttamente. Gli stati della funzione non modificano il comportamento delle chiamate della funzione o il modo in cui la funzione esegue il codice.

> **Note**
>
> Le definizioni dello stato delle funzioni differiscono leggermente per SnapStartle funzioni. Per ulteriori informazioni, consulta Lambda SnapStart e stati funzionali.

In molti casi, una tabella DynamoDB è il modo ideale per mantenere lo stato tra le chiamate, poiché fornisce un accesso ai dati a bassa latenza e può essere scalata con il servizio Lambda. Puoi anche archiviare dati in Amazon EFS for Lambda se utilizzi questo servizio e questo fornisce un accesso a bassa latenza allo storage del file system.

Gli stati delle funzioni includono:

- `Pending` – Dopo che Lambda ha creato la funzione, imposta lo stato su sospeso. Mentre lo stato è in sospeso, Lambda tenta di creare o configurare le risorse per la funzione, ad esempio le risorse VPC o EFS. Lambda non richiama una funzione durante lo stato in sospeso. Qualsiasi chiamata o altre operazioni API che operano sulla funzione avranno esito negativo.
- `Active` – La funzione passa allo stato attivo dopo che Lambda ha completato la configurazione e il provisioning delle risorse. Le funzioni possono essere richiamate correttamente solo quando sono attive.
- `Failed` – Indica che la configurazione o il provisioning delle risorse ha riscontrato un errore. Quando la creazione della funzione fallisce, Lambda imposta lo stato della funzione su Fallito ed è necessario eliminare e ricreare la funzione.
- `Inactive` – Una funzione diventa inattiva quando è stata inattiva abbastanza a lungo perché Lambda recuperi le risorse esterne configurate per essa. Quando si tenta di richiamare una funzione che è inattiva, la chiamata non riesce e Lambda imposta la funzione sullo stato in sospeso fino a quando le risorse della funzione non vengono ricreate. Se Lambda non riesce a ricreare le risorse, la funzione viene reimpostata sullo stato inattivo. Potrebbe essere necessario risolvere eventuali errori e ridistribuire la funzione per ripristinarla allo stato attivo.

Se utilizzi flussi di lavoro di automazione basati su SDK o chiami APIs direttamente il servizio di Lambda, assicurati di controllare lo stato di una funzione prima della chiamata per verificare che sia attiva. Puoi farlo con l'azione GetFunctionAPI Lambda o configurando un cameriere utilizzando l'SDK for AWS Java 2.0.

```
aws lambda get-function --function-name my-function --query 'Configuration.[State,
 LastUpdateStatus]'
```

Verrà visualizzato l'output seguente:

```
[
 "Active",
 "Successful"
]
```

Le operazioni seguenti non riescono mentre la creazione della funzione è in attesa:

- Invoke
- UpdateFunctionCode
- UpdateFunctionConfiguration
- PublishVersion

Stati delle funzioni durante gli aggiornamenti

Lambda prevede due operazioni per l'aggiornamento delle funzioni:

- UpdateFunctionCode: aggiorna il pacchetto di distribuzione della funzione
- UpdateFunctionConfiguration: aggiorna la configurazione della funzione

Lambda utilizza l'LastUpdateStatusattributo per tenere traccia dell'avanzamento di queste operazioni di aggiornamento. Mentre è in corso un aggiornamento (quando`"LastUpdateStatus": "InProgress"`):

- Lo stato della funzione rimane`Active`.
- Le chiamate continuano a utilizzare il codice e la configurazione precedenti della funzione fino al completamento dell'aggiornamento.
- Le seguenti operazioni hanno esito negativo:

- UpdateFunctionCode
- UpdateFunctionConfiguration
- PublishVersion
- TagResource

Quando un aggiornamento fallisce (quando`"LastUpdateStatus": "Failed"`):

- Lo stato della funzione rimane`Active`.
- Le chiamate continuano a utilizzare il codice e la configurazione precedenti della funzione.

Example GetFunctionConfiguration risposta

L'esempio seguente è il risultato della GetFunctionConfigurationrichiesta di una funzione in fase di aggiornamento.

```
{
    "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",
    ...
}
```

Informazioni sul comportamento relativo ai nuovi tentativi in Lambda

Quando invochi una funzione esplicitamente, stabilisci la strategia per la gestione degli errori relativi al codice della funzione. Lambda non riprova automaticamente questi tipi di errori per conto dell'utente. Per riprovare, puoi re-invocare manualmente la funzione, inviare l'evento che presenta l'errore a una coda per il debug o ignorare l'errore. Il codice della funzione può essere eseguito completamente, parzialmente o per niente. Se si riprova, accertare che il codice della funzione sia in grado di gestire lo stesso evento più volte senza provocare transazioni duplicate o altri effetti collaterali indesiderati.

Quando si invoca una funzione in modo indiretto, è necessario conoscere il comportamento del nuovo tentativo dell'invoker e qualsiasi servizio che la richiesta incontra. Questo include i seguenti scenari.

- Invocazione asincrona – Lambda tenta due volte gli errori di funzione. Se la funzione non dispone di capacità sufficiente per gestire tutte le richieste in entrata, gli eventi possono attendere in coda per ore o giorni per essere inviati alla funzione. È possibile configurare una coda DLQ sulla funzione per acquisire eventi che non sono stati elaborati. Per ulteriori informazioni, consulta the section called “Code di lettere morte”.
- Mappature evento di origine – Le mappature evento di origine che leggono dai flussi effettuano nuovi tentativi sull'intero batch di elementi. Errori ripetuti bloccano l'elaborazione degli shard interessati finché l'errore non viene risolto o gli elementi non scadono. Per rilevare shard in stallo, è possibile monitorare il parametro Età iteratore.

  Per le mappature origine evento che leggono da una coda, stabilire il tempo trascorso tra i nuovi tentativi e la destinazione degli eventi non riusciti configurando il timeout di visibilità e la policy di reindirizzamento sulla coda di origine. Per ulteriori informazioni, consulta In che modo Lambda elabora i record provenienti da origini eventi basate su flussi e code e gli argomenti specifici del servizio in Richiamare Lambda con eventi di altri servizi AWS.
- Servizi AWS – I servizi AWS possono invocare la funzione in modo sincrono o asincrono. Per l'invocazione sincrona, il servizio decide se effettuare un nuovo tentativo. Ad esempio, le operazioni batch Amazon S3 ritentano l'operazione se la funzione Lambda restituisce un codice di risposta `TemporaryFailure`. I servizi che il proxy richiede da un utente o un client upstream, possono disporre di una strategia per nuovi tentativi o possono inoltrare la risposta di errore di nuovo al richiedente. Ad esempio, API Gateway inoltra sempre di nuovo la risposta di errore al richiedente.

Per l'invocazione asincrona, la logica dei tentativi è la stessa indipendentemente dall'origine di invocazione. Per impostazione predefinita, Lambda ritenta una invocazione asincrona fallita un massimo di due volte. Per ulteriori informazioni, consulta In che modo Lambda gestisce gli errori e i nuovi tentativi con una invocazione asincrona.

- Altri account e client – Quando si concede l'accesso ad altri account, è possibile usare policy basate su risorse per limitare i servizi e le risorse che possono configurare per invocare la funzione. Per proteggere la funzione dal sovraccarico, considerare di inserire un livello API davanti alla funzione con Amazon API Gateway.

Per gestire gli errori nelle applicazioni Lambda, Lambda si integra con servizi quali Amazon CloudWatch e AWS X-Ray. È possibile utilizzare una combinazione di log, parametri, allarmi e traccia per rilevare e identificare rapidamente problemi di codice della funzione, API e altre risorse che supportano l'applicazione. Per ulteriori informazioni, consulta Monitoraggio, debug e risoluzione dei problemi delle funzioni Lambda.

Usa il rilevamento di un ciclo ricorsivo Lambda per prevenire loop infiniti

Quando configuri una funzione Lambda di modo che invii l'output sullo stesso servizio o risorsa che richiama la funzione, è possibile che venga creato un ciclo ricorsivo infinito. Ad esempio, una funzione Lambda potrebbe scrivere un messaggio a una coda di Amazon Simple Queue Service (Amazon SQS), che quindi richiama la stessa funzione. Questa invocazione fa sì che la funzione scriva un altro messaggio nella coda, che a sua volta richiama nuovamente la funzione.

I loop ricorsivi involontari possono comportare addebiti imprevisti. Account AWS I cicli ricorsivi possono anche far sì che Lambda si dimensioni e utilizzi tutta la simultaneità disponibile del tuo account. Per contribuire a ridurre l'impatto dei cicli involontari, Lambda è in grado di rilevare determinati tipi di cicli ricorsivi poco dopo che si sono verificati. Per impostazione predefinita, quando Lambda rileva un ciclo ricorsivo, interrompe l'invocazione della funzione e invia una notifica. Se il tuo progetto utilizza intenzionalmente modelli ricorsivi, puoi modificare la configurazione predefinita di una funzione per consentirne l'invocazione ricorsiva. Per ulteriori informazioni, consulta the section called “Consentire l'esecuzione di una funzione Lambda in un ciclo ricorsivo”.

Sections

- Comprendere il rilevamento dei cicli ricorsivi
- Supportato e Servizi AWS SDKs
- Notifiche dei cicli ricorsivi
- Risposta alle notifiche di rilevamento dei cicli ricorsivi
- Consentire l'esecuzione di una funzione Lambda in un ciclo ricorsivo
- Regioni supportate per il rilevamento ricorsivo del loop Lambda

Comprendere il rilevamento dei cicli ricorsivi

Il rilevamento dei cicli ricorsivi in Lambda si basa sul tracciamento gli eventi. Lambda è un servizio di elaborazione basato su eventi che esegue il codice della funzione quando si verificano determinati eventi. Ad esempio, quando un elemento viene aggiunto a una coda Amazon SQS o a un argomento di Amazon Simple Notification Service (Amazon SNS). Lambda trasmette gli eventi alla funzione come oggetti JSON, che contengono informazioni sulla modifica dello stato del sistema. Quando un evento causa l'esecuzione di una funzione, si parla di invocazione.

Per rilevare i cicli ricorsivi, Lambda utilizza le intestazioni di tracciamento AWS X-Ray. Quando dei Servizi AWS che supportano il rilevamento dei cicli ricorsivi inviano eventi a Lambda, tali eventi vengono automaticamente annotati con metadati. Quando la funzione Lambda scrive uno di questi eventi su un altro supportato Servizio AWS utilizzando una versione supportata di un AWS SDK, aggiorna questi metadati. I metadati aggiornati includono il numero di volte in cui l'evento ha richiamato la funzione.

> **Note**
>
> Non è necessario abilitare il tracciamento attivo X-Ray per far funzionare questa funzionalità. Il rilevamento dei cicli ricorsivi è attivato per impostazione predefinita per tutti i clienti AWS . L'utilizzo della funzionalità è gratuito.

Una catena di richieste è una sequenza di invocazioni Lambda causate dallo stesso evento di attivazione. Ad esempio, immagina che una coda Amazon SQS richiami la tua funzione Lambda. La funzione Lambda invia quindi l'evento elaborato alla stessa coda di Amazon SQS, che richiama nuovamente la funzione. In questo esempio, ogni invocazione della funzione rientra nella stessa catena di richieste.

Se la tua funzione viene richiamata più di 16 volte nella stessa catena di richieste, Lambda interrompe automaticamente la successiva invocazione della funzione in quella catena di richieste e invia una notifica all'utente. Ad esempio, se la funzione è configurata con più trigger, le invocazioni da altri trigger non sono interessate.

> **Note**
>
> Quando l'impostazione di `maxReceiveCount` sulla policy di reindirizzamento della coda di origine è superiore a 16, la protezione da ricorsione di Lambda non impedisce ad Amazon SQS di ritentare il messaggio dopo che un loop ricorsivo è stato rilevato e terminato. Quando Lambda rileva un ciclo ricorsivo e annulla le chiamate successive, restituisce `RecursiveInvocationException` allo strumento di mappatura dell'origine degli eventi. Questo incrementa il valore `receiveCount` sul messaggio. Lambda continua a riprovare il messaggio e continua a bloccare le chiamate di funzione, finché Amazon SQS non determina che il limite `maxReceiveCount` è stato superato e invia il messaggio alla coda DLQ configurata.

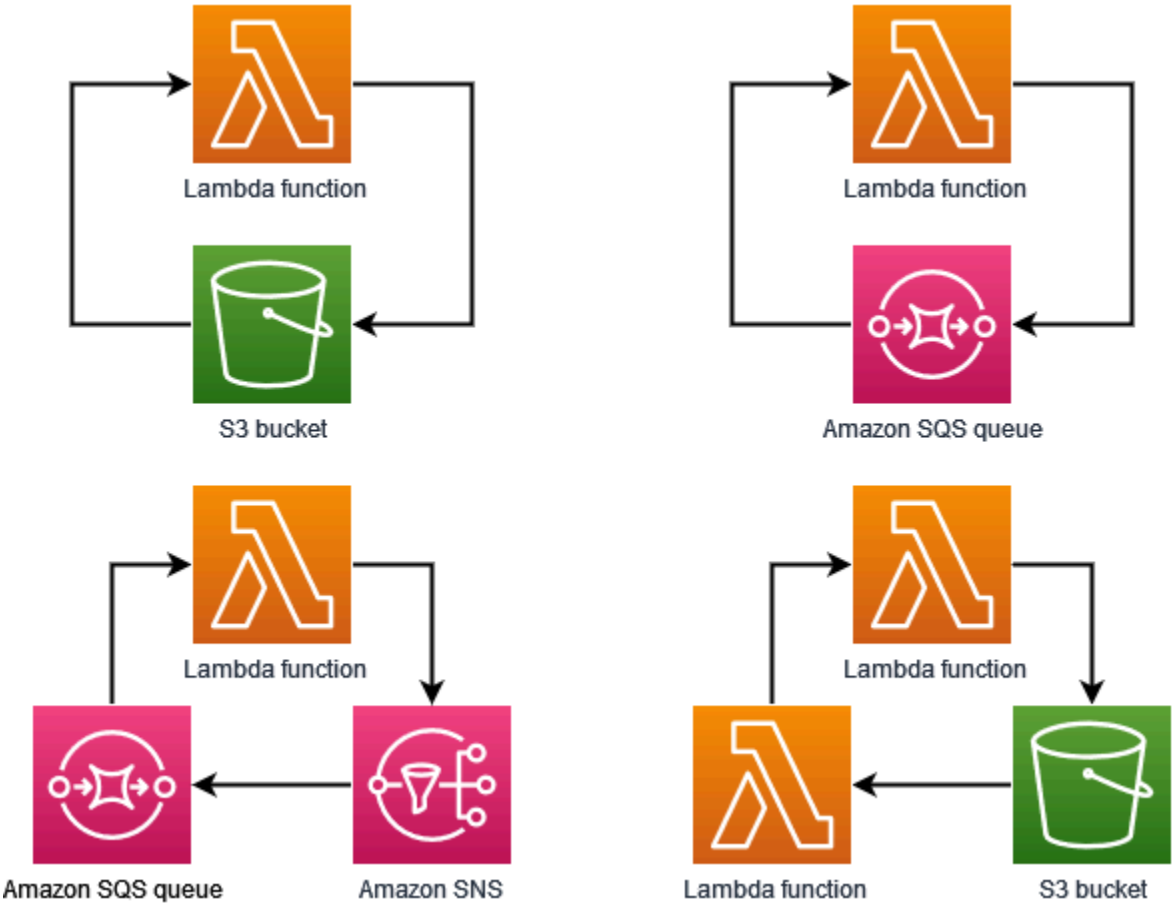
Se hai configurato una destinazione in caso di errore o una coda DLQ per la funzione, Lambda invia anche l'evento dall'invocazione interrotta alla destinazione o alla coda DLQ. Quando configuri una coda di destinazione o di lettere morte per la tua funzione, assicurati di non utilizzare un trigger di evento o una mappatura della fonte di eventi utilizzata anche dalla tua funzione. Se invii eventi alla stessa risorsa che richiama la tua funzione, puoi creare un altro ciclo ricorsivo e anche questo ciclo verrà interrotto. Se si disattiva il rilevamento del ciclo di ricorsione, questo ciclo non verrà interrotto.

Supportato e Servizi AWS SDKs

Lambda è in grado di rilevare solo i loop ricorsivi che includono determinati loop supportati. Servizi AWS Affinché i loop ricorsivi vengano rilevati, la funzione deve utilizzare anche uno dei circuiti supportati. AWS SDKs

Supportato Servizi AWS

Lambda attualmente rileva i cicli ricorsivi tra le tue funzioni, Amazon SQS, Amazon S3 e Amazon SNS. Lambda rileva anche i cicli ricorsivi composti solo da funzioni Lambda, che possono richiamarsi reciprocamente in modo sincrono o asincrono. I diagrammi seguenti mostrano alcuni esempi di ciclo ricorsivo che Lambda è in grado di rilevare:



Quando un altro Servizio AWS tipo Amazon DynamoDB fa parte del ciclo, Lambda al momento non è in grado di rilevarlo e fermarlo.

Poiché attualmente Lambda rileva solo loop ricorsivi che coinvolgono Amazon SQS, Amazon S3 e Amazon SNS, è ancora possibile che i loop che coinvolgono altri possano comportare un utilizzo non intenzionale delle funzioni Servizi AWS Lambda.

Per evitare che ti vengano addebitati addebiti imprevisti Account AWS, ti consigliamo di configurare gli CloudWatchallarmi Amazon per avvisarti di modelli di utilizzo insoliti. Ad esempio, puoi CloudWatch configurare la notifica dei picchi nella concorrenza o nelle chiamate della funzione Lambda. Puoi anche configurare un avviso di fatturazione che ti segnali quando la spesa nel tuo account supera una soglia da te specificata. In alternativa, puoi utilizzare AWS Cost Anomaly Detection per ricevere avvisi in merito a schemi di fatturazione insoliti.

Supportato AWS SDKs

Affinché Lambda possa rilevare i cicli ricorsivi, la funzione deve utilizzare un SDK delle versioni seguenti o successive:

Runtime	Versione AWS SDK minima richiesta
Node.js	2.1147.0 (SDK versione 2) 3.105.0 (SDK versione 3)
Python	1.24.46 (boto3) 1.27.46 (botocore)
Java 8 e Java 11	2.17.135
Java 17	2,20,81
Java 21	2,21,24
.NET	3,7,293,0
Ruby	3,134,0
PHP	3,232,0

Runtime	Versione AWS SDK minima richiesta
Go	V2 SDK 1.57.0

Alcuni runtime Lambda come Python e Node.js includono una versione dell'SDK. AWS Se la versione SDK inclusa nel runtime della funzione è inferiore al minimo richiesto, puoi aggiungere una versione supportata dell'SDK al pacchetto di implementazione della funzione. Puoi aggiungere una versione supportata dell'SDK alla funzione anche utilizzando un livello Lambda. Per un elenco dei componenti SDKs inclusi in ogni runtime Lambda, consulta. Runtime Lambda

Notifiche dei cicli ricorsivi

Quando Lambda interrompe un ciclo ricorsivo, ricevi notifiche tramite Health Dashboard e tramite e-mail. Puoi anche utilizzare le CloudWatch metriche per monitorare il numero di chiamate ricorsive interrotte da Lambda.

Health Dashboard notifiche

Quando Lambda interrompe una chiamata ricorsiva, Health Dashboard visualizza una notifica nella pagina Lo stato del tuo account, in Problemi aperti e recenti. Tieni presente che possono essere necessarie fino a 3,5 ore dopo che Lambda interrompe una chiamata ricorsiva prima che questa notifica venga visualizzata. Per ulteriori informazioni sulla visualizzazione degli eventi dell'account in Health Dashboard, consulta Getting started with your AWS Health Dashboard — Lo stato del tuo account nella AWS Health User Guide.

Avvisi via e-mail

Quando Lambda interrompe per la prima volta un'invocazione ricorsiva della funzione, ti invia un avviso e-mail. Lambda invia un massimo di un'e-mail ogni 24 ore per ogni funzione del tuo Account AWS. Dopo l'invio di una notifica e-mail da parte di Lambda, non riceverai altre e-mail per la stessa funzione per altre 24 ore, anche se Lambda interrompe ulteriori invocazioni ricorsive della funzione. Tieni presente che possono essere necessarie fino a 3,5 ore dopo che Lambda interrompe una chiamata ricorsiva prima di ricevere questo avviso e-mail.

Lambda invia avvisi e-mail a ciclo ricorsivo al contatto principale Account AWS dell'account e al contatto operativo alternativo del tuo account. Per informazioni sulla visualizzazione o l'aggiornamento degli indirizzi e-mail nel tuo account, consulta la pagina Aggiornamento delle informazioni di contatto della Guida generale di AWS .

CloudWatch Metriche Amazon

La CloudWatch metrica `RecursiveInvocationsDropped` registra il numero di chiamate di funzioni che Lambda ha interrotto perché la funzione è stata richiamata più di circa 16 volte in una singola catena di richieste. Lambda emette questo parametro non appena interrompe un'invocazione ricorsiva. Per visualizzare questa metrica, segui le istruzioni relative alla visualizzazione delle metriche sulla console e scegli la metrica. CloudWatch `RecursiveInvocationsDropped`

Risposta alle notifiche di rilevamento dei cicli ricorsivi

Quando la funzione viene richiamata più di 16 volte dallo stesso evento di attivazione, Lambda interrompe la successiva invocazione della funzione per quell'evento per interrompere il ciclo ricorsivo. Per evitare il ripetersi di un ciclo ricorsivo interrotto da Lambda, procedi come segue:

- Riduci la simultaneità disponibile della funzione a zero, il che limita tutte le invocazioni future.
- Rimuovi o disabilita il trigger o la mappatura dell'origine degli eventi che richiama la tua funzione.
- Identifica e correggi i difetti del codice che riscrivono gli eventi nella AWS risorsa che richiama la tua funzione. Una fonte comune di difetti è l'utilizzo di variabili per definire l'origine e la destinazione degli eventi di una funzione. Verifica di non utilizzare lo stesso valore per entrambe le variabili.

Inoltre, se l'origine degli eventi della funzione Lambda è una coda Amazon SQS, valuta la possibilità di configurare una coda DLQ nella coda di origine.

> **Note**
>
> Assicurati di configurare la coda DLQ sulla coda di origine, non sulla funzione Lambda. La coda DLQ che si configura su una funzione viene utilizzata per la coda di invocazione asincrona della funzione, non per le code di origine eventi.

Se l'origine degli eventi è un argomento di Amazon SNS, valuta la possibilità di aggiungere una destinazione in caso di errore per la funzione.

Azzeramento della simultaneità disponibile per la funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione .
3. Scegli Limitatore.

4. Nella finestra di dialogo Limita la tua funzione, scegli Conferma.

Rimozione di un trigger o di una mappatura dell'origine degli eventi per la funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione .
3. Scegli la scheda Configurazione, quindi scegli Trigger.
4. In Trigger, seleziona il trigger o la mappatura dell'origine degli eventi che desideri eliminare, quindi scegli Elimina.
5. Nella finestra di dialogo Elimina trigger, scegli Elimina.

Disabilitazione di una mappatura dell'origine degli eventi per la funzione (AWS CLI)

1. Per trovare l'UUID per la mappatura delle sorgenti degli eventi che desideri disabilitare, esegui il AWS Command Line Interface comando ().AWS CLIlist-event-source-mappings

```
aws lambda list-event-source-mappings
```

2. Per disabilitare la mappatura delle sorgenti degli eventi, esegui il comando seguente. AWS CLI update-event-source-mapping

```
aws lambda update-event-source-mapping --function-name MyFunction \
--uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 --no-enabled
```

Consentire l'esecuzione di una funzione Lambda in un ciclo ricorsivo

Se il tuo progetto utilizza intenzionalmente un ciclo ricorsivo, puoi modificare la configurazione predefinita di una funzione Lambda per consentirne l'invocazione ricorsiva. È preferibile evitare l'uso di cicli ricorsivi nel codice. Gli errori di implementazione possono portare a invocazioni ricorsive che utilizzano tutta la concorrenza disponibile e all'addebito Account AWS di addebiti imprevisti sul tuo account.

> **Important**
>
> Se utilizzi cicli ricorsivi, trattali con attenzione. Implementa i guardrail basati sulle best practice per ridurre al minimo i rischi di errori di implementazione. Per ulteriori informazioni

sulle best practice per l'utilizzo di modelli ricorsivi, consulta la pagina Schemi ricorsivi che causano funzioni Lambda indeterminate in Serverless Land.

Puoi configurare funzioni per consentire loop ricorsivi utilizzando la console Lambda, il AWS Command Line Interface (AWS CLI) e l'API. PutFunctionRecursionConfig Puoi anche configurare l'impostazione di rilevamento ricorsivo del loop di una funzione in and. AWS SAM CloudFormation

Per impostazione predefinita, Lambda rileva e termina i cicli ricorsivi. A meno che il tuo progetto non utilizzi intenzionalmente un ciclo ricorsivo, ti consigliamo di non modificare la configurazione predefinita delle funzioni.

Nota che quando configuri una funzione per consentire i loop ricorsivi, la metrica non viene CloudWatch emessa. `RecursiveInvocationsDropped`

Console

Consentire l'esecuzione di una funzione Lambda in un ciclo ricorsivo (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione per aprire la pagina dei dettagli della funzione.
3. Scegli la scheda Configurazione, quindi scegli Rilevamento di simultaneità e ricorsione.
4. Oltre al rilevamento di cicli ricorsivi, scegli Modifica.
5. Seleziona Consenti cicli ricorsivi.
6. Scegli Save (Salva).

AWS CLI

Puoi utilizzare l'PutFunctionRecursionConfigAPI per consentire alla tua funzione di essere richiamata in un ciclo ricorsivo. Specifica `Allow` per il parametro del ciclo ricorsivo. Ad esempio, puoi chiamare questa API con il comando: `put-function-recursion-config` AWS CLI

```
aws lambda put-function-recursion-config --function-name yourFunctionName --
recursive-loop Allow
```

Puoi riportare la configurazione della funzione all'impostazione predefinita in modo che Lambda termini i cicli ricorsivi quando li rileva. Modifica la configurazione della tua funzione utilizzando la console Lambda o la AWS CLI.

Console

Per configurare una funzione in modo che i cicli ricorsivi vengano terminati (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione per aprire la pagina dei dettagli della funzione.
3. Scegli la scheda Configurazione, quindi scegli Rilevamento di simultaneità e ricorsione.
4. Oltre al rilevamento di cicli ricorsivi, scegli Modifica.
5. Seleziona Termina i cicli ricorsivi.
6. Scegli Save (Salva).

AWS CLI

Puoi utilizzare l'PutFunctionRecursionConfigAPI per configurare la tua funzione in modo che Lambda termini i loop ricorsivi quando li rileva. Specifica `Terminate` per il parametro del ciclo ricorsivo. Ad esempio, puoi chiamare questa API con il comando: `put-function-recursion-config` AWS CLI

```
aws lambda put-function-recursion-config --function-name yourFunctionName --
recursive-loop Terminate
```

Regioni supportate per il rilevamento ricorsivo del loop Lambda

Il rilevamento ricorsivo del loop Lambda è supportato in tutte le regioni commerciali ad eccezione del Messico (Centrale) e dell'Asia Pacifico (Nuova Zelanda).

Creazione e gestione della funzione Lambda URLs

Un URL della funzione è un endpoint HTTP(S) dedicato alla funzione Lambda. È possibile creare e configurare un URL della funzione tramite la console Lambda o l'API Lambda.

> **Tip**
>
> Lambda offre due modi per richiamare una funzione tramite un endpoint HTTP: funzione e Amazon API URLs Gateway. In caso di dubbi su quale sia il metodo migliore per il tuo caso d'uso, consulta the section called “Funzione URLs e Amazon API Gateway”.

Quando si crea un URL della funzione, Lambda genera automaticamente un endpoint URL univoco. Dopo aver creato un URL della funzione, il suo endpoint URL non cambia mai. Gli endpoint URL della funzione hanno il formato seguente:

```
https://<url-id>.lambda-url.<region>.on.aws
```

> **Note**
>
> URLs Le funzioni non sono supportate nei seguenti paesi Regioni AWS: Asia Pacifico (Hyderabad) (`ap-south-2`), Asia Pacifico (Melbourne) (`ap-southeast-4`), Asia Pacifico (Malesia) (), Asia Pacifico (Nuova Zelandaap-southeast-5) (), Asia Pacifico (Tailandiaap-southeast-6) () (), Asia Pacifico (Taipeiap-southeast-7) (), Canada occidentale (Calgary) (`ap-east-2`), Europa (Spagna) () `ca-west-1` (), Europa (Zurigo) (`eu-south-2`), Israele (Tel Aviv) `eu-central-2` (), e Medio Oriente (Emirati Arabi Uniti) (`il-central-1`). `me-central-1`

URLs Le funzioni sono abilitate al dual stack, supportano IPv4 e IPv6. Dopo aver configurato un URL della funzione per la funzione utilizzata, è possibile richiamare la funzione attraverso il relativo endpoint HTTP(S) tramite un browser Web, curl, Postman o un client HTTP.

> **Note**
>
> Puoi accedere all'URL della funzione solo tramite Internet pubblico. Sebbene le funzioni Lambda supportino AWS PrivateLink, le funzioni no URLs .

La funzione Lambda URLs utilizza politiche basate sulle risorse per la sicurezza e il controllo degli accessi. La funzione supporta URLs anche le opzioni di configurazione CORS (Cross-Origin Resource Sharing).

È possibile applicare la funzione URLs a qualsiasi alias di funzione o alla versione della funzione `$LATEST` non pubblicata. Non è possibile aggiungere un URL della funzione a nessun'altra versione della funzione.

La sezione seguente mostra come creare e gestire l'URL di una funzione utilizzando la console Lambda e il AWS CLI modello CloudFormation

Argomenti

- Creazione di un URL della funzione (console)
- Creazione di un URL di funzione (AWS CLI)
- Aggiungere l'URL di una funzione a un CloudFormation modello
- Cross-Origin Resource Sharing (CORS)
- Funzione di limitazione URLs
- Funzione di disattivazione URLs
- Funzione di cancellazione URLs
- Controlla l'accesso alla funzione Lambda URLs
- Richiamo di URL di funzioni Lambda
- Monitoraggio degli URL della funzione Lambda
- Selezionare un metodo per richiamare la funzione Lambda tramite una richiesta HTTP
- Tutorial: creazione di un endpoint webhook utilizzando l'URL di una funzione Lambda

Creazione di un URL della funzione (console)

Segui queste fasi per creare un URL della funzione usando la console.

Per creare un URL di funzione per una funzione esistente

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione per la quale desideri creare l'URL della funzione.
3. Scegli la scheda Configurazione, quindi scegli URL della funzione.
4. Scegli Crea URL della funzione.

5. Per il tipo di autenticazione, scegliete AWS_IAMo NONE. Per ulteriori informazioni sull'autenticazione dell'URL della funzione, consulta Controllo accessi.
6. (Opzionale) Seleziona Configura CORS e quindi configura le impostazioni CORS per l'URL della funzione. Per ulteriori informazioni sulla funzionalità CORS, consulta Cross-Origin Resource Sharing (CORS).
7. Scegli Save (Salva).

In questo modo viene creato un URL di funzione per la versione non pubblicata di `$LATEST` della funzione. L'URL della funzione viene visualizzato nella sezione Panoramica della funzione della console.

Per creare l'URL di una funzione per un alias esistente

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione con l'alias per cui desideri creare l'URL della funzione.
3. Seleziona la scheda Aliases (Alias) e quindi scegli il nome dell'alias per cui desideri creare l'URL della funzione.
4. Scegli la scheda Configuration (Configurazione), quindi scegli Function URL (URL della funzione).
5. Scegli Crea URL della funzione.
6. Per il tipo di autenticazione, scegli AWS_IAMo NONE. Per ulteriori informazioni sull'autenticazione dell'URL della funzione, consulta Controllo accessi.
7. (Opzionale) Seleziona Configura CORS e quindi configura le impostazioni CORS per l'URL della funzione. Per ulteriori informazioni sulla funzionalità CORS, consulta Cross-Origin Resource Sharing (CORS).
8. Scegli Save (Salva).

In questo modo viene creato un URL della funzione per l'alias della funzione. L'URL della funzione viene visualizzato nella sezione Panoramica della funzione della console per l'alias.

Per creare una nuova funzione con l'URL di una funzione

Creazione di una nuova funzione con un URL della funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli Crea funzione.

3. In Basic information (Informazioni di base) eseguire queste operazioni:
   a. In Function name (Nome funzione), inserisci un nome per la funzione, ad esempio **`my-function`**.
   b. Per Runtime, scegliete il linguaggio di runtime che preferite, ad esempio Node.js 24.
   c. Per Architecture (Architettura), scegli x86_64 o arm64.
   d. Espandi Permissions (Autorizzazioni), quindi scegli se creare un nuovo ruolo di esecuzione o usarne uno esistente.
4. Espandi Advanced settings (Impostazioni avanzate) e quindi seleziona Function URL (URL della funzione).
5. Per il tipo di autenticazione, scegliete AWS_IAMo NONE. Per ulteriori informazioni sull'autenticazione dell'URL della funzione, consulta Controllo accessi.
6. (Opzionale) Seleziona Configure cross-origin resource sharing (CORS) (Configura CORS). Selezionando questa opzione durante la creazione della funzione, l'URL della funzione consente le richieste da tutte le origini per impostazione predefinita. È possibile modificare le impostazioni CORS per l'URL della funzione dopo aver creato la funzione. Per ulteriori informazioni sulla funzionalità CORS, consulta Cross-Origin Resource Sharing (CORS).
7. Scegli Crea funzione.

In questo modo viene creata una nuova funzione con un URL della funzione per la versione non pubblicata di `$LATEST` della funzione. L'URL della funzione viene visualizzato nella sezione Function overview (Panoramica della funzione) della console.

Creazione di un URL di funzione (AWS CLI)

Per creare un URL di funzione per una funzione Lambda esistente utilizzando AWS Command Line Interface (AWS CLI), esegui il comando seguente:

```
aws lambda create-function-url-config \
    --function-name my-function \
    --qualifier prod \ // optional
    --auth-type AWS_IAM
    --cors-config {AllowOrigins="https://example.com"} // optional
```

Un URL di funzione viene aggiunto al qualificatore **`prod`** per la funzione **`my-function`**. Per ulteriori informazioni su questi parametri di configurazione, consulta CreateFunctionUrlConfig nella documentazione di riferimento delle API.

> **Note**
>
> Per creare l'URL di una funzione tramite AWS CLI, la funzione deve già esistere.

Aggiungere l'URL di una funzione a un CloudFormation modello

Per aggiungere una `AWS::Lambda::Url` risorsa al CloudFormation modello, utilizzate la seguente sintassi:

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
```

Parameters

- (Obbligatorio)`AuthType`: definisce il tipo di autenticazione per l'URL della funzione. I valori possibili sono `AWS_IAM` o `NONE`. Per limitare l'accesso solo agli utenti autenticati, imposta su `AWS_IAM`. Per ignorare l'autenticazione IAM e consentire a qualsiasi utente di effettuare richieste alla propria funzione, imposta su `NONE`.
- (Opzionale)`Cors`: definisce le impostazioni CORS per l'URL della funzione. Per aggiungere `Cors` alla `AWS::Lambda::Url` risorsa in CloudFormation, utilizzate la seguente sintassi.

Example AWS::Lambda::Url.Cors (JSON)

```
{
  "AllowCredentials" : Boolean,
  "AllowHeaders" : [ String, ... ],
  "AllowMethods" : [ String, ... ],
  "AllowOrigins" : [ String, ... ],
  "ExposeHeaders" : [ String, ... ],
  "MaxAge" : Integer
}
```

Example AWS::Lambda::Url.Cors (YAML)

```
  AllowCredentials: Boolean
  AllowHeaders:
    - String
  AllowMethods:
    - String
  AllowOrigins:
    - String
  ExposeHeaders:
    - String
  MaxAge: Integer
```

- (Opzionale) `Qualifier`: il nome dell'alias.
- (Obbligatorio)`TargetFunctionArn`: il nome o l'Amazon Resource Name (ARN) della funzione Lambda. I formati dei nomi validi includono quanto segue:
  - Nome funzione – `my-function`
  - ARN funzione – `arn:aws:lambda:us-west-2:123456789012:function:my-function`
  - ARN parziale – `123456789012:function:my-function`.

Cross-Origin Resource Sharing (CORS)

Per definire in che modo le diverse origini possono accedere all'URL della funzione, utilizza cross-origin resource sharing (CORS). Si consiglia di configurare CORS se si intende chiamare l'URL della funzione da un dominio diverso. Lambda supporta le seguenti intestazioni CORS per la funzione. URLs

Intestazioni CORS	Proprietà di configurazione CORS	Valori di esempio

Access-Control-Allow-Origin	`AllowOrigins`	`*` (consente tutte le origini)
		`https://www.example.com`
		`http://localhost:60905`

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`

Quando configuri CORS per l'URL di una funzione utilizzando la console Lambda o il AWS CLI, Lambda aggiunge automaticamente le intestazioni CORS a tutte le risposte tramite l'URL della funzione. In alternativa, è possibile aggiungere manualmente le intestazioni CORS alla risposta della funzione. Se ci sono intestazioni in conflitto, il comportamento previsto dipende dal tipo di richiesta:

- Per le richieste di preflight come le richieste OPTIONS, le intestazioni CORS configurate nella funzione URL hanno la precedenza. Lambda restituisce solo queste intestazioni CORS nella risposta.
- Per le richieste non preflight come le richieste GET o POST, Lambda restituisce sia le intestazioni CORS configurate nella funzione URL, sia le intestazioni CORS restituite dalla funzione. Ciò può comportare intestazioni CORS duplicate nella risposta. È possibile che venga visualizzato un errore simile al seguente: `The 'Access-Control-Allow-Origin' header contains multiple values '*, *', but only one is allowed.`

In generale, consigliamo di configurare tutte le impostazioni CORS nella funzione URL anziché inviare manualmente le intestazioni CORS nella risposta della funzione.

Funzione di limitazione URLs

Il throttling limita la frequenza con cui la funzione elabora le richieste. Ciò è utile in molte situazioni, ad esempio per impedire alla funzione di sovraccaricare le risorse a valle o per gestire un improvviso aumento delle richieste.

È possibile limitare la frequenza delle richieste elaborate dalla funzione Lambda tramite un URL di funzione configurando la simultaneità riservata. La simultaneità riservata limita il numero massimo di richiami simultanei della funzione. La frequenza massima di richieste al secondo (RPS) della funzione equivale a 10 volte la simultaneità riservata configurata. Ad esempio, se si configura la funzione con una simultaneità riservata di 100, l'RPS massimo è 1.000.

Ogni volta che la simultaneità della funzione supera la simultaneità riservata, l'URL della funzione restituisce un codice di stato HTTP `429`. Se la funzione riceve una richiesta che supera 10 volte l'RPS massimo in base alla simultaneità riservata configurata, viene ricevuto anche un errore HTTP `429`. Per ulteriori informazioni sulla simultaneità riservata, consulta Configurazione della simultaneità riservata per una funzione.

Funzione di disattivazione URLs

In caso di emergenza, potresti voler rifiutare tutto il traffico verso l'URL della funzione. Per disattivare l'URL della funzione, imposta la simultaneità riservata su zero. In questo modo vengono limitate tutte le richieste all'URL della funzione, con conseguenti risposte di stato HTTP `429`. Per riattivare l'URL della funzione, elimina la configurazione di simultaneità riservata o imposta la configurazione su una quantità maggiore di zero.

Funzione di cancellazione URLs

Quando si elimina un URL di funzione, non è possibile ripristinarlo. La creazione di un nuovo URL di funzione determinerà un indirizzo URL diverso.

> **Note**
>
> Se elimini l'URL della funzione con il tipo di autenticazione `NONE`, Lambda non elimina automaticamente la policy basata sulle risorse associata. Se desideri eliminare questa policy, dovrai farlo manualmente.

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere il nome della funzione.
3. Scegli la scheda Configurazione, quindi scegli URL della funzione.
4. Scegli Elimina.
5. Inserisci la parola delete (elimina) nel campo per confermare l'eliminazione.
6. Scegli Elimina.

> **Note**
>
> Quando si elimina una funzione con una funzione URL, Lambda elimina in modo asincrono la funzione URL. Se crei immediatamente una nuova funzione con lo stesso nome nello stesso account, è possibile che la funzione URL originale venga mappata alla nuova funzione anziché eliminata.

Controlla l'accesso alla funzione Lambda URLs

> **Note**
>
> A partire da ottobre 2025, la nuova funzione URLs richiederà entrambe le `lambda:InvokeFunctionUrl` autorizzazioni`lambda:InvokeFunction`.

Puoi controllare l'accesso alla tua funzione Lambda URLs utilizzando il AuthTypeparametro combinato con le politiche basate sulle risorse allegate alla tua funzione specifica. La configurazione di questi due componenti determina chi può richiamare o eseguire altre azioni amministrative sull'URL della funzione.

Il parametro `AuthType` determina il modo in cui Lambda autentica o autorizza le richieste all'URL della funzione. Quando configuri l'URL della funzione, è necessario specificare una delle seguenti opzioni `AuthType`:

- `AWS_IAM`— Lambda utilizza AWS Identity and Access Management (IAM) per autenticare e autorizzare le richieste in base alla policy di identità del principale IAM e alla policy basata sulle risorse della funzione. Scegli questa opzione se desideri che solo gli utenti e i ruoli autenticati invochino la tua funzione utilizzando l'URL della funzione.

- `NONE`: Lambda non esegue alcuna autenticazione prima di richiamare la funzione. Tuttavia, la policy basata sulle risorse della funzione è sempre valida e deve concedere l'accesso pubblico prima che l'URL della funzione possa ricevere richieste. Scegli questa opzione per consentire l'accesso pubblico e non autenticato all'URL della funzione.

Per ulteriori approfondimenti sulla sicurezza, puoi utilizzare AWS Identity and Access Management Access Analyzer per ottenere un'analisi completa dell'accesso esterno all'URL della tua funzione. IAM Access Analyzer controlla inoltre le autorizzazioni nuove o aggiornate sulle funzioni Lambda per aiutarti a identificare le autorizzazioni che garantiscono l'accesso pubblico e tra account. Puoi utilizzare IAM Access Analyzer gratuitamente. Per le nozioni di base su IAM Access Analyzer, consulta Utilizzo di AWS IAM Access Analyzer.

Questa pagina contiene esempi di politiche basate sulle risorse per entrambi i tipi di autenticazione e come creare queste politiche utilizzando il funzionamento dell'AddPermissionAPI o la console Lambda. Per informazioni su come richiamare l'URL della funzione dopo aver impostato le autorizzazioni, consulta. Richiamo di URL di funzioni Lambda

Argomenti

- Utilizzo del tipo di autenticazione AWS_IAM
- Utilizzo del tipo di autenticazione NONE
- Governance e controllo degli accessi

Utilizzo del tipo di autenticazione **`AWS_IAM`**

Se scegli il tipo di `AWS_IAM` autenticazione, gli utenti che devono richiamare l'URL della funzione Lambda devono disporre delle autorizzazioni and. `lambda:InvokeFunctionUrl` `lambda:InvokeFunction` A seconda di chi effettua la richiesta di chiamata, potrebbe essere necessario concedere questa autorizzazione utilizzando una politica basata sulle risorse.

Se il principale che effettua la richiesta corrisponde all'URL della funzione, deve disporre delle `lambda:InvokeFunction` autorizzazioni nella propria politica basata sull'identità oppure avere le autorizzazioni `lambda:InvokeFunctionUrl` concesse nella politica basata sulle risorse della funzione. Account AWS In altre parole, una politica basata sulle risorse è facoltativa se l'utente dispone già delle autorizzazioni nella propria politica basata sull'identità. `lambda:InvokeFunctionUrl` `lambda:InvokeFunction` La valutazione delle politiche segue le regole delineate nella Logica di valutazione delle politiche.

Se il principale che effettua la richiesta si trova in un account diverso, deve disporre sia di una politica basata sull'identità che gli fornisca le autorizzazioni `lambda:InvokeFunctionUrl` e le `lambda:InvokeFunction` autorizzazioni concesse in una politica basata sulle risorse sulla funzione che sta tentando di richiamare. La valutazione delle politiche segue le regole descritte in Determinare se è consentita una richiesta tra più account.

La seguente politica basata sulle risorse consente al `example` ruolo in di richiamare l'URL della funzione associata Account AWS `444455556666` alla funzione. `my-function` La chiave lambda: InvokedViaFunctionUrl context limita l'azione alle chiamate URL di funzione. `lambda:InvokeFunction` Ciò significa che il principale deve utilizzare l'URL della funzione per richiamare la funzione. Se non lo includi`lambda:InvokedViaFunctionUrl`, il principale può richiamare la tua funzione tramite altri metodi di invocazione, oltre all'URL della funzione.

Example — Politica basata sulle risorse per più account

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"
        }
      }
    }
  ]
}
```

È possibile creare questa politica basata sulle risorse tramite la console utilizzando i seguenti passaggi:

Per concedere le autorizzazioni di richiamo URL a un altro account (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione per la quale desideri concedere autorizzazioni di richiamo URL.
3. Quindi, seleziona la scheda Configuration (Configurazione) e poi Permissions (Autorizzazioni).
4. In Resource-based policy (Policy basata sulle risorse), scegli Add permissions (Aggiungi autorizzazioni).
5. Scegli Function URL (URL funzione).
6. Per il tipo di autenticazione, scegli. AWS_IAM
7. Inserisci un ID della dichiarazione per la tua dichiarazione politica.
8. Per Principal, inserisci l'ID dell'account o l'Amazon Resource Name (ARN) dell'utente o del ruolo a cui desideri concedere le autorizzazioni. Ad esempio: **444455556666**.
9. Scegli Save (Salva).

In alternativa, puoi creare questa policy utilizzando i seguenti comandi add-permission AWS Command Line Interface ()AWS CLI. Quando si utilizza il AWS CLI, è necessario aggiungere le `lambda:InvokeFunction` istruzioni `lambda:InvokeFunctionUrl` and separatamente. Esempio:

```
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
```

Utilizzo del tipo di autenticazione **NONE**

> **Important**
>
> Se il tipo di autenticazione dell'URL della funzione è `NONE` e si dispone di una politica basata sulle risorse che garantisce l'accesso pubblico, qualsiasi utente non autenticato con l'URL della funzione può richiamare la funzione.

In alcuni casi, potresti volere che l'URL della tua funzione sia pubblico. Ad esempio, potrebbe essere preferibile inviare le richieste effettuate direttamente da un browser Web. Per consentire l'accesso pubblico all'URL della funzione, scegli il tipo di autenticazione `NONE`.

Se scegli il tipo di autenticazione `NONE`, Lambda non utilizza IAM per autenticare le richieste all'URL della funzione. Tuttavia, la funzione deve avere una politica basata sulle risorse che consenta e. `lambda:InvokeFunctionUrl` `lambda:InvokeFunction` Quando crei un URL di funzione con tipo di autenticazione `NONE` utilizzando la console o AWS Serverless Application Model (AWS SAM), Lambda crea automaticamente la politica basata sulle risorse per te. Se utilizzi direttamente l'API Lambda AWS CLI AWS CloudFormation, o l'API Lambda, devi aggiungere tu stesso la policy.

Ti consigliamo di includere la chiave lambda: InvokedViaFunctionUrl context nelle politiche basate sulle risorse quando usi il tipo di autenticazione. `NONE` Questa chiave di contesto garantisce che la funzione possa essere richiamata solo tramite l'URL della funzione e non tramite altri metodi di invocazione.

Tieni presente quanto segue su questa politica:

- Tutte le entità possono chiamare `lambda:InvokeFunctionUrl` `elambda:InvokeFunction`. Ciò significa che chiunque disponga dell'URL della tua funzione può richiamarla.
- Il valore della chiave della `lambda:FunctionUrlAuthType` condizione è`NONE`. Ciò significa che l'informativa consente l'accesso solo quando lo è anche `NONE` il tipo di autenticazione dell'URL della funzione.

- La `lambda:InvokedViaFunctionUrl` condizione garantisce che la funzione possa essere richiamata solo tramite l'URL della funzione e non tramite altri metodi di invocazione.

Example — Politica predefinita basata sulle risorse per il tipo di autenticazione 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"
        }
      }
    }
  ]
}
```

Creare la politica basata sulle risorse utilizzando il AWS CLI

A meno che non utilizzi la console o AWS SAM crei un URL di funzione con tipo di autenticazione`NONE`, devi aggiungere tu stesso la politica basata sulle risorse. Utilizzate i seguenti comandi per creare istruzioni per le autorizzazioni e. `lambda:InvokeFunctionUrl` `lambda:InvokeFunction` Ogni istruzione deve essere aggiunta in un comando separato.

```
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**
>
> Se elimini l'URL della funzione con il tipo di autenticazione `NONE`, Lambda non elimina automaticamente la policy basata sulle risorse associata. Se desideri eliminare questa policy, dovrai farlo manualmente.

Se la politica basata sulle risorse di una funzione non concede `lambda:invokeFunctionUrl` `lambda:InvokeFunction` autorizzazioni, gli utenti riceveranno un codice di errore 403 Forbidden quando cercheranno di richiamare l'URL della funzione. Ciò si verificherà anche se l'URL della funzione utilizza il tipo di autenticazione. `NONE`

Governance e controllo degli accessi

Oltre ai permessi di invocazione dell'URL della funzione, puoi anche controllare l'accesso alle azioni utilizzate per configurare la funzione. URLs Lambda supporta le seguenti azioni politiche IAM per la funzione: URLs

- `lambda:InvokeFunctionUrl`: richiamo di una funzione Lambda utilizzando l'URL della funzione.

- `lambda:CreateFunctionUrlConfig`: creazione di un URL della funzione e impostazione del relativo `AuthType`.
- `lambda:UpdateFunctionUrlConfig`: aggiornamento della configurazione dell'URL della funzione e del relativo `AuthType`.
- `lambda:GetFunctionUrlConfig`: visualizzazione dei dettagli dell'URL della funzione.
- `lambda:ListFunctionUrlConfigs`: elenco delle configurazioni dell'URL della funzione.
- `lambda:DeleteFunctionUrlConfig`: eliminazione dell'URL della funzione.

Per consentire o negare l'accesso all'URL della funzione ad altre AWS entità, includi queste azioni nelle policy IAM. Ad esempio, la seguente politica concede il `example` ruolo nelle Account AWS `444455556666` autorizzazioni per aggiornare l'URL della funzione per la funzione nell'account. **`my-function`** `123456789012`

Example Policy dell'URL della funzione tra account

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"
        }
    ]
}
```

Chiavi di condizione

Per un controllo granulare degli accessi alla funzione URLs, utilizzate i tasti contestuali delle condizioni. Lambda supporta i seguenti tasti contestuali per la funzione: URLs

- `lambda:FunctionUrlAuthType`— Definisce un valore enum che descrive il tipo di autenticazione utilizzato dall'URL della funzione. Il valore può essere `AWS_IAM` o `NONE`.
- `lambda:InvokedViaFunctionUrl`— Limita l'`lambda:InvokeFunction`azione alle chiamate effettuate tramite l'URL della funzione. Ciò garantisce che la funzione possa essere richiamata solo utilizzando l'URL della funzione e non tramite altri metodi di invocazione. Per esempi di politiche basate sulle risorse che utilizzano la chiave di `lambda:InvokedViaFunctionUrl` contesto, consulta gli esempi in and. Utilizzo del tipo di autenticazione `AWS_IAM` Utilizzo del tipo di autenticazione `NONE`

È possibile utilizzare queste chiavi di contesto nelle politiche associate alla propria funzione. Ad esempio, potresti voler limitare chi può apportare modifiche alla configurazione della tua funzione URLs. Per negare tutte le richieste `UpdateFunctionUrlConfig` a qualsiasi funzione con tipo di autenticazione URL `NONE`, è possibile definire la seguente policy:

Example Policy dell'URL della funzione con rifiuto esplicito

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"
                }
            }
        }
    ]
}
```

Per concedere il `example` ruolo nelle Account AWS `444455556666` autorizzazioni `CreateFunctionUrlConfig` e nelle `UpdateFunctionUrlConfig` richieste sulle funzioni con tipo di autenticazione URL`AWS_IAM`, puoi definire la seguente politica:

Example Policy dell'URL della funzione con permesso esplicito

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"
                }
            }
        }
    ]
}
```

È inoltre possibile utilizzare questa chiave di condizione in una policy di controllo dei servizi (SCP). Utilizzalo SCPs per gestire le autorizzazioni in un'intera organizzazione in. AWS Organizations Ad esempio, per impedire agli utenti di creare o aggiornare funzioni URLs che utilizzano qualcosa di diverso dal tipo di `AWS_IAM` autenticazione, utilizza la seguente politica di controllo del servizio:

Example SCP dell'URL della funzione con rifiuto esplicito

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"
                }
            }
        }
    ]
}
```

Richiamo di URL di funzioni Lambda

Un URL della funzione è un endpoint HTTP(S) dedicato alla funzione Lambda. È possibile creare e configurare un URL della funzione tramite la console Lambda o l'API Lambda.

> **Tip**
>
> Lambda offre due modi per richiamare una funzione tramite un endpoint HTTP: URL di funzione e Amazon API Gateway. Se non sei certo del metodo più adatto al tuo caso d'uso, consultarethe section called “Funzione URLs e Amazon API Gateway”.

Quando si crea un URL della funzione, Lambda genera automaticamente un endpoint URL univoco. Dopo aver creato un URL della funzione, il suo endpoint URL non cambia mai. Gli endpoint URL della funzione hanno il formato seguente:

```
https://<url-id>.lambda-url.<region>.on.aws
```

> **Note**
>
> Gli URL delle funzioni non sono supportati nei seguenti paesiRegioni AWS: Asia Pacifico (Hyderabad) (`ap-south-2`), Asia Pacifico (Melbourne) (`ap-southeast-4`), Asia Pacifico (Malesia) (), Asia Pacifico (Nuova Zelandaap-southeast-5) (), Asia Pacifico (Tailandia) ()ap-southeast-6, Asia Pacifico (Taipeiap-southeast-7) (), Canada occidentale (Calgary) ()ap-east-2, Europa (Spagna) ()ca-west-1, Europa (Zurigo) () eu-south-2eu-central-2, Israele (Tel Aviv) () e Medio Oriente (Emirati Arabi Uniti) (). il-central-1 me-central-1

Gli URL della funzione sono abilitati alla rete dual stack e supportano IPv4 e IPv6. Dopo aver configurato l'URL della funzione, è possibile richiamare la funzione attraverso il relativo endpoint HTTP(S) tramite un browser Web, curl, Postman o un client HTTP. Per richiamare un URL della funzione, è necessario disporre di autorizzazioni . Per ulteriori informazioni, consulta Controllo accessi.

Argomenti

- Nozioni di base sul richiamo di URL di funzioni

- Payload di richieste e risposte

Nozioni di base sul richiamo di URL di funzioni

Se l'URL della funzione utilizza il tipo di autenticazione `AWS_IAM`, è necessario firmare ogni richiesta HTTP utilizzando AWS Signature Version 4 (SigV4). Strumenti come awscurl, Postman e SigV4 Proxy offrono modalità integrate per firmare le richieste con Sigv4.

Se non utilizzi uno strumento per firmare le richieste HTTP all'URL della funzione, è necessario firmare manualmente ogni richiesta utilizzando Sigv4. Quando l'URL della funzione riceve una richiesta, Lambda calcola anche la firma Sigv4. Lambda elabora la richiesta solo se le firme corrispondono. Per le istruzioni su come firmare manualmente le richieste con SigV4, consulta Firma delle richieste AWS con Signature Version 4 nella Guida di riferimento generale di Riferimenti generali di Amazon Web Services.

Se l'URL della funzione utilizza il tipo di autenticazione `NONE`, non è necessario firmare le richieste utilizzando Sigv4. Puoi richiamare la funzione utilizzando un browser Web, curl, Postman o un client HTTP.

Per verificare semplici richieste `GET` alla funzione, usa un browser Web. Ad esempio, se l'URL della funzione è `https://abcdefg.lambda-url.us-east-1.on.aws` e richiede un parametro stringa `message`, l'URL della richiesta potrebbe essere simile al seguente:

```
https://abcdefg.lambda-url.us-east-1.on.aws/?message=HelloWorld
```

Per verificare altre richieste HTTP, ad esempio una richiesta `POST`, puoi utilizzare uno strumento come curl. Ad esempio, se desideri includere alcuni dati JSON in una richiesta `POST` all'URL della funzione, potresti utilizzare il comando curl seguente:

```
curl -v 'https://abcdefg.lambda-url.us-east-1.on.aws/?message=HelloWorld' \
-H 'content-type: application/json' \
-d '{ "example": "test" }'
```

Payload di richieste e risposte

Quando un client chiama l'URL della funzione, Lambda mappa la richiesta a un oggetto evento prima di passarlo alla funzione. La risposta della funzione viene quindi mappata a una risposta HTTP che Lambda invia al client tramite l'URL della funzione.

I formati degli eventi di richiesta e risposta seguono lo stesso schema del tipo di formato del payload di Gateway Amazon API versione 2.0.

Formato del payload di richiesta

Un payload di richiesta ha la seguente struttura:

```
{
  "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
}
```

Parametro	Descrizione	Esempio
`version`	Il tipo di formato del payload per questo evento. Gli URL della funzione Lambda attualmente supportano il tipo di formato del payload versione 2.0.	`2.0`
`routeKey`	Gli URL della funzione non utilizzano questo parametro . Lambda lo imposta su `$default` come segnaposto.	`$default`
`rawPath`	Percorso della richiesta. Ad esempio, se l'URL della richiesta è `https://{url-id}.lambda-url.{region}.on.aws/example/test/demo` , il valore raw del percorso è `/example/test/demo` .	`/example/test/demo`

Parametro	Descrizione	Esempio
`rawQueryString`	La stringa raw contenente i parametri della stringa di query della richiesta. I caratteri supportati includono `a-z`, `A-Z`, `0-9`, `.`, `_`, `-`, `%`, `&`, `=` e `+`.	`"?parameter1=value1&parameter2=value2"`
`cookies`	Un array contenente tutti i cookie inviati come parte della richiesta.	`["Cookie_1=Value_1", "Cookie_2=Value_2"]`
`headers`	L'elenco delle intestazioni della richiesta, presentate come coppie chiave-valore.	`{"header1": "value1", "header2": "value2"}`
`queryStringParameters`	I parametri di query per la richiesta. Ad esempio, se l'URL della richiesta è `https://{url-id}.lambda-url.{region}.on.aws/example?name=Jane`, il valore `queryStringParameters` è un oggetto JSON con una chiave di `name` e un valore di `Jane`.	`{"name": "Jane"}`
`requestContext`	Un oggetto contenente informazioni aggiuntive sulla richiesta, ad esempio `requestId`, l'ora della richiesta e l'identità del chiamante se autorizzato tramite AWS Identity and Access Management (IAM).	

Parametro	Descrizione	Esempio
`requestContext.accountId`	L'ID Account AWS del proprietario della funzione.	`"123456789012"`
`requestContext.apiId`	L'ID dell'URL della funzione.	`"33anwqw8fj"`
`requestContext.authentication`	Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su `null`.	`null`
`requestContext.authorizer`	Un oggetto contenente informazioni sull'identità del chiamante, se l'URL della funzione utilizza il tipo di autenticazione `AWS_IAM`. Altrimenti, Lambda lo imposta su `null`.	
`requestContext.authorizer.iam.accessKey`	La chiave di accesso dell'identità del chiamante.	`"AKIAIOSFODNN7EXAMPLE"`
`requestContext.authorizer.iam.accountId`	L'ID Account AWS dell'identità del chiamante.	`"111122223333"`
`requestContext.authorizer.iam.callerId`	L'ID (ID utente) del chiamante.	`"AIDACKCEVSQ6C2EXAMPLE"`
`requestContext.authorizer.iam.cognitoIdentity`	Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su `null` o lo esclude dal JSON.	`null`

Parametro	Descrizione	Esempio
`requestContext.authorizer.iam.principalOrgId`	L'ID dell'organizzazione principale associato all'identità del chiamante.	`"AIDACKCEVSQORGEXAMPLE"`
`requestContext.authorizer.iam.userArn`	L'Amazon Resource Name (ARN) utente dell'identità del chiamante.	`"arn:aws:iam::111122223333:user/example-user"`
`requestContext.authorizer.iam.userId`	L'ID utente dell'identità del chiamante.	`"AIDACOSFODNN7EXAMPLE2"`
`requestContext.domainName`	Il nome di dominio dell'URL della funzione.	`"<url-id>.lambda-url.us-west-2.on.aws"`
`requestContext.domainPrefix`	Il prefisso di dominio dell'URL della funzione.	`"<url-id>"`
`requestContext.http`	Un oggetto contenente i dettagli sulla richiesta HTTP.	
`requestContext.http.method`	Il metodo HTTP utilizzato nella richiesta. I valori validi includono `GET`, `POST`, `PUT`, `HEAD`, `OPTIONS`, `PATCH` e `DELETE`.	`GET`
`requestContext.http.path`	Percorso della richiesta. Ad esempio, se l'URL della richiesta è `https://{url-id}.lambda-url.{region}.on.aws/example/test/demo`, il valore del percorso è `/example/test/demo`.	`/example/test/demo`

Parametro	Descrizione	Esempio
`requestContext.http.protocol`	Il protocollo della richiesta.	`HTTP/1.1`
`requestContext.http.sourceIp`	L'indirizzo IP di origine della connessione TCP immediata da cui proviene la richiesta.	`123.123.123.123`
`requestContext.http.userAgent`	Il valore dell'intestazione della richiesta User-Agent.	`Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) Gecko/20100101 Firefox/42.0`
`requestContext.requestId`	L'l'ID della richiesta di richiamo. È possibile utilizzare questo ID per tenere traccia dei registri dei richiami correlati alla funzione.	`e1506fd5-9e7b-434f-bd42-4f8fa224b599`
`requestContext.routeKey`	Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su `$default` come segnaposto.	`$default`
`requestContext.stage`	Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su `$default` come segnaposto.	`$default`
`requestContext.time`	Il timestamp della richiesta.	`"07/Sep/2021:22:50:22 +0000"`
`requestContext.timeEpoch`	Il timestamp della richiesta, in formato temporale Unix epoch.	`"1631055022677"`

Parametro	Descrizione	Esempio
body	Il corpo della richiesta. Se il tipo di contenuto della richiesta è binario, il corpo è con codifica base64.	`{"key1": "value1", "key2": "value2"}`
`pathParameters`	Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su `null` o lo esclude dal JSON.	`null`
`isBase64Encoded`	`TRUE` se il corpo è un payload binario e con codifica base64. `FALSE` in caso contrario.	`FALSE`
`stageVariables`	Gli URL della funzione non utilizzano questo parametro. Lambda lo imposta su `null` o lo esclude dal JSON.	`null`

Formato del payload di risposta

Quando la funzione restituisce una risposta, Lambda analizza la risposta e la converte in una risposta HTTP. I payload di risposta della funzione hanno il formato seguente:

```
{
   "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 deduce il formato di risposta per l'utente. Se la funzione Lambda restituisce un JSON valido e non restituisce un `statusCode`, Lambda presuppone quanto segue:

- `statusCode` is `200`.

> **Note**
>
> I valori validi `statusCode` sono compresi tra 100 e 599.

- `content-type` is `application/json`.
- `body` è la risposta della funzione.
- `isBase64Encoded` is `false`.

Gli esempi seguenti mostrano come l'output della funzione Lambda viene mappato al payload della risposta e come il payload della risposta viene mappato alla risposta HTTP finale. Quando il client richiama l'URL della funzione, viene visualizzata la risposta HTTP.

Esempio di output per una risposta stringa

Risultato della funzione Lambda	Output di risposta interpretata	Risposta HTTP (cosa vede il client)
`"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!"`

Esempio di output per una risposta JSON

Risultato della funzione Lambda	Output di risposta interpretata	Risposta HTTP (cosa vede il client)

{
  "message": "Hello,
 world!"
}
	{
	  "statusCode": 200,
	  "body": {
	  "message": "Hello,
	 world!"
	  },
	  "headers": {
	  "content-type":
	 "application/json"
	  },
	  "isBase64Encoded":
	 false
	}
		HTTP/2 200
		date: Wed, 08 Sep 2021
		 18:02:24 GMT
		content-type: applicati
		on/json
		content-length: 34
		
		{
		  "message": "Hello,
		 world!"
		}

Esempio di output per una risposta personalizzata

Risultato della funzione Lambda	Output di risposta interpretata	Risposta HTTP (cosa vede il client)

{
  "statusCode": 201,
  "headers": {
  "Content-Type":
 "application/json",
  "My-Custom-
Header": "Custom Value"
  },
  "body": JSON.stri
ngify({
  "message":
 "Hello, world!"
  }),
  "isBase64Encoded":
 false
}
	{
	  "statusCode": 201,
	  "headers": {
	  "Content-Type":
	 "application/json",
	  "My-Custom-
	Header": "Custom Value"
	  },
	  "body": JSON.stri
	ngify({
	  "message":
	 "Hello, world!"
	  }),
	  "isBase64Encoded":
	 false
	}
		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!"
		}

Cookie

Per restituire i cookie della funzione, non aggiungere manualmente intestazioni `set-cookie`. Al contrario, includi i cookie nell'oggetto payload di risposta. Lambda li interpreta automaticamente e li aggiunge come intestazioni `set-cookie` nella risposta HTTP, come nell'esempio seguente.

Risultato della funzione Lambda	Risposta HTTP (cosa vede il client)

```
{
   "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!"
}
```

Monitoraggio degli URL della funzione Lambda

Puoi utilizzare AWS CloudTrail e Amazon CloudWatch per monitorare gli URL della funzione.

Argomenti

- Monitoraggio degli URL della funzione con CloudTrail
- Parametri CloudWatch per gli URL della funzione

Monitoraggio degli URL della funzione con CloudTrail

Per gli URL della funzione, Lambda supporta automaticamente la registrazione delle seguenti operazioni API come eventi nei file di log di CloudTrail:

- CreateFunctionUrlConfig
- UpdateFunctionUrlConfig
- DeleteFunctionUrlConfig
- GetFunctionUrlConfig
- ListFunctionUrlConfigs

Ogni voce di registro contiene informazioni sull'identità del chiamante, sul momento in cui è stata effettuata la richiesta e altri dettagli. Puoi vedere tutti gli eventi degli ultimi 90 giorni visualizzando la cronologia eventi di CloudTrail. Per conservare i record degli ultimi 90 giorni, puoi creare un percorso.

Per impostazione predefinita, CloudTrail non registra le richieste `InvokeFunctionUrl`, che sono considerate eventi di dati. È comunque possibile attivare la registrazione degli eventi di dati in CloudTrail. Per ulteriori informazioni, consulta Registrazione di eventi di dati per i percorsi nella Guida per l'utente di AWS CloudTrail.

Parametri CloudWatch per gli URL della funzione

Lambda invia parametri aggregati relativi alle richieste degli URL della funzione a CloudWatch. Con questi parametri, puoi monitorare gli URL della funzione, creare dashboard e configurare allarmi nella console di CloudWatch.

Gli URL della funzione supportano i seguenti parametri di richiamo. È consigliabile visualizzare questi parametri con le statistiche di `Sum`.

- `UrlRequestCount`: il numero di richieste inviate all'URL della funzione.
- `Url4xxCount`: il numero di richieste che hanno restituito un codice di stato HTTP 4XX. I codici della serie 4XX indicano errori lato client, ad esempio richieste non valide.
- `Url5xxCount`: il numero di richieste che hanno restituito un codice di stato HTTP 5XX. I codici della serie 5XX indicano errori lato server, ad esempio errori di funzione e timeout.

Gli URL della funzione supportano anche il seguente parametro di prestazioni. È consigliabile visualizzare questo parametro con le statistiche di `Average` o `Max`.

- `UrlRequestLatency`: il periodo di tempo che intercorre tra il momento in cui l'URL della funzione riceve una richiesta e il momento in cui l'URL della funzione restituisce una risposta.

Ciascuno di questi parametri di richiamo e prestazioni supporta le seguenti dimensioni:

- `FunctionName`: visualizza i parametri aggregati per gli URL della funzione assegnati a una versione `$LATEST` non pubblicata di una funzione o a uno degli alias della funzione. Ad esempio, `hello-world-function`.
- `Resource`: visualizza i parametri relativi a un URL della funzione specifico. Questo URL è definito dal nome di una funzione, insieme alla versione `$LATEST` non pubblicata della funzione o a uno degli alias della funzione. Ad esempio, `hello-world-function:$LATEST`.
- `ExecutedVersion`: visualizza i parametri per un URL di funzione specifico, in base alla versione eseguita. Puoi utilizzare questa dimensione principalmente per tenere traccia dell'URL della funzione assegnato alla versione `$LATEST` non pubblicata.

Selezionare un metodo per richiamare la funzione Lambda tramite una richiesta HTTP

Numerosi casi d'uso comuni per Lambda implicano l'invocazione della funzione tramite una richiesta HTTP. Ad esempio, potrebbe essere preferibile che un'applicazione Web invochi la funzione tramite una richiesta del browser. Le funzioni Lambda possono essere utilizzate anche per creare REST completo APIs, gestire le interazioni degli utenti da app mobili, elaborare dati da servizi esterni tramite chiamate HTTP o creare webhook personalizzati.

Le sezioni seguenti spiegano quali sono le tue scelte per richiamare Lambda tramite HTTP e forniscono informazioni per aiutarti a prendere la decisione giusta per il tuo caso d'uso particolare.

Quali sono le tue scelte quando selezioni un metodo di invocazione HTTP?

Lambda offre due metodi principali per richiamare una funzione utilizzando una richiesta HTTP: funzione e API URLs Gateway. Le differenze tra queste due opzioni sono le seguenti:

- La funzione Lambda URLs fornisce un endpoint HTTP semplice e diretto per una funzione Lambda. Sono ottimizzati per semplicità ed economicità e forniscono il percorso più veloce per esporre una funzione Lambda tramite HTTP.
- API Gateway è un servizio più avanzato per la creazione di funzionalità complete APIs. API Gateway è ottimizzato per la creazione e la gestione di produzioni APIs su larga scala e fornisce strumenti completi per la sicurezza, il monitoraggio e la gestione del traffico.

Suggerimenti se conosci già le tue esigenze

Se hai già le idee chiare sulle tue esigenze, ecco i nostri suggerimenti di base:

Consigliamo la funzionalità URLs per applicazioni semplici o per la prototipazione in cui sono necessari solo metodi di autenticazione e request/response gestione di base e dove si desidera ridurre al minimo costi e complessità.

API Gateway è la scelta migliore per le applicazioni di produzione su larga scala o per i casi in cui sono necessarie funzionalità più avanzate come il supporto OpenAPI Description, una scelta di opzioni di autenticazione, nomi di dominio personalizzati o una request/response gestione avanzata che include throttling, caching e trasformazione. request/response

Cosa considerare quando si seleziona un metodo per richiamare la funzione Lambda

Nella scelta tra funzione URLs e API Gateway, è necessario considerare i seguenti fattori:

- Le tue esigenze di autenticazione, ad esempio se richiedi OAuth o Amazon Cognito per autenticare gli utenti
- I tuoi requisiti di scalabilità e la complessità dell'API che desideri implementare
- Se hai bisogno di funzionalità avanzate come la convalida e la formattazione delle richieste request/response
- I tuoi requisiti di monitoraggio
- I tuoi obiettivi di costo

Comprendendo questi fattori, è possibile selezionare l'opzione che meglio bilancia i requisiti di sicurezza, complessità e costi.

Le informazioni seguente riepilogano le differenze principali tra le due opzioni.

Autenticazione

- La funzione URLs fornisce opzioni di autenticazione di base tramite AWS Identity and Access Management (IAM). Puoi configurare gli endpoint in modo che siano pubblici (senza autenticazione) o che richiedano l'autenticazione IAM. Con l'autenticazione IAM, puoi utilizzare AWS credenziali standard o ruoli IAM per controllare l'accesso. Sebbene sia semplice da configurare, questo approccio offre opzioni limitate rispetto ad altri metodi di autenticazione.
- API Gateway fornisce l'accesso a una gamma più completa di opzioni di autenticazione. Oltre all'autenticazione IAM, puoi utilizzare gli autorizzatori Lambda (logica di autenticazione personalizzata), i pool di utenti di Amazon Cognito e i flussi .0. OAuth2 Questa flessibilità consente di implementare schemi di autenticazione complessi, inclusi provider di autenticazione di terze parti, autenticazione basata su token e autenticazione a più fattori.

Gestione di richieste/risposte

- La funzione URLs fornisce la gestione di base delle richieste e delle risposte HTTP. Supportano i metodi HTTP standard e includono il supporto integrato per CORS (cross-origin resource sharing). Sebbene siano in grado di gestire i payload JSON e i parametri di query in modo naturale, non offrono funzionalità di trasformazione o convalida delle richieste. La gestione delle risposte è altrettanto semplice: il client riceve la risposta dalla funzione Lambda esattamente come la restituisce Lambda.
- API Gateway offre sofisticate funzionalità di gestione delle richieste e delle risposte. È possibile definire validatori di richiesta, trasformare richieste e risposte utilizzando modelli di mappatura, impostare request/response intestazioni e implementare la memorizzazione nella cache delle risposte. API Gateway supporta anche payload binari e nomi di dominio personalizzati e può modificare le risposte prima che raggiungano il client. È possibile configurare modelli per la request/response convalida e la trasformazione utilizzando JSON Schema.

Dimensionamento

- Le funzioni si URLs adattano direttamente ai limiti di concorrenza della funzione Lambda e gestisci i picchi di traffico scalando la funzione fino al limite di concorrenza massimo configurato. Una

volta raggiunto tale limite, Lambda risponde alle richieste aggiuntive con risposte HTTP 429. Non esiste un meccanismo di accodamento integrato, pertanto la gestione della scalabilità dipende interamente dalla configurazione della funzione Lambda. Per impostazione predefinita, le funzioni Lambda hanno un limite di 1.000 esecuzioni simultanee per. Regione AWS

- API Gateway offre funzionalità di scalabilità aggiuntive oltre alla scalabilità propria di Lambda. Include controlli integrati per l'accodamento e la limitazione delle richieste che consentono di gestire i picchi di traffico con maggiore efficienza. Per impostazione predefinita, API Gateway è in grado di gestire fino a 10.000 richieste al secondo per regione, con una capacità di espansione di 5.000 richieste al secondo. Fornisce inoltre strumenti per limitare le richieste a diversi livelli (API, fase o metodo) per proteggere il backend.

Monitoraggio

- La funzione URLs offre un monitoraggio di base tramite i CloudWatch parametri di Amazon, tra cui il conteggio delle richieste, la latenza e i tassi di errore. Puoi accedere a parametri e log Lambda standard, che mostrano le richieste non elaborate che arrivano alla tua funzione. Sebbene ciò fornisca una visibilità operativa essenziale, i parametri si concentrano principalmente sull'esecuzione delle funzioni.
- API Gateway offre funzionalità di monitoraggio complete, tra cui parametri dettagliati, opzioni di registrazione e tracciamento. Puoi monitorare le chiamate API, la latenza, i tassi di errore e i tassi di cache hit/miss tramite. CloudWatch API Gateway si integra anche con AWS X-Ray il tracciamento distribuito e fornisce formati di registrazione personalizzabili.

Costo

- Le funzioni URLs seguono il modello di prezzo standard Lambda: paghi solo per le chiamate di funzione e il tempo di calcolo. Non sono previsti costi aggiuntivi per l'endpoint URL stesso. Ciò lo rende una scelta conveniente per applicazioni semplici APIs o a basso traffico se non sono necessarie le funzionalità aggiuntive di API Gateway.
- API Gateway offre un piano gratuito che include un milione di chiamate API ricevute per REST APIs e un milione di chiamate API ricevute per HTTP APIs. Successivamente, API Gateway addebita i costi per le chiamate API, il trasferimento dei dati e la memorizzazione nella cache (se abilitata). Consulta la pagina dei prezzi di API Gateway per conoscere i costi relativi al tuo caso d'uso.

Altre funzionalità

- URLsLe funzioni sono progettate per la semplicità e l'integrazione diretta con Lambda. Supportano endpoint HTTP e HTTPS, offrono supporto CORS integrato e forniscono endpoint dual-stack (e). IPv4 IPv6 Sebbene non dispongano di funzionalità avanzate, eccellono negli scenari in cui è necessario un modo rapido e diretto per esporre le funzioni Lambda tramite HTTP.
- API Gateway include numerose funzionalità aggiuntive come il controllo delle versioni delle API, la gestione delle fasi, le chiavi API per i piani di utilizzo, la documentazione delle API tramite Swagger/OpenAPI, l'accesso WebSocket APIs privato all'interno di APIs un VPC e l'integrazione WAF per una maggiore sicurezza. Supporta anche implementazioni Canary, integrazioni fittizie per i test e l'integrazione con altri sistemi oltre a Lambda. Servizi AWS

Selezionare un metodo per richiamare la funzione Lambda

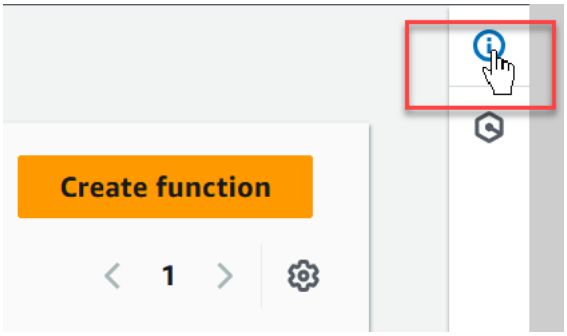
Ora che hai letto i criteri per la selezione tra la funzione Lambda URLs e l'API Gateway e le principali differenze tra di essi, puoi selezionare l'opzione più adatta alle tue esigenze e utilizzare le seguenti risorse per iniziare a utilizzarla.

Function URLs

Inizia a usare la funzione URLs con le seguenti risorse

- Segui il tutorial Creazione di una funzione Lambda con la funzione URL
- Scopri di più sulla funzione URLs nel the section called “Funzione URLs” capitolo di questa guida
- Prova il tutorial guidato dalla console Creare una semplice app Web effettuando le seguenti operazioni:

1. Apri la pagina Funzioni della console Lambda.
2. Apri il pannello di aiuto scegliendo l'icona nell'angolo in alto a destra della schermata.



3. Seleziona Tutorial.
4. In Crea una semplice app Web, scegli Avvia tutorial.

API Gateway

Iniziare a usare Lambda e API Gateway con le seguenti risorse

- Segui il tutorial Utilizzo di Lambda con API Gateway per creare una REST API integrata con una funzione Lambda di backend.
- Scopri di più sui diversi tipi di API offerti da API Gateway nelle seguenti sezioni della Guida per gli sviluppatori di Gateway Amazon API:
  - API Gateway REST APIs
  - Gateway API HTTP APIs
  - API Gateway WebSocket APIs
- Prova uno o più esempi nella sezione Tutorial e workshop della Guida per gli sviluppatori di Gateway Amazon API.

Tutorial: creazione di un endpoint webhook utilizzando l'URL di una funzione Lambda

In questo tutorial, crei l'URL di una funzione Lambda per implementare un endpoint webhook. Un webhook è una comunicazione leggera basata sugli eventi che invia automaticamente i dati tra le applicazioni tramite HTTP. È possibile utilizzare un webhook per ricevere aggiornamenti immediati sugli eventi che si verificano in un altro sistema, ad esempio quando un nuovo cliente si registra su un sito Web, viene elaborato un pagamento o viene caricato un file.

Con Lambda, i webhook possono essere implementati utilizzando la funzione Lambda o l'API Gateway. URLs URLs Le funzioni sono un'ottima scelta per webhook semplici che non richiedono funzionalità come l'autorizzazione avanzata o la convalida delle richieste.

> **Tip**
>
> Se non sei sicuro di quale sia la soluzione migliore per il tuo caso d'uso particolare, consulta. the section called “Funzione URLs e Amazon API Gateway”

Prerequisiti

Per completare questo tutorial, devi avere Python (versione 3.8 o successiva) o Node.js (versione 18 o successiva) installato sul tuo computer locale.

Per testare l'endpoint utilizzando una richiesta HTTP, il tutorial utilizza curl, uno strumento a riga di comando che è possibile utilizzare per trasferire dati utilizzando vari protocolli di rete. Fai riferimento alla documentazione di curl per scoprire come installare lo strumento se non lo possiedi già.

Creazione della funzione Lambda

Per prima cosa crea la funzione Lambda che viene eseguita quando una richiesta HTTP viene inviata al tuo endpoint webhook. In questo esempio, l'applicazione di invio invia un aggiornamento ogni volta che viene inviato un pagamento e indica nel corpo della richiesta HTTP se il pagamento è andato a buon fine. La funzione Lambda analizza la richiesta e interviene in base allo stato del pagamento. In questo esempio, il codice stampa solo l'ID dell'ordine per il pagamento, ma in un'applicazione reale è possibile aggiungere l'ordine a un database o inviare una notifica.

La funzione implementa anche il metodo di autenticazione più comune utilizzato per i webhook, l'autenticazione dei messaggi basata su hash (HMAC). Con questo metodo, sia l'applicazione di

invio che quella di ricezione condividono una chiave segreta. L'applicazione di invio utilizza un algoritmo di hashing per generare una firma univoca utilizzando questa chiave insieme al contenuto del messaggio e include la firma nella richiesta del webhook come intestazione HTTP. L'applicazione ricevente ripete quindi questo passaggio, generando la firma utilizzando la chiave segreta e confronta il valore risultante con la firma inviata nell'intestazione della richiesta. Se il risultato corrisponde, la richiesta viene considerata legittima.

Crea la funzione utilizzando la console Lambda con il runtime Python o Node.js.

Python

Creazione della funzione Lambda

1. Aprire la pagina Funzioni della console Lambda.
2. Crea una funzione di base «Hello world» effettuando le seguenti operazioni:
   a. Scegli Crea funzione.
   b. Scegli Crea da zero.
   c. Nel campo Function name (Nome funzione), immettere **`myLambdaWebhook`**.
   d. Per Runtime, seleziona python3.14.
   e. Scegli Crea funzione.
3. Nel riquadro Codice sorgente, sostituisci il codice esistente copiando e incollando quanto segue:

```
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. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione.

Node.js

Creazione della funzione Lambda

1. Aprire la pagina Funzioni della console Lambda.

2. Crea una funzione di base «Hello world» effettuando le seguenti operazioni:

   a. Scegli Crea funzione.

   b. Scegli Crea da zero.

   c. Nel campo Function name (Nome funzione), immettere **`myLambdaWebhook`**.

   d. Per Runtime, seleziona nodejs24.x.

   e. Scegli Crea funzione.

3. Nel riquadro Codice sorgente, sostituisci il codice esistente copiando e incollando quanto segue:

```
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. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione.

Crea la chiave segreta

Per autenticare la richiesta webhook, la funzione Lambda utilizza una chiave segreta che condivide con l'applicazione chiamante. In questo esempio, la chiave viene memorizzata in una variabile di ambiente. In un'applicazione di produzione, non includere informazioni sensibili come le password nel codice funzione. Invece, crea un Gestione dei segreti AWS segreto e poi usa l'estensione Lambda AWS Parameters and Secrets per recuperare le credenziali nella funzione Lambda.

Crea e archivia la chiave segreta del webhook

1. Genera una stringa lunga e casuale utilizzando un generatore di numeri casuali crittograficamente sicuro. Puoi usare i seguenti frammenti di codice in Python o Node.js per generare e stampare un segreto di 32 caratteri o usare il tuo metodo preferito.

   Python

   Example codice per generare un segreto

   ```
   import secrets
   webhook_secret = secrets.token_urlsafe(32)
   print(webhook_secret)
   ```

   Node.js

   Example codice per generare un segreto (formato del modulo ES)

   ```
   import crypto from 'crypto';
   let webhookSecret = crypto.randomBytes(32).toString('base64');
   console.log(webhookSecret)
   ```

2. Memorizza la stringa generata come variabile di ambiente per la tua funzione effettuando le seguenti operazioni:

   a. Nella scheda Configurazione della funzione, seleziona Variabili di ambiente.
   b. Scegli Modifica.
   c. Scegli Add environment variable (Aggiungi variabile d'ambiente).
   d. Per Chiave**`WEBHOOK_SECRET`**, inserisci, quindi per Valore, inserisci il segreto generato nel passaggio precedente.
   e. Scegli Save (Salva).

Dovrai utilizzare nuovamente questo segreto più avanti nel tutorial per testare la tua funzione, quindi prendine nota ora.

Crea l'endpoint URL della funzione

Crea un endpoint per il tuo webhook utilizzando l'URL di una funzione Lambda. Poiché utilizzi il tipo di autenticazione di `NONE` per creare un endpoint con accesso pubblico, chiunque disponga dell'URL

può richiamare la tua funzione. Per ulteriori informazioni sul controllo dell'accesso alla funzione, consulta. URLs the section called “Controllo accessi” Se hai bisogno di opzioni di autenticazione più avanzate per il tuo webhook, prendi in considerazione l'utilizzo di API Gateway.

Crea l'endpoint URL della funzione

1. Nella scheda Configurazione della funzione, selezionate Function URL.
2. Scegli Crea URL della funzione.
3. Per il tipo di autenticazione, seleziona NESSUNA.
4. Scegli Save (Salva).

L'endpoint per l'URL della funzione appena creato viene visualizzato nel riquadro URL della funzione. Copia l'endpoint per utilizzarlo più avanti nel tutorial.

Prova la funzione nella console

Prima di utilizzare una richiesta HTTP per richiamare la funzione utilizzando l'endpoint URL, testala nella console per confermare che il codice funzioni come previsto.

Per verificare la funzione nella console, devi prima calcolare una firma webhook utilizzando il segreto generato in precedenza nel tutorial con il seguente payload JSON di test:

```
{
    "type": "payment.success",
    "orderId": "1234",
    "amount": "99.99"
}
```

Usa uno dei seguenti esempi di codice Python o Node.js per calcolare la firma del webhook usando il tuo segreto.

Python

Calcola la firma del webhook

1. Salva il codice seguente come file denominato`calculate_signature.py`. Sostituisci il webhook secret nel codice con il tuo valore.

```
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. Calcola la firma eseguendo il comando seguente dalla stessa directory in cui hai salvato il codice. Copia la firma generata dal codice.

```
python calculate_signature.py
```

Node.js

Calcola la firma del webhook

1. Salva il codice seguente come file denominato`calculate_signature.mjs`. Sostituisci il webhook secret nel codice con il tuo valore.

```
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. Calcola la firma eseguendo il comando seguente dalla stessa directory in cui hai salvato il codice. Copia la firma generata dal codice.

```
node calculate_signature.mjs
```

Ora puoi testare il codice della funzione utilizzando una richiesta HTTP di prova nella console.

Prova la funzione nella console

1. Seleziona la scheda Codice relativa alla tua funzione.
2. Nella sezione EVENTI DI TEST, scegli Crea nuovo evento di test
3. Per Event name (Nome evento), immettere **myEvent**.
4. Sostituisci il JSON esistente copiando e incollando quanto segue nel riquadro Event JSON. Sostituisci la firma del webhook con il valore calcolato nel passaggio precedente.

```
{
  "headers": {
    "Content-Type": "application/json",
    "x-webhook-signature":
 "2d672e7a0423fab740fbc040e801d1241f2df32d2ffd8989617a599486553e2a"
  },
  "body": "{\"type\": \"payment.success\", \"orderId\": \"1234\", \"amount\":
 \"99.99\"}"
}
```

5. Scegli Save (Salva).
6. Scegli Richiama .

   Verrà visualizzato un output simile al seguente:

   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
```

Prova la funzione utilizzando una richiesta HTTP

Usa lo strumento da riga di comando curl per testare il tuo endpoint webhook.

Prova la funzione usando le richieste HTTP

1. In un terminale o in un programma shell, esegui il seguente comando curl. Sostituisci l'URL con il valore dell'endpoint URL della tua funzione e sostituisci la firma del webhook con la firma calcolata utilizzando la tua chiave segreta.

```
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"}'
```

Verrà visualizzato l'output seguente:

```
{"received": true}
```

2. Esamina CloudWatch i log della tua funzione per confermare che abbia analizzato correttamente il payload effettuando le seguenti operazioni:

   a. Apri la pagina del gruppo Logs nella CloudWatch console Amazon.

   b. Seleziona il gruppo di log della tua funzione (`/aws/lambda/myLambdaWebhook`).

   c. Seleziona il flusso di log più recente.

   Dovresti vedere un output simile al seguente nei log della tua funzione:

   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. Verifica che il codice rilevi una firma non valida eseguendo il seguente comando curl. Sostituisci l'URL con il tuo endpoint URL della funzione.

```
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"}'
```

Verrà visualizzato l'output seguente:

```
{"error": "Invalid signature"}
```

Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili a tuo carico. Account AWS

Per eliminare la funzione Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **`confirm`** nel campo di immissione del testo, quindi scegli Elimina.

Quando hai creato la funzione Lambda nella console, Lambda ha anche creato un ruolo di esecuzione per la tua funzione.

Come eliminare il ruolo di esecuzione

1. Aprire la pagina Ruoli della console IAM.
2. Seleziona il ruolo di esecuzione creato da Lambda. Il ruolo ha il formato `myLambdaWebhook-role-<random string>` del nome.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Elimina.

Informazioni sulla scalabilità della funzione Lambda

La simultaneità è il numero di richieste in transito che la funzione AWS Lambda può gestire nello stesso momento. Per ogni richiesta simultanea, Lambda fornisce un'istanza separata del tuo ambiente di esecuzione. Man mano che le funzioni ricevono più richieste, Lambda gestisce automaticamente il dimensionamento del numero di ambienti di esecuzione fino al raggiungimento del limite di simultaneità dell'account. Per impostazione predefinita, Lambda fornisce all'account un limite totale di simultaneità pari a 1.000 esecuzioni simultanee per tutte le funzioni in una Regione AWS. Per soddisfare le esigenze specifiche dell'account, è possibile richiedere un aumento della quota e configurare i controlli di simultaneità a livello di funzione in modo che le funzioni critiche non subiscano limitazioni.

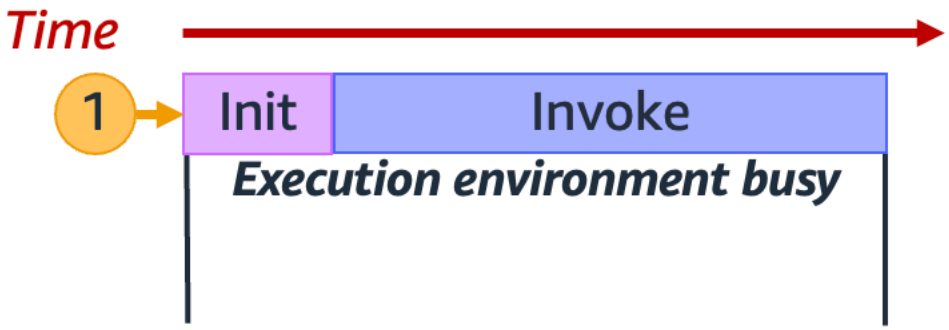
In questo argomento sono descritti i concetti di simultaneità e la scalabilità delle funzioni in Lambda. Alla fine di questo argomento, sarà possibile capire come calcolare la simultaneità, visualizzare le due principali opzioni di controllo della simultaneità (riservata e fornita), stimare le impostazioni di controllo della simultaneità appropriate e visualizzare i parametri per un'ulteriore ottimizzazione.

Sezioni

- Comprendere e visualizzare la simultaneità
- Calcolo della simultaneità per una funzione
- Informazioni sulla simultaneità riservata e simultaneità fornita
- Informazioni sulla simultaneità e le richieste al secondo
- Quote di simultaneità
- Configurazione della simultaneità riservata per una funzione
- Configurazione della simultaneità fornita per una funzione
- Comportamento del dimensionamento Lambda
- Monitoraggio della simultaneità

Comprendere e visualizzare la simultaneità

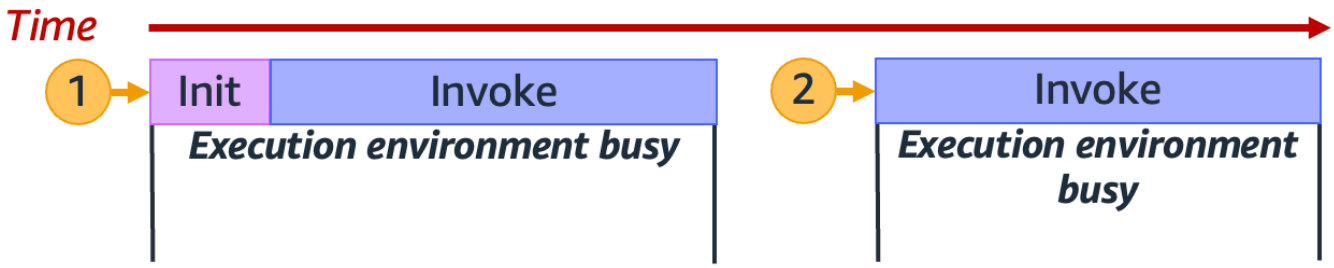
Lambda richiama la funzione in un ambiente di esecuzione sicuro e isolato. Per gestire una richiesta, Lambda deve prima inizializzare un ambiente di esecuzione (la fase Init) e poi utilizzare tale ambiente per richiamare la funzione (la fase Invoke):



> **Note**
>
> Le durate effettive delle fasi Init e Invoke possono variare in base a molti fattori, come il runtime scelto e il codice della funzione Lambda. Il diagramma precedente non intende rappresentare le proporzioni esatte delle durate delle fasi Init e Invoke.

Il rettangolo nel diagramma rappresenta un singolo ambiente di esecuzione. Quando la funzione riceve la sua primissima richiesta (rappresentata dal cerchio giallo con etichetta 1), Lambda crea un nuovo ambiente di esecuzione ed esegue il codice all'esterno del gestore principale durante la fase Init. Quindi, esegue il codice del gestore principale della funzione durante la fase Invoke. Durante l'intero processo, questo ambiente di esecuzione è occupato e non può elaborare altre richieste.

Quando Lambda termina l'elaborazione della prima richiesta, questo ambiente di esecuzione potrà elaborare richieste aggiuntive per la stessa funzione. Per le richieste successive, Lambda non deve reinizializzare l'ambiente.

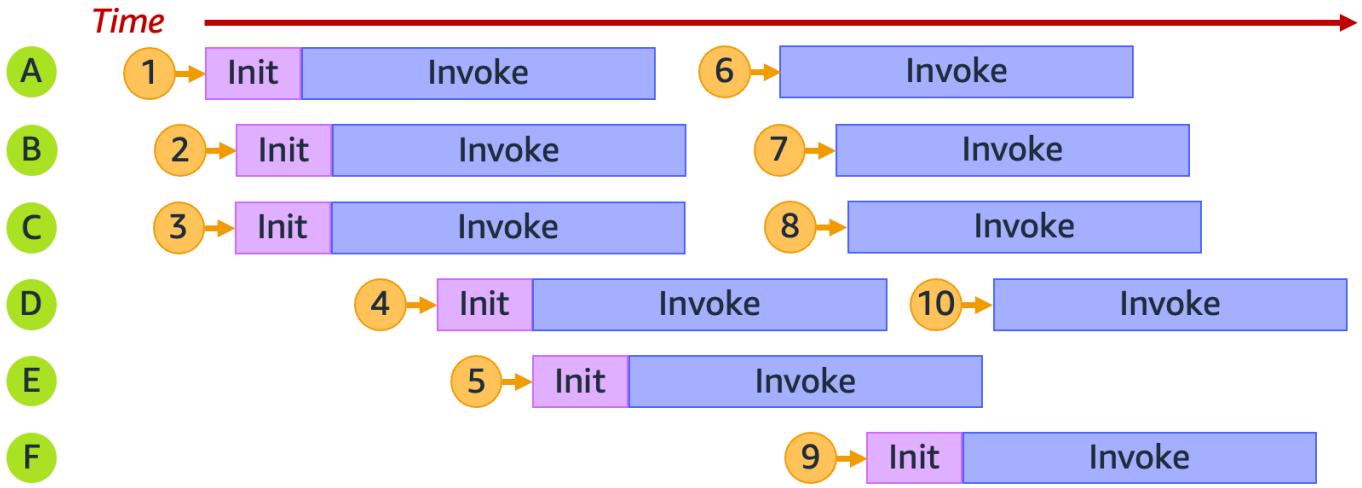


Nel diagramma precedente, Lambda riutilizza l'ambiente di esecuzione per gestire la seconda richiesta (rappresentata dal cerchio giallo con etichetta 2).

Finora ci siamo concentrati su una sola istanza dell'ambiente di esecuzione (ossia una simultaneità pari a 1). In pratica, Lambda potrebbe dover fornire più istanze dell'ambiente di esecuzione in parallelo in modo da gestire tutte le richieste in entrata. Quando la funzione riceve una nuova richiesta, può succedere una delle due cose:

- Se è disponibile un'istanza dell'ambiente di esecuzione pre-inizializzata, Lambda la utilizza per elaborare la richiesta.
- Altrimenti, Lambda crea una nuova istanza dell'ambiente di esecuzione.

Ad esempio, vediamo cosa accade quando la funzione riceve 10 richieste:

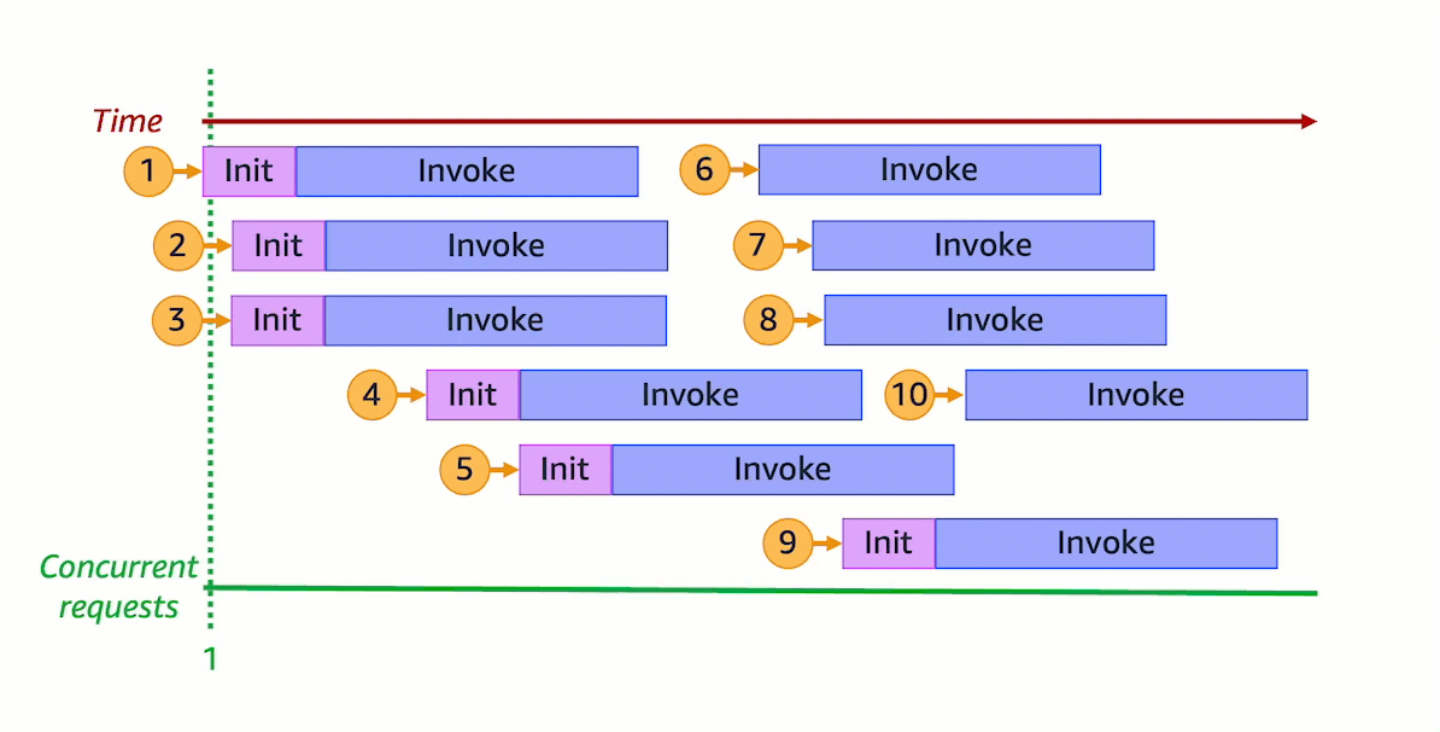


Nel diagramma precedente, ogni piano orizzontale rappresenta una singola istanza dell'ambiente di esecuzione (etichettata da A a F). Ecco come Lambda gestisce ogni richiesta:

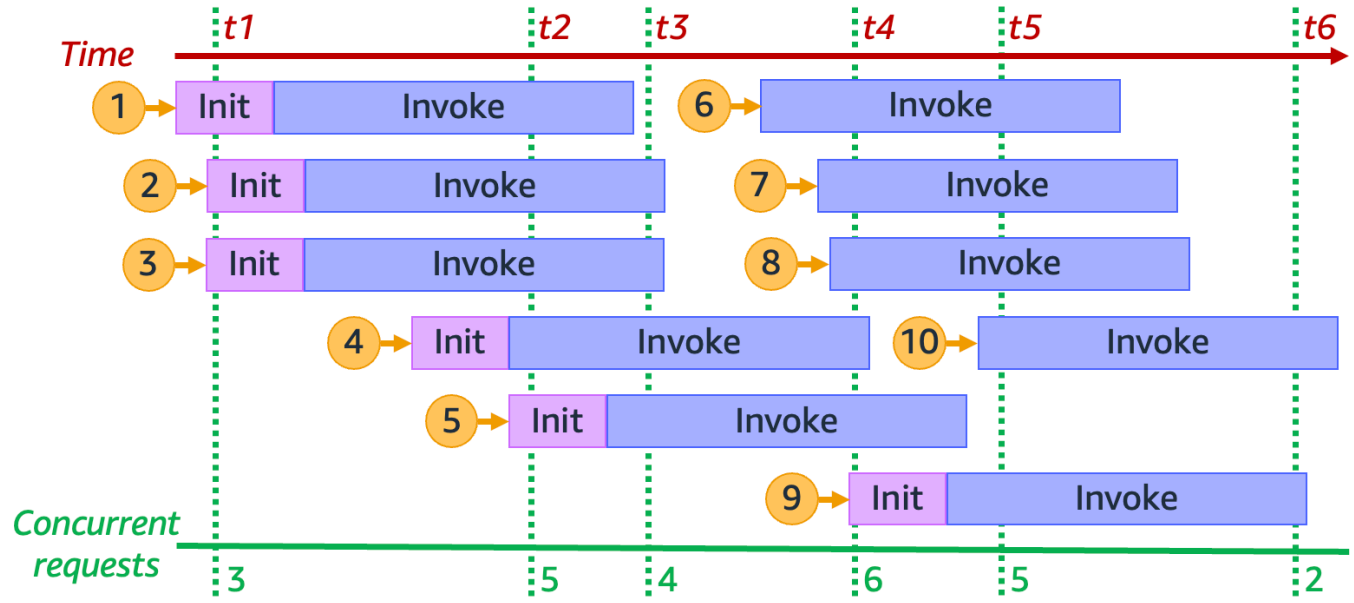
Richiesta	Comportamento di Lambda	Ragionamento
1	Fornisce il nuovo ambiente A	Questa è la prima richiesta; nessuna istanza dell'ambiente di esecuzione è disponibile.
2	Fornisce il nuovo ambiente B	L'istanza A dell'ambiente di esecuzione esistente è occupata.

Richiesta	Comportamento di Lambda	Ragionamento
3	Fornisce il nuovo ambiente C	Le istanze A e B dell'ambiente di esecuzione esistenti sono entrambe occupate.
4	Fornisce il nuovo ambiente D	Le istanze A, B e C dell'ambiente di esecuzione esistenti sono tutte occupate.
5	Fornisce il nuovo ambiente E	Le istanze A, B, C e D dell'ambiente di esecuzione esistenti sono tutte occupate.
6	Riutilizza l'ambiente A	L'istanza A dell'ambiente di esecuzione ha terminato l'elaborazione della richiesta 1 ed è ora disponibile.
7	Riutilizza l'ambiente B	L'istanza B dell'ambiente di esecuzione ha terminato l'elaborazione della richiesta 2 ed è ora disponibile.
8	Riutilizza l'ambienteC	L'istanza C dell'ambiente di esecuzione ha terminato l'elaborazione della richiesta 3 ed è ora disponibile.
9	Fornisce il nuovo ambiente F	Le istanze A, B, C, D e E dell'ambiente di esecuzione esistenti sono tutte occupate.
10	Riutilizza l'ambiente D	L'istanza D dell'ambiente di esecuzione ha terminato l'elaborazione della richiesta 4 ed è ora disponibile.

Man mano che la funzione riceve più richieste simultanee, Lambda aumenta il numero di istanze dell'ambiente di esecuzione in risposta. L'animazione seguente tiene traccia del numero di richieste simultanee nel tempo:



Osservando l'animazione precedente in sei momenti distinti nel tempo, otteniamo il seguente diagramma:



Nel diagramma precedente, possiamo tracciare una linea verticale in qualsiasi momento e contare il numero di ambienti che intersecano questa linea. Questo ci dà il numero di richieste simultanee in quel momento. Ad esempio, al momento `t1`, ci sono tre ambienti attivi che gestiscono tre richieste simultanee. Il numero massimo di richieste simultanee in questa simulazione si verifica nel momento `t4`, quando ci sono sei ambienti attivi che gestiscono sei richieste simultanee.

Per riassumere, la simultaneità della funzione è il numero di richieste simultanee che sono gestite nello stesso momento. In risposta a un aumento della simultaneità della funzione, Lambda fornisce più istanze dell'ambiente di esecuzione per soddisfare la domanda di richieste.

Calcolo della simultaneità per una funzione

In generale, la simultaneità di un sistema è la capacità di elaborare più di un'attività contemporaneamente. In Lambda, la simultaneità è il numero di richieste in corso che la funzione può gestire nello stesso momento. Un modo rapido e pratico per misurare la simultaneità di una funzione Lambda consiste nell'utilizzare la seguente formula:

```
Concurrency = (average requests per second) * (average request duration in seconds)
```

La simultaneità è diversa dalle richieste al secondo. Ad esempio, supponiamo che la funzione riceva in media 100 richieste al secondo. Se la durata media della richiesta è di 1 secondo, anche la simultaneità è 100:

```
Concurrency = (100 requests/second) * (1 second/request) = 100
```

Tuttavia, se la durata media della richiesta è di 500 ms, la simultaneità è 50:

```
Concurrency = (100 requests/second) * (0.5 second/request) = 50
```

Cosa significa in pratica una simultaneità pari a 50? Se la durata media della richiesta è di 500 ms, si può immaginare che un'istanza della funzione sia in grado di gestire 2 richieste al secondo. Quindi, sono necessarie 50 istanze della funzione per gestire un carico di 100 richieste al secondo. Una simultaneità pari a 50 significa che Lambda deve fornire 50 istanze dell'ambiente di esecuzione per gestire in modo efficiente questo carico di lavoro senza alcuna limitazione. Ecco come esprimerlo in forma di equazione:

```
Concurrency = (100 requests/second) / (2 requests/second) = 50
```

Se la funzione riceve il doppio del numero di richieste (200 richieste al secondo), ma richiede solo la metà del tempo per elaborare ciascuna richiesta (250 ms), la simultaneità è ancora 50:

```
Concurrency = (200 requests/second) * (0.25 second/request) = 50
```

Verifica se hai capito come funziona la simultaneità

Supponiamo di avere una funzione che richiede, in media, 200 ms per essere eseguita. Durante i picchi di carico, si osservano 5.000 richieste al secondo. Qual è la simultaneità della tua funzione durante i picchi di carico?

Risposta

La durata media della funzione è di 200 ms o 0,2 secondi. Utilizzando la formula della simultaneità, a partire da questi numeri si ottiene una simultaneità pari a 1.000:

```
Concurrency = (5,000 requests/second) * (0.2 seconds/request) = 1,000
```

In alternativa, una durata media della funzione di 200 ms significa che la funzione può elaborare 5 richieste al secondo. Per gestire il carico di lavoro di 5.000 richieste al secondo, sono necessarie 1.000 istanze dell'ambiente di esecuzione. Pertanto, la simultaneità è 1.000:

```
Concurrency = (5,000 requests/second) / (5 requests/second) = 1,000
```

Informazioni sulla simultaneità riservata e simultaneità fornita

Per impostazione predefinita, il tuo account ha un limite di simultaneità pari a 1.000 esecuzioni simultanee per tutte le funzioni in una regione. Le funzioni condividono questo pool di simultaneità 1.000 su richiesta. Se si esaurisce la simultaneità disponibile, la tua funzione subisce una limitazione della larghezza di banda della rete, ossia inizia a ignorare le richieste.

Alcune delle tue funzioni potrebbero essere più critiche di altre. Di conseguenza, potresti voler configurare le impostazioni di simultaneità in modo da garantire che le funzioni critiche ottengano la simultaneità di cui hanno bisogno. Lambda fornisce due tipi di controlli della simultaneità: la simultaneità riservata e la simultaneità fornita.

- Utilizza la simultaneità riservata per impostare il numero massimo e minimo di istanze simultanee per riservare una parte della simultaneità riservata del tuo account per una funzione. Ciò è utile se

non si desidera che altre funzioni occupino tutta la simultaneità non riservata disponibile. Quando una funzione ha la simultaneità riservata, nessun'altra funzione può utilizzare tale simultaneità.

- Usa la simultaneità fornita per pre-inizializzare una serie di istanze di ambiente per una funzione. Ciò è utile per ridurre le latenze di avviamento a freddo.

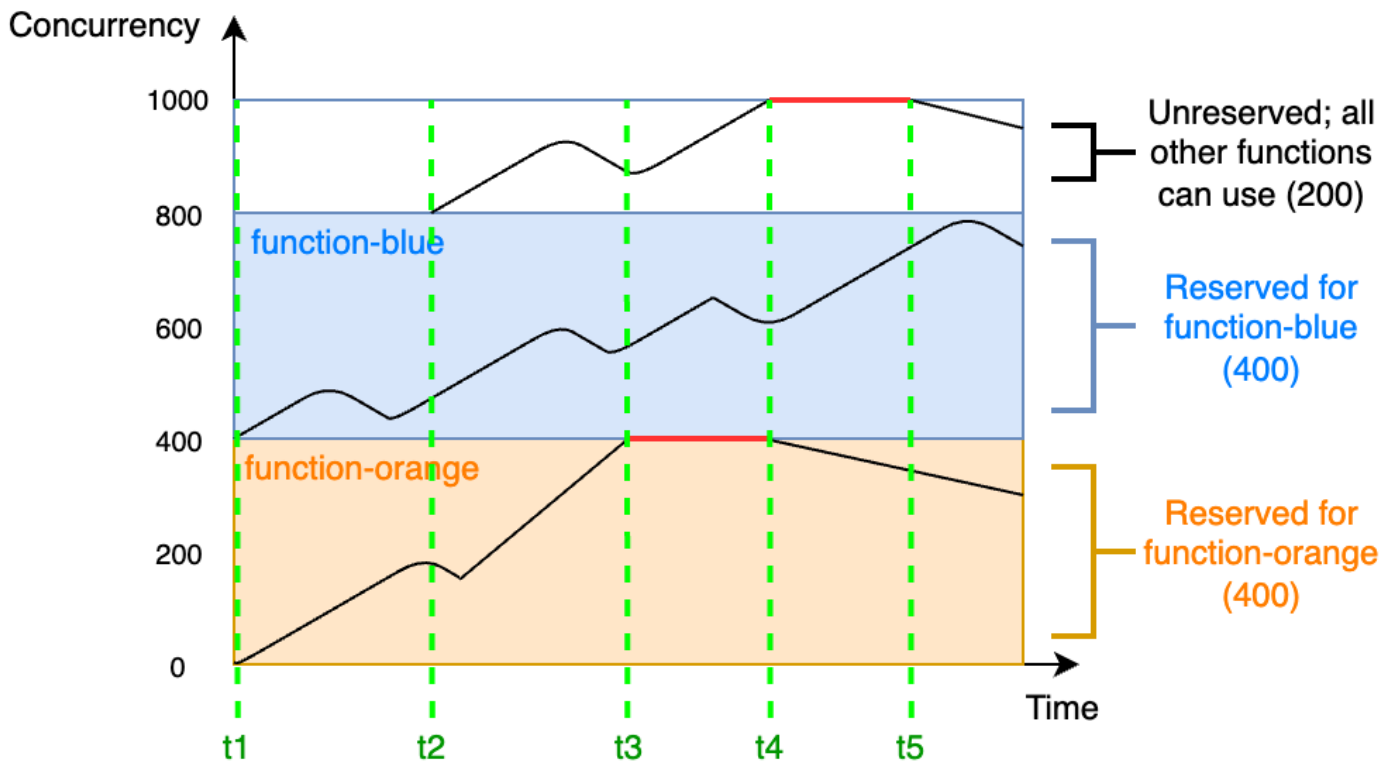
Simultaneità riservata

Se vuoi garantire che una certa quantità di simultaneità sia disponibile per la tua funzione in qualsiasi momento, usa la simultaneità riservata.

La simultaneità riservata è il numero massimo di istanze simultanee che desideri allocare alla funzione. Quando una funzione ha la simultaneità riservata dedicata, nessun'altra funzione potrà utilizzare tale simultaneità. In altre parole, l'impostazione della simultaneità riservata può influire sul pool di simultaneità disponibile per altre funzioni. Le funzioni che non dispongono di simultaneità riservata condividono il pool rimanente di simultaneità non riservata.

La configurazione della simultaneità riservata viene conteggiata ai fini del limite complessivo di simultaneità dell'account. Non è previsto alcun addebito per la configurazione della simultaneità riservata per una funzione.

Per comprendere meglio la simultaneità riservata, considera il seguente diagramma:



In questo diagramma, il limite di simultaneità dell'account per tutte le funzioni in questa regione è il limite predefinito di 1.000. Supponiamo di avere due funzioni critiche `function-blue` e `function-orange` che si aspettino di ottenere regolarmente volumi di chiamate elevati. Decidi di assegnare 400 unità di simultaneità riservata a `function-blue` e 400 unità di simultaneità riservata a `function-orange`. In questo esempio, tutte le altre funzioni del tuo account dovranno condividere le restanti 200 unità di simultaneità non riservata.

Il diagramma presenta cinque punti di interesse:

- Su `t1`, sia `function-orange` che `function-blue` iniziano a ricevere richieste. Ogni funzione inizia a utilizzare la parte allocata di unità di simultaneità riservata.
- Al momento `t2`, `function-orange` e `function-blue` stanno ricevendo costantemente più richieste. Allo stesso tempo, vengono implementate altre funzioni Lambda, che iniziano a ricevere richieste. La simultaneità riservata non viene assegnata a queste altre funzioni. Iniziano a utilizzare le restanti 200 unità di simultaneità non riservata.
- Su `t3`, `function-orange` raggiunge la simultaneità massima di 400. Sebbene sia presente una simultaneità inutilizzata altrove nel tuo account, `function-orange` non può accedervi. La linea

rossa indica che per `function-orange` si sta verificando una limitazione e Lambda potrebbe eliminare le richieste.

- Su `t4`, `function-orange` inizia a ricevere meno richieste e non è più limitato. Tuttavia, le altre funzioni registrano un picco di traffico e iniziano a rallentare. Sebbene sia presente una simultaneità inutilizzata altrove nel tuo account, queste funzioni non possono accedervi. La linea rossa indica che le altre funzioni sono soggette a limitazioni.
- Su `t5`, le altre funzioni iniziano a ricevere meno richieste e non sono più limitate.

Da questo esempio, nota che riservare la simultaneità ha i seguenti effetti:

- La funzione può essere dimensionata indipendentemente dalle altre funzioni nel tuo account. Tutte le funzioni dell'account nella stessa regione che non dispongono di simultaneità riservata condividono il pool di simultaneità non riservata. Senza simultaneità riservata, altre funzioni possono utilizzare potenzialmente tutta la simultaneità disponibile. Ciò impedisce alle funzioni critiche di aumentare quando necessario.
- La tua funzione non può essere aumentata orizzontalmente senza controllo. La simultaneità riservata pone un limite alla simultaneità massima della funzione. Ciò significa che la funzione non può utilizzare la simultaneità riservata ad altre funzioni o la simultaneità dal pool non riservato. Inoltre, la concorrenza riservata funge sia da limite inferiore che da limite superiore: riserva la capacità specificata esclusivamente per la funzione, impedendone al contempo la scalabilità oltre tale limite. È possibile riservare la simultaneità per evitare che la funzione utilizzi tutta la simultaneità disponibile nell'account oppure sovraccarichi le risorse in downstream.
- Potresti non essere in grado di utilizzare tutta la simultaneità disponibile del tuo account. La simultaneità di prenotazione viene conteggiata ai fini del limite di simultaneità dell'account, ma ciò significa anche che altre funzioni non possono utilizzare quella parte di simultaneità riservata. Se la tua funzione non utilizza tutta la simultaneità riservata, stai effettivamente sprecando quella simultaneità. Questo non è un problema a meno che altre funzioni del tuo account non possano trarre vantaggio dallo spreco di simultaneità.

Per gestire le impostazioni di simultaneità riservata per le tue funzioni, consulta la pagina Configurazione della simultaneità riservata per una funzione.

Simultaneità fornita

La simultaneità riservata viene utilizzata per definire il numero massimo di ambienti di esecuzione riservati a una funzione Lambda. Tuttavia, nessuno di questi ambienti è pre-inizializzato. Di

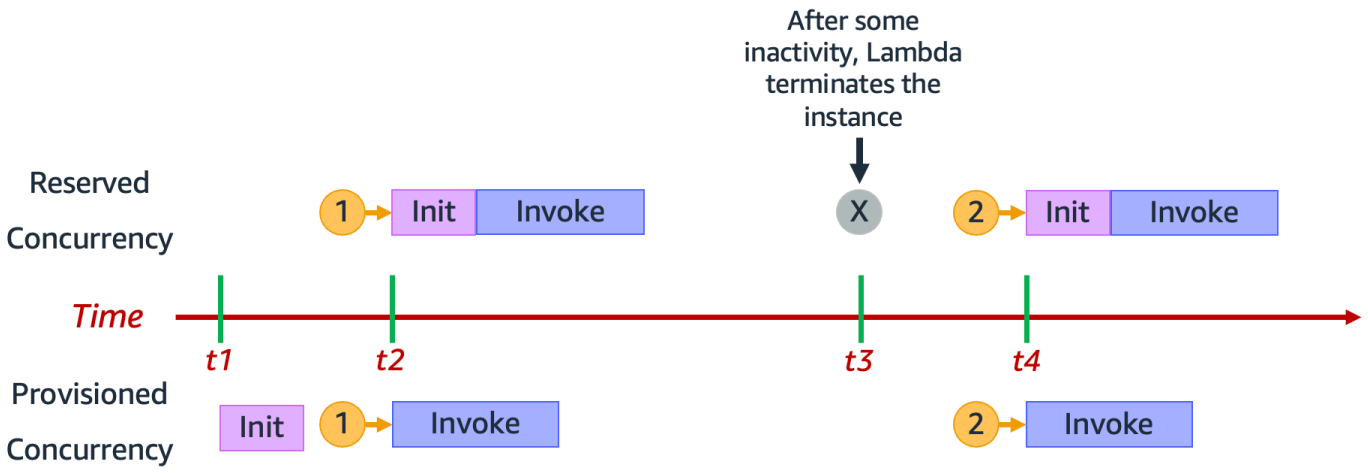
conseguenza, le chiamate delle funzioni potrebbero richiedere più tempo perché Lambda deve inizializzare il nuovo ambiente prima di poterlo utilizzare per richiamare la funzione. Quando Lambda deve inizializzare un nuovo ambiente per eseguire una chiamata, si parla di avvio a freddo. Per mitigare gli avviamenti a freddo, è possibile utilizzare la simultaneità fornita.

La simultaneità assegnata è il numero di ambienti di esecuzione pre-inizializzati che si desidera allocare alla funzione. Se si imposta la simultaneità fornita su una funzione, Lambda inizia quel numero di ambienti di esecuzione in modo che siano preparati a rispondere immediatamente alle richieste della funzione.

> **Note**
>
> La configurazione della simultaneità fornita comporta addebiti sul tuo account. Se lavori con i runtime Java 11 o Java 17, puoi anche utilizzare Lambda SnapStart per mitigare i problemi di avvio a freddo senza costi aggiuntivi. SnapStart utilizza snapshot memorizzati nella cache dell'ambiente di esecuzione per migliorare significativamente le prestazioni di avvio. Non è possibile utilizzare sia SnapStart sia la simultaneità fornita nella stessa versione della funzione. Per ulteriori informazioni sulle funzionalità, le limitazioni e le regioni supportate da SnapStart, consulta la pagina Migliorare le prestazioni di avvio con Lambda SnapStart.

Quando si utilizza la simultaneità fornita, Lambda riavvia comunque gli ambienti di esecuzione in background. Ad esempio, ciò può verificarsi dopo un errore di invocazione. Tuttavia, in qualsiasi momento, Lambda garantisce sempre che il numero di ambienti pre-inizializzati sia uguale al valore dell'impostazione di simultaneità fornita dalla funzione. È importante sottolineare che, anche se utilizzi la concorrenza fornita, puoi comunque riscontrare un ritardo di avvio a freddo se Lambda deve reimpostare l'ambiente di esecuzione.

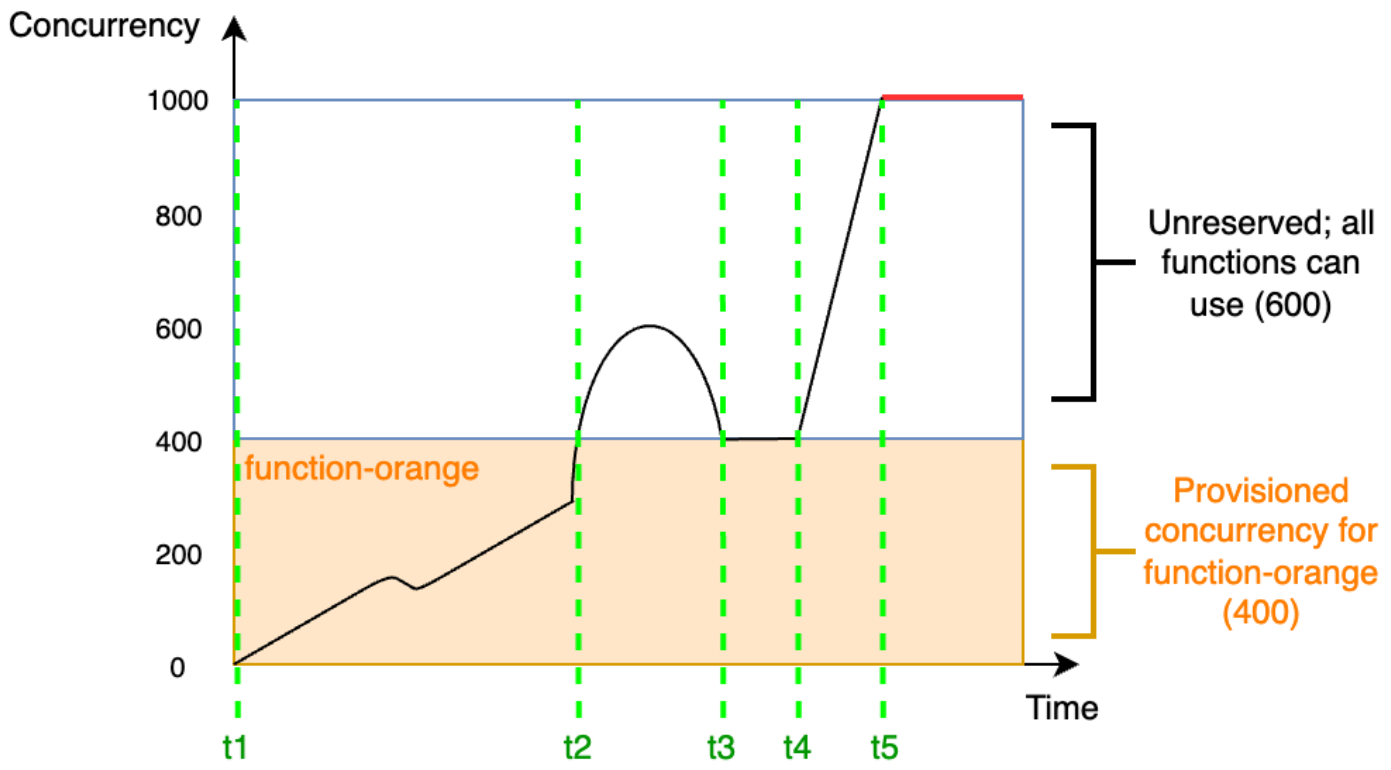
Al contrario, quando si utilizza la simultaneità riservata, Lambda può terminare completamente un ambiente dopo un periodo di inattività. Il diagramma seguente illustra ciò confrontando il ciclo di vita di un singolo ambiente di esecuzione quando si configura la funzione utilizzando la simultaneità riservata rispetto alla simultaneità fornita.



Il diagramma presenta 4 punti di interesse:

Orario	Simultaneità riservata	Simultaneità fornita
t1	Non succede niente.	Lambda pre-inizializza un'istanza dell'ambiente di esecuzione.
t2	Arriva la richiesta 1. Lambda deve inizializzare una nuova istanza dell'ambiente di esecuzione.	Arriva la richiesta 1. Lambda utilizza l'istanza pre-inizi alizzata dell'ambiente.
t3	Dopo una certa inattività, Lambda termina l'istanza dell'ambiente attivo.	Non succede niente.
t4	Arriva la richiesta 2. Lambda deve inizializzare una nuova istanza dell'ambiente di esecuzione.	Arriva la richiesta 2. Lambda utilizza l'istanza pre-inizi alizzata dell'ambiente.

Per comprendere meglio la simultaneità fornita, considera il seguente diagramma:



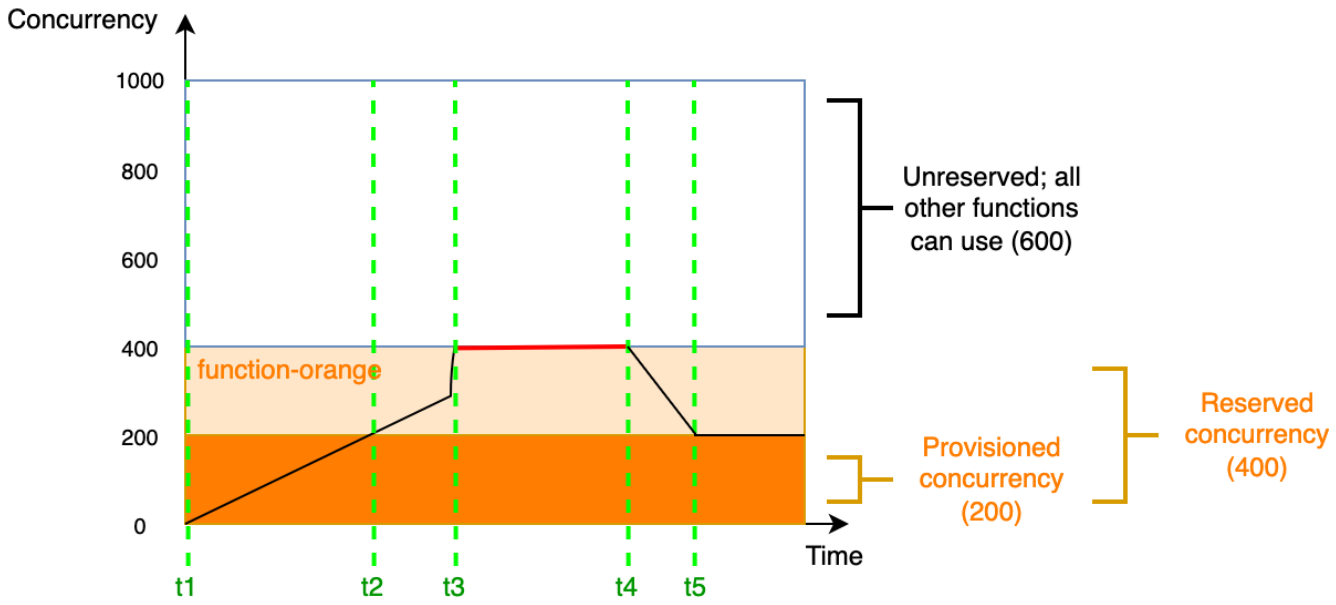
In questo diagramma, hai un limite di simultaneità dell'account pari a 1.000. Decidi di assegnare 400 unità di simultaneità fornita a `function-orange`. Tutte le funzioni del tuo account, inclusa `function-orange`, possono utilizzare le restanti 600 unità di simultaneità non riservata.

Il diagramma presenta cinque punti di interesse:

- Su `t1`, `function-orange` inizia a ricevere richieste. Poiché Lambda ha pre-inizializzato 400 istanze dell'ambiente di esecuzione, `function-orange` è pronta per la chiamata immediata.
- Su `t2`, `function-orange` raggiunge le 400 richieste simultanee. Di conseguenza, `function-orange` esaurisce la simultaneità fornita. Tuttavia, poiché è ancora disponibile la simultaneità non riservata, Lambda può utilizzarla per gestire richieste aggiuntive a `function-orange` (senza limitazioni). Lambda deve creare nuove istanze per soddisfare queste richieste e la funzione potrebbe presentare latenze di avvio a freddo.
- Su `t3`, `function-orange` torna a 400 richieste simultanee dopo un breve picco di traffico. Lambda è nuovamente in grado di gestire tutte le richieste senza latenze di avvio a freddo.
- Su `t4`, le funzioni del tuo account subiscono un'impennata di traffico. Questa impennata può provenire da `function-orange` o da qualsiasi altra funzione del tuo account. Per gestire queste richieste, Lambda utilizza la simultaneità non riservata.

- Su `t5`, le funzioni del tuo account raggiungono il limite massimo di simultaneità pari a 1.000 e subiscono una limitazione.

L'esempio precedente considerava solo la simultaneità assegnata. In pratica, su una funzione è possibile impostare sia la simultaneità fornita che la simultaneità riservata. Ciò sarebbe possibile se si avesse una funzione che gestisce un carico costante di chiamate nei giorni feriali, ma registra regolarmente picchi di traffico durante i fine settimana. In questo caso, è possibile utilizzare la simultaneità fornita per impostare una quantità di base di ambienti per gestire le richieste durante i giorni feriali e utilizzare la simultaneità riservata per gestire i picchi del fine settimana. Considera il seguente diagramma:



In questo diagramma, supponiamo di configurare 200 unità di simultaneità fornita e 400 unità di simultaneità riservata per `function-orange`. Poiché è stata configurata la simultaneità riservata, `function-orange` non può utilizzare nessuna delle 600 unità di simultaneità non riservata.

Il diagramma presenta cinque punti di interesse:

- Su `t1`, `function-orange` inizia a ricevere richieste. Poiché Lambda ha pre-inizializzato 200 istanze dell'ambiente di esecuzione, `function-orange` è pronta per la chiamata immediata.
- Su `t2`, `function-orange` utilizza tutta la simultaneità fornita. `function-orange` può continuare a gestire le richieste utilizzando la simultaneità riservata, ma queste richieste potrebbero presentare latenze di avvio a freddo.

- Su `t3`, `function-orange` raggiunge le 400 richieste simultanee. Di conseguenza, `function-orange` utilizza tutta la sua simultaneità riservata. Poiché `function-orange` non può utilizzare la simultaneità non riservata, le richieste iniziano a rallentare.
- Su `t4`, `function-orange` inizia a ricevere meno richieste e non è più limitato.
- Al momento `t5`, `function-orange` scende a 200 richieste simultanee, quindi tutte le richieste possono nuovamente utilizzare la simultaneità assegnata (ossia nessuna latenza di avvio a freddo).

Sia la simultaneità riservata che la simultaneità fornita vengono conteggiate ai fini del limite di simultaneità dell'account e delle quote regionali. In altre parole, l'impostazione della simultaneità riservata e fornita può influire sul pool di simultaneità disponibile per altre funzioni. La configurazione della simultaneità fornita comporta addebiti sul tuo Account AWS.

> **Note**
>
> Se la quantità di simultaneità assegnata nelle versioni e negli alias di una funzione si somma alla simultaneità riservata della funzione, tutte le chiamate vengono eseguite sulla simultaneità assegnata. Questa configurazione ha anche l'effetto di limitare il throttling della funzione (`$LATEST`), che ne impedisce l'esecuzione. Non è possibile allocare più simultaneità fornita rispetto alla simultaneità riservata per una funzione.

Per gestire le impostazioni di simultaneità riservata delle tue funzioni, consulta la pagina Configurazione della simultaneità fornita per una funzione. Per automatizzare il dimensionamento della simultaneità assegnata in base a una pianificazione o all'utilizzo dell'applicazione, consulta la pagina Utilizzo di Application Auto Scaling per automatizzare la gestione della simultaneità fornita.

Come Lambda alloca la simultaneità fornita

La simultaneità fornita non è disponibile online immediatamente dopo la configurazione. Lambda avvia l'allocazione della simultaneità fornita dopo uno o due minuti di preparazione. Per ogni funzione, Lambda può fornire fino a 6.000 ambienti di esecuzione al minuto, indipendentemente da Regione AWS. È esattamente la stessa della velocità di scalabilità simultanea per le funzioni.

Quando invii una richiesta di allocazione della simultaneità fornita, non puoi accedere a nessuno di questi ambienti finché Lambda non completa l'allocazione. Ad esempio, se richiedi 5.000 simultaneità

fornita, nessuna delle tue richieste potrà utilizzare la simultaneità fornita fino a che Lambda completa l'allocazione dei 5.000 ambienti di esecuzione.

Simultaneità riservata e simultaneità fornita di Lambda.

Di seguito è riportata una tabella che riassume e mette a confronto la simultaneità riservata e la simultaneità assegnata.

Argomento	Simultaneità riservata	Simultaneità fornita
Definizione	Numero massimo di istanze dell'ambiente di esecuzione per la funzione.	Imposta il numero massimo di istanze dell'ambiente di esecuzione pre-fornito per la funzione.
Comportamento del provision ing	Lambda fornisce nuove istanze su richiesta.	Lambda fornisce le istanze in anticipo, ossia prima che la funzione inizi a ricevere richieste.
Comportamento dell'avvio a freddo	È possibile una latenza di avvio a freddo, poiché Lambda deve creare nuove istanze su richiesta.	La latenza di avvio a freddo non è possibile, poiché Lambda non deve creare istanze on demand.
Comportamento della limitazio ne	Funzione limitata quando viene raggiunto il limite di simultaneità riservata.	Se la simultaneità riservata non è impostata: quando viene raggiunto il limite di simultane ità fornita la funzione utilizza la simultaneità non riservata. Se è impostata la simultane ità riservata: quando viene raggiunto il limite di simultane ità riservata la funzione viene limitata.

Argomento	Simultaneità riservata	Simultaneità fornita
Comportamento predefinito se non impostato	La funzione utilizza la simultaneità non riservata disponibile nel tuo account.	Lambda non fornisce alcuna istanza in anticipo. Invece, se la simultaneità riservata non è impostata: la funzione utilizza la simultaneità non riservata disponibile nel tuo account. Se è impostata la simultaneità riservata: la funzione utilizza la simultaneità riservata.
Prezzi	Nessun costo aggiuntivo.	Incorre in costi aggiuntivi.

Informazioni sulla simultaneità e le richieste al secondo

Come indicato nella sezione precedente, la simultaneità è diversa dalle richieste al secondo. Questa distinzione è particolarmente rilevante quando si lavora con funzioni con una durata media della richiesta inferiore a 100 ms.

In tutte le funzioni del tuo account, Lambda applica un limite di richieste al secondo pari a 10 volte la simultaneità dell'account. Ad esempio, poiché il limite di simultaneità predefinito dell'account è 1.000, le funzioni dell'account possono gestire un massimo di 10.000 richieste al secondo.

Ad esempio, consideriamo una funzione con una durata media della richiesta di 50 ms. A 20.000 richieste al secondo, ecco la simultaneità di questa funzione:

```
Concurrency = (20,000 requests/second) * (0.05 second/request) = 1,000
```

In base a questo risultato, potresti aspettarti che il limite di simultaneità dell'account pari a 1.000 sia sufficiente per gestire questo carico. Tuttavia, a causa del limite di 10.000 richieste al secondo, la funzione può gestire solo 10.000 richieste al secondo sulle 20.000 richieste totali. Questa funzione subisce una limitazione.

Pertanto, quando si configurano le impostazioni di simultaneità per le funzioni, è necessario considerare sia la simultaneità sia le richieste al secondo. In questo caso, devi richiedere un aumento

del limite di simultaneità dell'account a 2.000, poiché ciò aumenterebbe il limite totale di richieste al secondo a 20.000.

> **Note**
>
> In base a questo limite di richieste al secondo, non è corretto affermare che ogni ambiente di esecuzione Lambda può gestire solo un massimo di 10 richieste al secondo. Invece di osservare il carico su ogni singolo ambiente di esecuzione, Lambda considera solo la simultaneità complessiva e le richieste complessive al secondo nel calcolo delle quote.

Metti alla prova la tua comprensione della simultaneità (funzioni inferiori a 100 ms)

Supponiamo di avere una funzione che richiede, in media, 20 ms per essere eseguita. Durante i picchi di carico, si registrano 30.000 richieste al secondo. Qual è la simultaneità della tua funzione durante i picchi di carico?

Risposta

La durata media della funzione è di 20 ms o 0,02 secondi. Utilizzando la formula della simultaneità, a partire da questi numeri si ottiene una simultaneità pari a 600:

```
Concurrency = (30,000 requests/second) * (0.02 seconds/request) = 600
```

Per impostazione predefinita, il limite di simultaneità dell'account pari a 1.000 sembra sufficiente per gestire questo carico. Tuttavia, il limite di 10.000 richieste al secondo non è sufficiente per gestire le 30.000 richieste in entrata al secondo. Per soddisfare appieno le 30.000 richieste, devi richiedere un aumento del limite di simultaneità dell'account a 3.000 o più.

Il limite di richieste al secondo si applica a tutte le quote in Lambda che prevedono la simultaneità. In altre parole, si applica alle funzioni on demand sincrone, alle funzioni che utilizzano la simultaneità fornita e al comportamento di scalabilità della simultaneità. Ad esempio, ecco alcuni scenari in cui è necessario considerare attentamente sia i limiti di simultaneità che quelli di richieste al secondo:

- Una funzione che utilizza la simultaneità on demand può registrare un aumento improvviso di 500 richieste simultanee ogni 10 secondi o di 5.000 richieste al secondo ogni 10 secondi, a seconda dell'evento che si verifica per primo.

- Si supponga di disporre di una funzione con un'allocazione di simultaneità fornita pari a 10. Questa funzione si ripercuote sulla simultaneità on demand dopo 10 operazioni simultanee o 100 richieste al secondo, a seconda dell'evento che si verifica per primo.

Quote di simultaneità

Lambda imposta le quote per la quantità totale di simultaneità che è possibile utilizzare in tutte le funzioni di una regione. Queste quote esistono su due livelli:

- A livello di account: per impostazione predefinita, le funzioni possono avere fino a 1.000 unità di simultaneità. Per aumentare questo limite, consulta Richiesta di aumento delle quote nella Guida per l'utente di Service Quotas.
- A livello di funzione, per impostazione predefinita, è possibile riservare fino a 900 unità di simultaneità in tutte le regioni. Indipendentemente dal limite totale di simultaneità dell'account, Lambda riserva sempre 100 unità di simultaneità per le funzioni che non la riservano esplicitamente. Ad esempio, se hai aumentato il limite di simultaneità del tuo account a 2.000, puoi riservare fino a 1.900 unità di simultaneità a livello di funzione.
- Sia a livello di account che a livello di funzione, Lambda impone anche un limite di richieste al secondo pari a 10 volte la quota di simultaneità corrispondente. Ad esempio, ciò si applica alla simultaneità a livello di account, alle funzioni che utilizzano la simultaneità on demand, alle funzioni che utilizzano la simultaneità fornita e al comportamento di scalabilità della simultaneità. Per ulteriori informazioni, consulta the section called “Informazioni sulla simultaneità e le richieste al secondo”.

Per verificare la quota di simultaneità a livello di account corrente, utilizza l'AWS Command Line Interface (AWS CLI) per eseguire il seguente comando:

```
aws lambda get-account-settings
```

L'output restituito dovrebbe essere simile al seguente:

```
{
    "AccountLimit": {
        "TotalCodeSize": 80530636800,
        "CodeSizeUnzipped": 262144000,
        "CodeSizeZipped": 52428800,
        "ConcurrentExecutions": 1000,
```

```
        "UnreservedConcurrentExecutions": 900
    },
    "AccountUsage": {
        "TotalCodeSize": 410759889,
        "FunctionCount": 8
    }
}
```

`ConcurrentExecutions` è la quota di simultaneità totale a livello di account. `UnreservedConcurrentExecutions` è la quantità di simultaneità riservata che puoi ancora destinare alle tue funzioni.

Man mano che la funzione riceve altre richieste, Lambda aumenta automaticamente il numero di ambienti di esecuzione per gestire le richieste fino al raggiungimento della quota di simultaneità dell'account. Tuttavia, per proteggersi dall'eccessivo aumento in risposta a improvvisi picchi di traffico, Lambda limita la velocità di dimensionamento delle funzioni. Questa velocità di scalabilità della simultaneità è la velocità massima alla quale le funzioni del tuo account possono scalare in risposta all'aumento delle richieste. Si tratta della velocità con cui Lambda può creare nuovi ambienti di esecuzione. Il tasso di scalabilità della simultaneità è diverso dal limite di simultaneità a livello di account, che è la quantità totale di simultaneità disponibile per le funzioni.

In ogni Regione AWS e per ogni funzione, la velocità di dimensionamento della simultaneità è di 1.000 istanze dell'ambiente di esecuzione ogni 10 secondi (o 10.000 richieste al secondo ogni 10 secondi). In altre parole, ogni 10 secondi, Lambda può allocare al massimo 1.000 istanze aggiuntive dell'ambiente di esecuzione o gestire 10.000 richieste aggiuntive al secondo, per ciascuna delle tue funzioni.

Di solito, non è necessario preoccuparsi di questa limitazione. La velocità di dimensionamento di Lambda è sufficiente per la maggior parte dei casi d'uso.

È importante sottolineare che la velocità di scalabilità della simultaneità è un limite a livello di funzione. Ciò significa che ogni funzione del tuo account può dimensionarsi indipendentemente dalle altre funzioni.

Per ulteriori informazioni sul comportamento di dimensionamento, consulta la pagina Comportamento del dimensionamento Lambda.

Configurazione della simultaneità riservata per una funzione

In Lambda, la simultaneità è il numero di richieste in transito che la funzione sta gestendo attualmente. Sono disponibili due tipi di controlli di simultaneità:

- Simultaneità riservata: rappresenta il numero massimo e minimo di istanze simultanee allocate alla funzione. Quando una funzione ha la simultaneità riservata, nessun'altra funzione può utilizzare tale simultaneità. La simultaneità riservata è utile per garantire che le funzioni più critiche abbiano sempre una simultaneità sufficiente per gestire le richieste in arrivo. Inoltre, la simultaneità riservata può essere utilizzata per limitare la simultaneità ed evitare un sovraccarico di risorse a valle, come le connessioni al database. La simultaneità riservata funge sia da limite inferiore che da limite superiore: riserva la capacità specificata esclusivamente per la funzione, impedendone al contempo l'aumento oltre tale limite. La configurazione della simultaneità riservata per una funzione non comporta alcun addebito ulteriore.
- Simultaneità fornita: il numero di ambienti di esecuzione pre-inizializzati che desideri allocare alla funzione. Questi ambienti di esecuzione sono pronti a rispondere immediatamente alle richieste di funzioni in arrivo. La simultaneità fornita è utile per ridurre le latenze di avvii a freddo per le funzioni ed è progettata per rendere disponibili funzioni con tempi di risposta a due cifre in millisecondi. In genere, i carichi di lavoro interattivi traggono il massimo vantaggio da questa funzionalità. Si tratta di applicazioni in cui gli utenti avviano richieste, come le applicazioni Web e per dispositivi mobili, e sono le più sensibili alla latenza. I carichi di lavoro asincroni, come le pipeline di elaborazione dei dati, sono spesso meno sensibili alla latenza e quindi di solito non richiedono la simultaneità fornita. La configurazione della simultaneità fornita comporta ulteriori addebiti sul tuo Account AWS.

In questo argomento viene descritta in dettaglio la modalità gestire e configurare la simultaneità riservata. Per una panoramica concettuale di questi due tipi di controlli della simultaneità, consulta la sezione Simultaneità riservata e simultaneità fornita. Per informazioni sulla configurazione della simultaneità fornita, consulta la sezione the section called "Configurazione della simultaneità fornita".

> **Note**
>
> Le funzioni Lambda collegate a uno strumento di mappatura dell'origine degli eventi Amazon MQ hanno una simultaneità massima predefinita. Per Apache Active MQ, il numero massimo di istanze simultanee è 5. Per Rabbit MQ, il numero massimo di istanze simultanee è 1. L'impostazione della simultaneità sottoposta a provisioning o riservata per la funzione non

modifica questi limiti. Per richiedere un aumento della simultaneità massima predefinita quando si utilizza Amazon MQ, contatta Supporto.

Sezioni

- Configurazione della simultaneità riservata
- Stima accurata della simultaneità riservata richiesta per una funzione

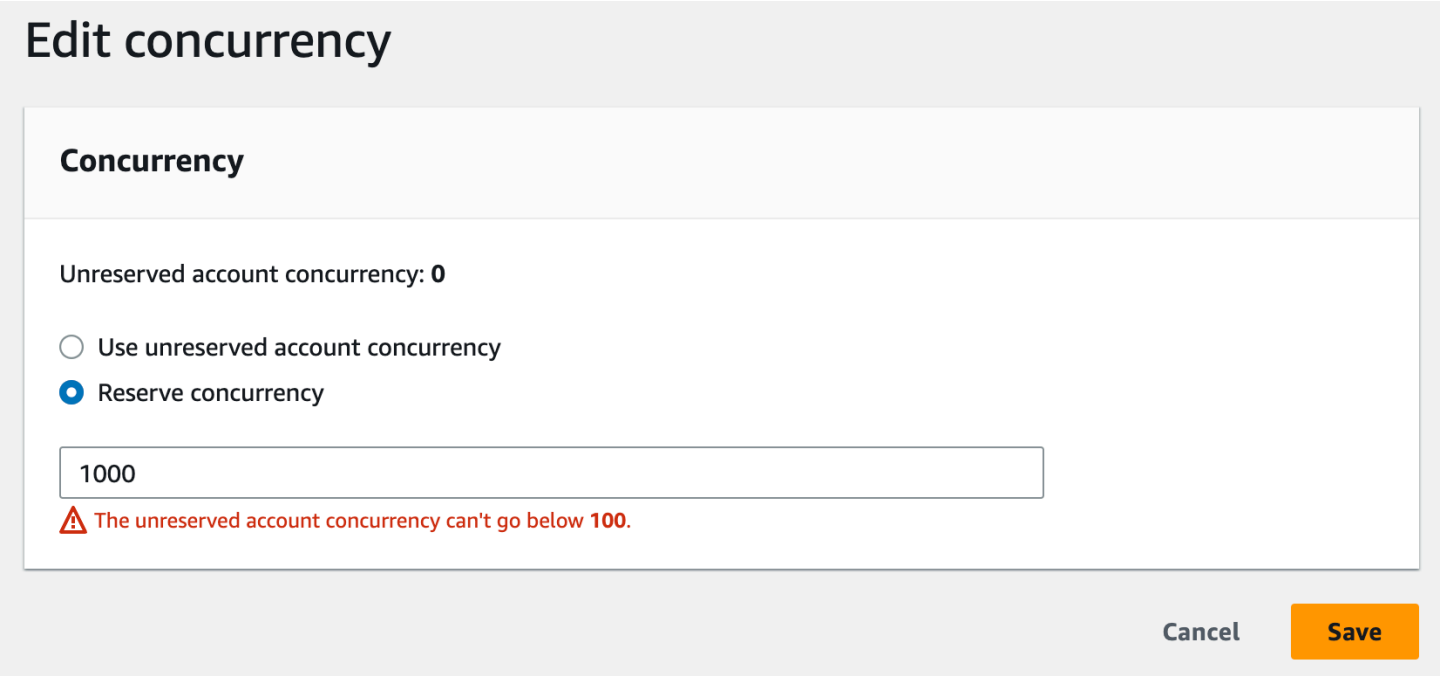
Configurazione della simultaneità riservata

È possibile configurare le simultaneità fornita per una funzione utilizzando la console Lambda o l'API Lambda.

Riserva della simultaneità per una funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la funzione per la quale desideri prenotare la simultaneità.
3. Scegliere Configuration (Configurazione) e quindi scegliere Concurrency (Simultaneità).
4. In Concurrency (Concorrenza), scegliere Edit (Modifica).
5. Scegliere Reserve concurrency (Impegna concorrenza). Inserire la quantità di simultaneità da riservare per la funzione.
6. Selezionare Salva.

Puoi prenotare fino al valore di Simultaneità dell'account non riservata meno 100. Le restanti 100 unità di simultaneità sono destinate a funzioni che non utilizzano la simultaneità riservata. Ad esempio, se l'account ha un limite di simultaneità di 1.000, non puoi utilizzare tutte le 1.000 unità di simultaneità per una singola funzione.



La riserva della simultaneità per una funzione può influire sul pool di simultaneità disponibile per altre funzioni. Ad esempio, se riservi 100 unità di simultaneità per `function-a`, le altre funzioni del tuo account devono condividere le 900 unità di simultaneità rimanenti, anche se `function-a` non utilizza tutte le 100 unità di simultaneità riservata.

Per limitare intenzionalmente una funzione, imposta la simultaneità riservata su 0. In questo modo, viene interrotta la capacità della funzione di elaborare ulteriori eventi fino a quando il limite non viene rimosso.

Per configurare la simultaneità riservata con l'API Lambda, utilizza le operazioni dell'API seguenti.

- PutFunctionConcurrency
- GetFunctionConcurrency
- DeleteFunctionConcurrency

Per configurare la simultaneità riservata con l'AWS Command Line Interface (CLI), utilizza il comando `put-function-concurrency`. Il comando seguente riserva 100 unità di simultaneità per una funzione denominata `my-function`:

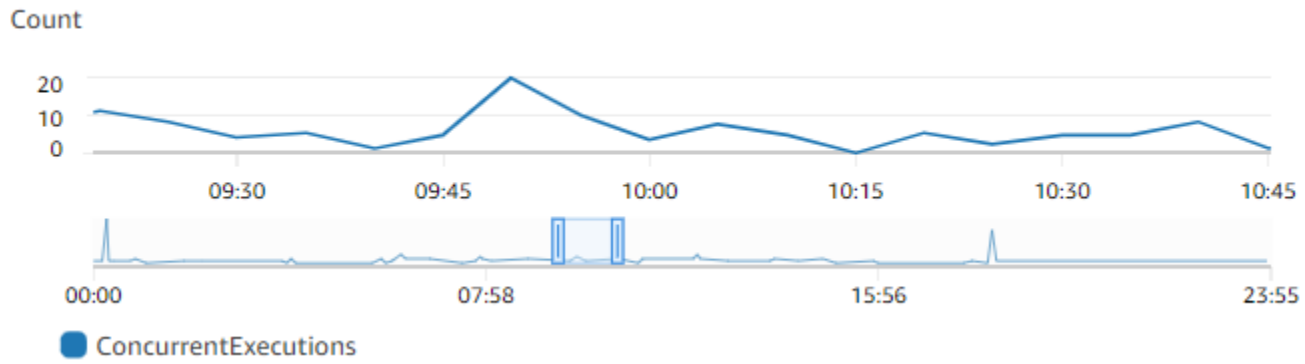
```
aws lambda put-function-concurrency --function-name my-function \
    --reserved-concurrent-executions 100
```

L'output restituito dovrebbe essere simile al seguente:

```
{
    "ReservedConcurrentExecutions": 100
}
```

Stima accurata della simultaneità riservata richiesta per una funzione

Se la tua funzione attualmente gestire del traffico, puoi facilmente visualizzarne i parametri di simultaneità utilizzando i parametri di CloudWatch. In particolare, il parametro `ConcurrentExecutions` mostra il numero di chiamate simultanee per ciascuna funzione del tuo account.



Il grafico precedente mostra che questa funzione gestisce in media da 5 a 10 richieste simultanee in ogni momento e in un giorno tipico raggiunge un massimo di 20 richieste. Supponiamo che nel tuo account siano presenti molte altre funzioni. Se questa funzione è fondamentale per la tua applicazione e non vuoi che alcuna richiesta venga tralasciata, puoi utilizzare un numero uguale a o maggiore di 20 come impostazione della simultaneità riservata.

In alternativa, ricorda che è possibile calcolare la simultaneità anche utilizzando la seguente formula:

```
Concurrency = (average requests per second) * (average request duration in seconds)
```

Moltiplicando le richieste medie al secondo per la durata media delle richieste in secondi, si ottiene una stima approssimativa della quantità di simultaneità che è necessario riservare. Puoi stimare le richieste medie al secondo utilizzando il parametro `Invocation` e la durata media delle richieste in secondi utilizzando il parametro `Duration`. Per ulteriori dettagli, consulta Utilizzo dei parametri di CloudWatch Logs con Lambda.

Devi anche essere esperto dei vincoli di throughput a monte e a valle. Sebbene le funzioni Lambda si dimensioni perfettamente in base al carico, le dipendenze a monte e a valle potrebbero non avere le stesse capacità di throughput. Se devi limitare il valore massimo di scalabilità della tua funzione, puoi configurare la simultaneità riservata sulla funzione.

Configurazione della simultaneità fornita per una funzione

In Lambda, la simultaneità è il numero di richieste in transito che la funzione sta gestendo attualmente. Sono disponibili due tipi di controlli di simultaneità:

- Simultaneità riservata: rappresenta il numero massimo di istanze simultanee allocate alla funzione. Quando una funzione ha la simultaneità riservata, nessun'altra funzione può utilizzare tale simultaneità. La simultaneità riservata è utile per garantire che le funzioni più critiche abbiano sempre una simultaneità sufficiente per gestire le richieste in arrivo. Inoltre, la concorrenza riservata può essere utilizzata per limitare la concorrenza ed evitare un sovraccarico di risorse a valle, come le connessioni al database. La concorrenza riservata funge sia da limite inferiore che da limite superiore: riserva la capacità specificata esclusivamente per la funzione, impedendone al contempo la scalabilità oltre tale limite. La configurazione della simultaneità riservata per una funzione non comporta alcun addebito ulteriore.
- Simultaneità fornita: il numero di ambienti di esecuzione pre-inizializzati che desideri allocare alla funzione. Questi ambienti di esecuzione sono pronti a rispondere immediatamente alle richieste di funzioni in arrivo. La simultaneità fornita è utile per ridurre le latenze di avvii a freddo per le funzioni ed è progettata per rendere disponibili funzioni con tempi di risposta a due cifre in millisecondi. In genere, i carichi di lavoro interattivi traggono il massimo vantaggio da questa funzionalità. Si tratta di applicazioni in cui gli utenti avviano richieste, come le applicazioni Web e per dispositivi mobili, e sono le più sensibili alla latenza. I carichi di lavoro asincroni, come le pipeline di elaborazione dei dati, sono spesso meno sensibili alla latenza e quindi di solito non richiedono la simultaneità fornita. La configurazione della simultaneità fornita comporta ulteriori addebiti sull'Account AWS.

In questo argomento viene descritta in dettaglio la modalità gestire e configurare la simultaneità fornita. Per una panoramica concettuale di questi due tipi di controlli della simultaneità, consulta la sezione Simultaneità riservata e simultaneità fornita. Per ulteriori informazioni sulla configurazione della simultaneità riservata, consulta la sezione the section called “Configurazione della simultaneità riservata”.

> **Note**
>
> Le funzioni Lambda collegate a uno strumento di mappatura dell'origine degli eventi Amazon MQ hanno una simultaneità massima predefinita. Per Apache Active MQ, il numero massimo di istanze simultanee è 5. Per Rabbit MQ, il numero massimo di istanze simultanee è 1. L'impostazione della simultaneità sottoposta a provisioning o riservata per la funzione non

modifica questi limiti. Per richiedere un aumento della simultaneità massima predefinita quando si utilizza Amazon MQ, contatta Supporto.

Sezioni

- Configurazione della simultaneità fornita
- Stima accurata della simultaneità fornita richiesta per una funzione
- Ottimizzazione del codice della funzione quando si utilizza la simultaneità fornita
- Utilizzo di variabili di ambiente per visualizzare e controllare il comportamento della simultaneità fornita
- Informazioni sul comportamento di registrazione e fatturazione con la simultaneità fornita
- Utilizzo di Application Auto Scaling per automatizzare la gestione della simultaneità fornita

Configurazione della simultaneità fornita

È possibile configurare le impostazioni di simultaneità fornita per una funzione utilizzando la console Lambda o l'API Lambda.

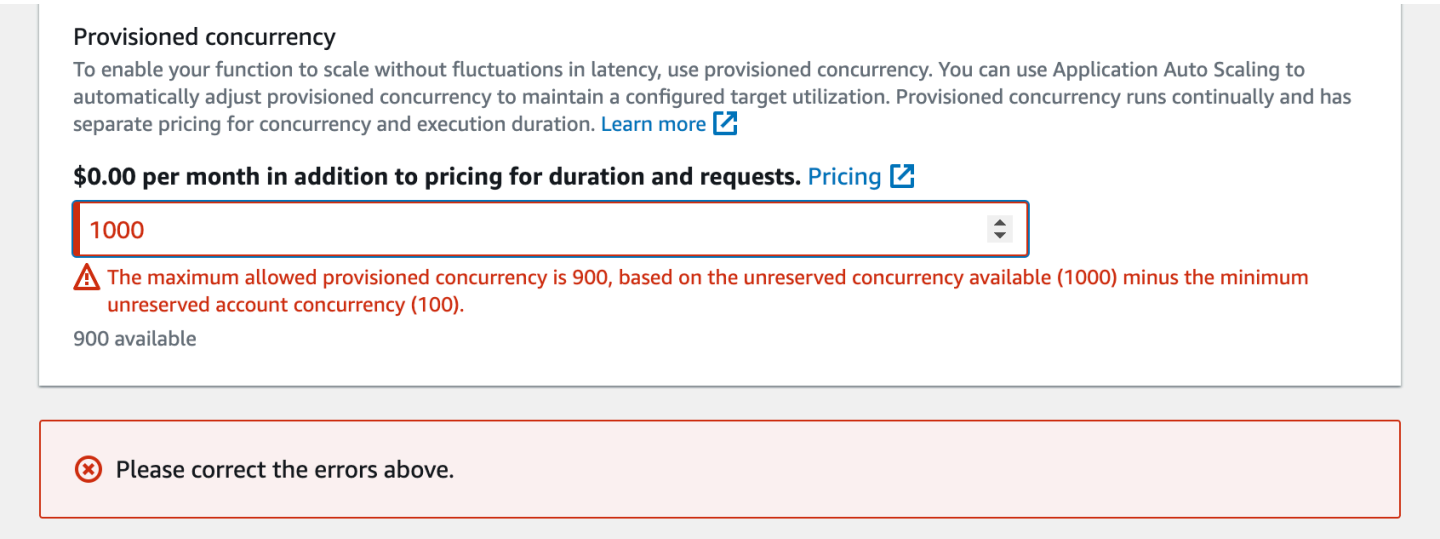
Allocazione della simultaneità fornita per una funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la funzione per la quale desideri allocare la simultaneità fornita.
3. Scegliere Configuration (Configurazione) e quindi scegliere Concurrency (Simultaneità).
4. In Provisioned concurrency configurations (Configurazioni di simultaneità fornita), scegliere Add configuration (Aggiungi configurazione).
5. Scegli il tipo di qualificatore e l'alias o la versione.

> **Note**
>
> Non è possibile utilizzare la simultaneità fornita con la versione $LATEST di alcuna funzione.
> Se la funzione dispone di un'origine eventi, assicurati che tale origine punti all'alias corretto o alla versione corretta della funzione. In caso contrario, la funzione non utilizzerà gli ambienti di simultaneità fornita.

6. Inserisci un numero in Simultaneità fornita.
7. Selezionare Salva.

Puoi configurare fino al valore massimo di Simultaneità dell'account non riservata meno 100. Le restanti 100 unità di simultaneità sono destinate a funzioni che non utilizzano la simultaneità riservata. Ad esempio, se il tuo account ha un limite di simultaneità di 1.000 e non hai assegnato simultaneità riservata o fornita ad alcuna delle tue altre funzioni, puoi configurare un massimo di 900 unità di simultaneità fornita per una singola funzione.



La configurazione della simultaneità fornita per una funzione influisce sul pool di simultaneità disponibile per altre funzioni. Ad esempio, se configuri 100 unità di simultaneità fornita per `function-a`, le altre funzioni nell'account devono condividere le 900 unità di simultaneità rimanenti. Ciò vale anche se `function-a` non utilizza tutte le 100 unità.

Per la stessa funzione è possibile allocare sia la simultaneità riservata che la simultaneità fornita. In questi casi, la concorrenza fornita non può superare la concorrenza riservata.

Questa limitazione si estende alle versioni della funzione. La simultaneità fornita massima che si può assegnare a una versione specifica della funzione corrisponde alla simultaneità riservata della funzione meno la simultaneità fornita su altre versioni della funzione.

Per configurare la simultaneità fornita con l'API Lambda, utilizza le operazioni dell'API seguenti.

- PutProvisionedConcurrencyConfig
- GetProvisionedConcurrencyConfig
- ListProvisionedConcurrencyConfigs

- DeleteProvisionedConcurrencyConfig

Ad esempio, per configurare la simultaneità fornita con l'AWS Command Line Interface (CLI), utilizza il comando `put-provisioned-concurrency-config`. Il comando seguente alloca 100 unità di simultaneità fornita per l'alias `BLUE` di una funzione denominata `my-function`:

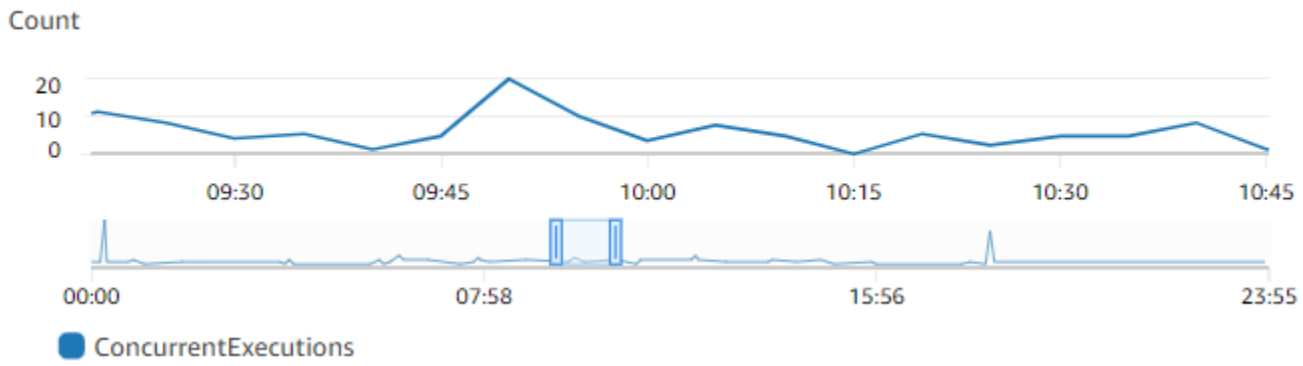
```
aws lambda put-provisioned-concurrency-config --function-name my-function \
  --qualifier BLUE \
  --provisioned-concurrent-executions 100
```

L'output restituito dovrebbe essere simile al seguente:

```
{
  "Requested ProvisionedConcurrentExecutions": 100,
  "Allocated ProvisionedConcurrentExecutions": 0,
  "Status": "IN_PROGRESS",
  "LastModified": "2023-01-21T11:30:00+0000"
}
```

Stima accurata della simultaneità fornita richiesta per una funzione

Puoi visualizzare i parametri di concorrenza di qualsiasi funzione attiva utilizzando i parametri di CloudWatch. Nello specifico, il parametro `ConcurrentExecutions` mostra il numero di invocazioni simultanee per le funzioni nell'account.



Il grafico precedente mostra che questa funzione gestisce in media da 5 a 10 richieste simultanee in qualsiasi momento dato, e ha un picco di 20 richieste. Supponiamo che nel tuo account siano presenti molte altre funzioni. Se questa funzione è fondamentale per la tua applicazione e necessiti

di una risposta a bassa latenza per ogni invocazione, configura almeno 20 unità di simultaneità fornita.

Ricorda che puoi anche calcolare la simultaneità utilizzando la seguente formula:

```
Concurrency = (average requests per second) * (average request duration in seconds)
```

Per stimare la quantità di simultaneità necessaria, moltiplica le richieste medie al secondo per la durata media delle richieste in secondi. Puoi stimare le richieste medie al secondo utilizzando il parametro `Invocation` e la durata media delle richieste in secondi utilizzando il parametro `Duration`.

Quando si configura la simultaneità fornita, Lambda suggerisce di aggiungere un buffer del 10% oltre alla quantità di simultaneità che generalmente occorre alla funzione. Ad esempio, se la funzione di solito raggiunge il picco di 200 richieste simultanee, imposta la simultaneità fornita su 220 (200 richieste simultanee + 10% = 220 unità di simultaneità fornita).

Ottimizzazione del codice della funzione quando si utilizza la simultaneità fornita

Se utilizzi la simultaneità fornita, prendi in considerazione la possibilità di ristrutturare il codice della funzione per ottimizzarlo per una bassa latenza. Per le funzioni eseguite con simultaneità fornita, Lambda esegue qualsiasi codice di inizializzazione, come il caricamento di librerie e la creazione di istanze di client, in fase di allocazione. Pertanto, è consigliabile trasferire quanti più processi di inizializzazione di istanze all'esterno del gestore della funzione principale per evitare di influire sulla latenza durante le invocazioni effettive della funzione. Al contrario, l'inizializzazione di librerie o la creazione di istanze di client all'interno del codice principale dell'handler implica che la funzione deve ripetere l'esecuzione ogni volta che viene richiamata, a prescindere dal fatto che tu stia utilizzando la simultaneità fornita o meno.

Per le invocazioni on demand, Lambda potrebbe dover eseguire nuovamente il codice di inizializzazione ogni volta che la funzione subisce un avvio a freddo. Per tali funzioni, puoi scegliere di rinviare l'inizializzazione di una funzionalità specifica fino a quando la funzione non ne ha necessità. Ad esempio, considera il seguente flusso di controllo per un gestore Lambda:

```
def handler(event, context):
    ...
    if ( some_condition ):
        // Initialize CLIENT_A to perform a task
```

```
    else:
        // Do nothing
```

Nell'esempio precedente, invece di inizializzare `CLIENT_A` all'esterno del gestore principale, lo sviluppatore lo ha inizializzato all'interno dell'istruzione `if`. In tal modo, Lambda esegue il codice solo se la condizione `some_condition` è soddisfatta. Se esegui l'inizializzazione di `CLIENT_A` all'esterno del gestore principale, Lambda esegue quel codice a ogni avvio a freddo. Ciò può aumentare la latenza complessiva.

Puoi misurare le partenze a freddo man mano che Lambda aumenta aggiungendo il monitoraggio X-Ray alla tua funzione. Una funzione che utilizza la simultaneità fornita non presenta un comportamento di avvio a freddo poiché l'ambiente di esecuzione viene preparato prima dell'invocazione. Tuttavia, la simultaneità fornita deve essere applicata a una versione o alias specifici di una funzione, non alla versione $LATEST. Nel caso in cui continui a riscontrare un comportamento di avvio a freddo, assicurati di richiamare la versione dell'alias con la simultaneità fornita configurata.

Utilizzo di variabili di ambiente per visualizzare e controllare il comportamento della simultaneità fornita

È possibile che la funzione utilizzi tutta la simultaneità fornita. Per gestire il traffico in eccesso, Lambda utilizza istanze on demand. Per determinare il tipo di inizializzazione che Lambda ha utilizzato per un ambiente specifico, controlla il valore della variabile di ambiente `AWS_LAMBDA_INITIALIZATION_TYPE`. Questa variabile ammette due valori possibili: `provisioned-concurrency` o `on-demand`. Il valore di `AWS_LAMBDA_INITIALIZATION_TYPE` è immutabile e rimane costante per tutta la durata dell'ambiente. Per verificare il valore di una variabile di ambiente nel codice della funzione, consulta Recupero delle variabili di ambiente Lambda.

Se utilizzi runtime .NET 6 o .NET 7, puoi configurare la variabile di ambiente `AWS_LAMBDA_DOTNET_PREJIT` per migliorare la latenza delle funzioni, anche laddove non utilizzino la simultaneità fornita. Il runtime .NET impiega la compilazione e l'inizializzazione lenta per ciascuna libreria a cui il codice effettua la chiamata per la prima volta. Di conseguenza, la prima invocazione di una funzione Lambda può richiedere più tempo delle successive. Per ovviare a questo problema, puoi scegliere tra tre valori per `AWS_LAMBDA_DOTNET_PREJIT`:

- `ProvisionedConcurrency`: Lambda esegue la compilazione JIT anticipata per tutti gli ambienti utilizzando la simultaneità fornita. Si tratta del valore di default.
- `Always`: Lambda esegue la compilazione JIT anticipata per ogni ambiente, anche se la funzione non utilizza la simultaneità fornita.

- `Never`: Lambda disattiva la compilazione JIT anticipata per tutti gli ambienti.

Informazioni sul comportamento di registrazione e fatturazione con la simultaneità fornita

Per gli ambienti con simultaneità fornita, il codice di inizializzazione della funzione viene eseguito durante l'allocazione e periodicamente mentre Lambda ricicla le istanze attive dell'ambiente. Tuttavia, Lambda addebita i costi di inizializzazione anche se l'istanza non elabora mai richieste. La simultaneità fornita viene eseguita continuamente e prevede una fatturazione separata rispetto ai costi di inizializzazione e invocazione. Per maggiori dettagli, consulta Prezzi di AWS Lambda.

Inoltre, quando configuri una funzione Lambda con la simultaneità fornita, Lambda preinizializza l'ambiente di esecuzione in modo che sia disponibile prima delle richieste di invocazione della funzione. Lambda registra il campo Init Duration della funzione in un evento di registro Platform-InitReport in formato di registrazione JSON ogni volta che l'ambiente viene inizializzato. Per visualizzare questo evento di registro, configura il livello di registro JSON su almeno. `INFO` Puoi anche utilizzare l'API di telemetria per utilizzare gli eventi della piattaforma in cui viene riportato il campo Init Duration.

Utilizzo di Application Auto Scaling per automatizzare la gestione della simultaneità fornita

Puoi utilizzare Application Auto Scaling per gestire la simultaneità fornita in base a una pianificazione o all'utilizzo. Se la funzione riceve modelli di traffico prevedibili, utilizza il dimensionamento pianificato. Se desideri che la funzione mantenga una percentuale di utilizzo specifica, utilizza una policy di dimensionamento con monitoraggio degli obiettivi.

> **Note**
>
> Se utilizzi Application Auto Scaling per gestire la simultaneità fornita dalla funzione, assicurati di configurare prima un valore di simultaneità iniziale. Se la funzione non dispone di un valore di simultaneità fornita iniziale, Application Auto Scaling potrebbe non gestire correttamente la scalabilità delle funzioni.

Dimensionamento programmato

Con Application Auto Scaling, puoi creare una pianificazione personalizzata in base alle variazioni di carico prevedibili. Per ulteriori informazioni ed esempi, consulta le sezioni Dimensionamento pianificato per Application Auto Scaling nella Guida per l'utente di Application Auto Scaling e Pianificazione della simultaneità fornita di AWS Lambda per i picchi di utilizzo ricorrenti nel blog di calcolo AWS.

Monitoraggio degli obiettivi

Con il monitoraggio degli obiettivi, Application Auto Scaling crea e gestisce un set di allarmi di CloudWatch in base al modo in cui definisci la policy di dimensionamento. Quando questi allarmi si attivano, Application Auto Scaling regola automaticamente la quantità di ambienti allocati utilizzando la simultaneità fornita. Utilizza il monitoraggio degli obiettivi per le applicazioni che non presentano modelli di traffico prevedibili.

Per dimensionare la simultaneità fornita utilizzando il tracciamento degli obiettivi, utilizza le operazioni `RegisterScalableTarget` e `PutScalingPolicy` dell'API di Application Auto Scaling. Ad esempio, se utilizzi l'AWS Command Line Interface (CLI), segui questi passaggi:

1. Registrare l'alias di una funzione come target di dimensionamento. Nell'esempio seguente viene registrato l'alias BLUE di una funzione denominata `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. Applicare una policy di dimensionamento alla destinazione. Nell'esempio seguente viene configurato Application Auto Scaling per regolare la configurazione della simultaneità fornita per un alias per mantenerne l'utilizzo intorno al 70% (ma è possibile applicare qualsiasi valore compreso tra il 10% e il 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" }}'
```

L'output visualizzato dovrebbe essere di questo tipo:

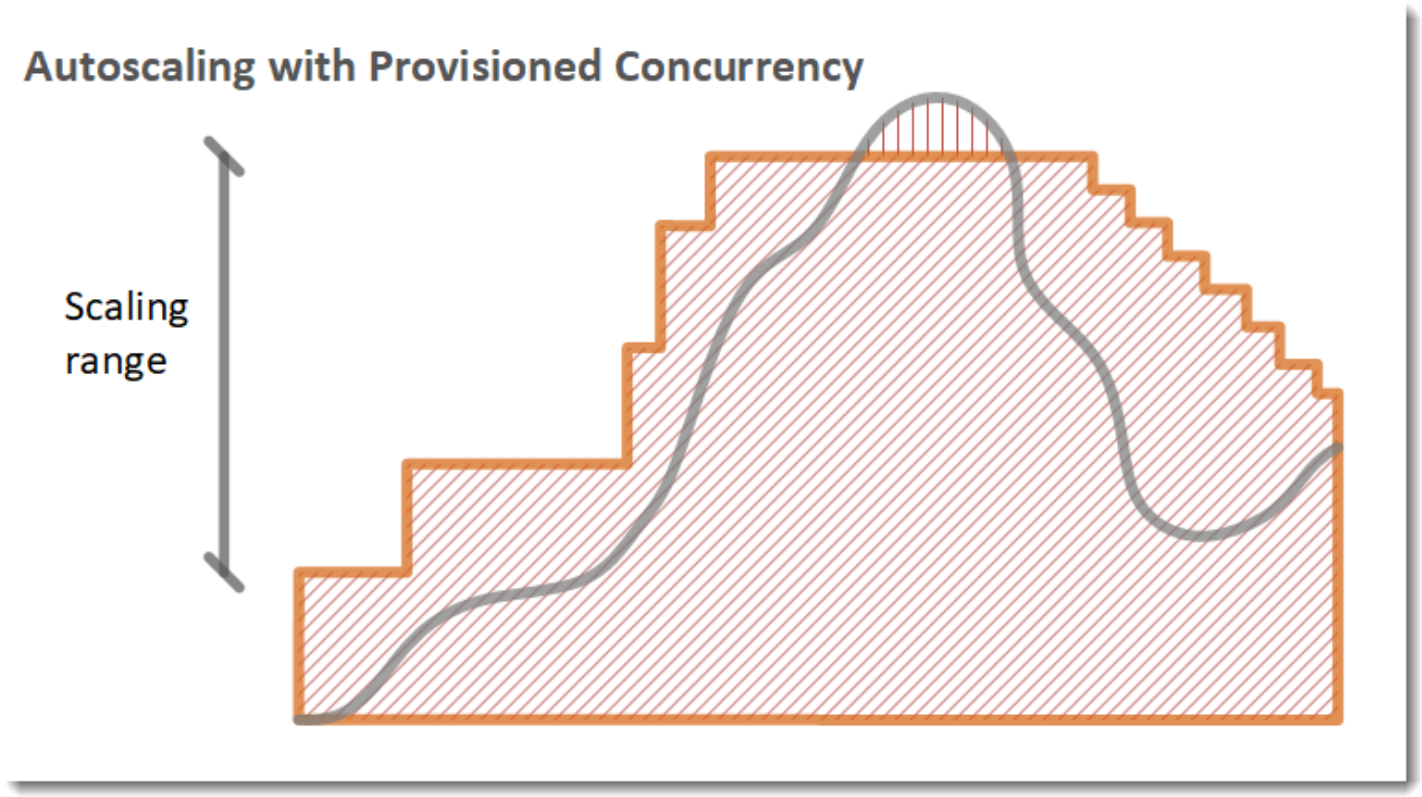
```
{
    "PolicyARN": "arn:aws:autoscaling:us-
east-2:123456789012:scalingPolicy:12266dbb-1524-xmpl-a64e-9a0a34b996fa:resource/lambda/
function:my-function:BLUE:policyName/my-policy",
    "Alarms": [
        {
            "AlarmName": "TargetTracking-function:my-function:BLUE-AlarmHigh-aed0e274-
xmpl-40fe-8cba-2e78f000c0a7",
            "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 crea due allarmi in CloudWatch. Il primo allarme si attiva quando l'utilizzo della simultaneità fornita supera costantemente il 70%. In questo caso, Application Auto Scaling alloca più simultaneità fornita per ridurre l'utilizzo. Il secondo allarme si attiva quando l'utilizzo è costantemente inferiore al 63% (90% dell'obiettivo del 70%). In questo caso, Application Auto Scaling riduce la simultaneità di provisioning dell'alias.

> **Note**
>
> Lambda emette la `ProvisionedConcurrencyUtilization` metrica solo quando la funzione è attiva e riceve richieste. Durante i periodi di inattività, non viene emessa alcuna metrica e gli allarmi con autoscaling entreranno in funzione. `INSUFFICIENT_DATA` Di

conseguenza, Application Auto Scaling non sarà in grado di regolare la simultaneità fornita dalla funzione. Ciò può comportare una fatturazione imprevista.

Nell'esempio seguente, una funzione si dimensiona tra una quantità minima e massima di concorrenza di cui è stato eseguito il provisioning in base all'utilizzo.



Legenda

- Istanze di funzione
- Richieste aperte
- Simultaneità fornita
- Simultaneità standard

Quando il numero di richieste aperte aumenta, Application Auto Scaling aumenta la simultaneità fornita a grandi scaglioni fino a raggiungere il massimo configurato. Successivamente, la funzione può continuare a dimensionarsi in base alla simultaneità non riservata standard se non hai raggiunto il limiti di simultaneità dell'account. Quando l'utilizzo scende e rimane basso, Application Auto Scaling riduce la simultaneità fornita a piccoli scaglioni periodici.

Entrambi gli allarmi di Application Auto Scaling utilizzano per impostazione predefinita la statistica media. Le funzioni che rilevano picchi di traffico rapidi potrebbero non attivare questi allarmi. Ad esempio, supponiamo che la funzione Lambda venga eseguita velocemente (ovvero in 20-100 ms) e che il traffico presenti picchi rapidi. In tal caso, il numero di richieste supera la simultaneità fornita allocata durante il picco. Tuttavia, Application Auto Scaling richiede che il carico di picco duri almeno 3 minuti allo scopo di fornire ambienti aggiuntivi. Inoltre, entrambi gli allarmi di CloudWatch richiedono 3 punti dati che raggiungano la media obiettivo per attivare la policy di dimensionamento automatico. Se la funzione registra rapidi picchi di traffico, l'utilizzo della statistica Maximum anziché della statistica Average può essere più efficace nel dimensionare la simultaneità fornita per ridurre al minimo gli avvii a freddo.

Per ulteriori informazioni sulle policy di dimensionamento con monitoraggio degli obiettivi, consulta la sezione Policy di dimensionamento con monitoraggio degli obiettivi per Application Auto Scaling.

Comportamento del dimensionamento Lambda

Man mano che la funzione riceve altre richieste, Lambda aumenta automaticamente il numero di ambienti di esecuzione per gestire le richieste fino al raggiungimento della quota di simultaneità dell'account. Tuttavia, per proteggersi dall'eccessivo aumento in risposta a improvvisi picchi di traffico, Lambda limita la velocità di dimensionamento delle funzioni. Questa velocità di scalabilità della simultaneità è la velocità massima alla quale le funzioni del tuo account possono scalare in risposta all'aumento delle richieste. Si tratta della velocità con cui Lambda può creare nuovi ambienti di esecuzione. Il tasso di scalabilità della simultaneità è diverso dal limite di simultaneità a livello di account, che è la quantità totale di simultaneità disponibile per le funzioni.

Velocità di dimensionamento della simultaneità

In ogni Regione AWS e per ogni funzione, la velocità di dimensionamento della simultaneità è di 1.000 istanze dell'ambiente di esecuzione ogni 10 secondi (o 10.000 richieste al secondo ogni 10 secondi). In altre parole, ogni 10 secondi, Lambda può allocare al massimo 1.000 istanze aggiuntive dell'ambiente di esecuzione o gestire 10.000 richieste aggiuntive al secondo, per ciascuna delle tue funzioni.

Di solito, non è necessario preoccuparsi di questa limitazione. La velocità di dimensionamento di Lambda è sufficiente per la maggior parte dei casi d'uso.

È importante sottolineare che la velocità di scalabilità della simultaneità è un limite a livello di funzione. Ciò significa che ogni funzione del tuo account può dimensionarsi indipendentemente dalle altre funzioni.

> **Note**
>
> In pratica, Lambda fa del suo meglio per ricaricare la velocità di dimensionamento della simultaneità in modo continuo nel tempo, anziché con una singola ricarica di 1.000 unità ogni 10 secondi.

Lambda non accumula porzioni inutilizzate della velocità di dimensionamento della simultaneità. Ciò significa che in qualsiasi momento, la velocità di dimensionamento è sempre al massimo di 1.000 unità di simultaneità. Ad esempio, se non utilizzi nessuna delle 1.000 unità di simultaneità disponibili in un intervallo di 10 secondi, non accumulerai 1.000 unità aggiuntive nel successivo intervallo di 10 secondi. La tua velocità di dimensionamento della simultaneità sarà ancora di 1.000 unità nel successivo intervallo di 10 secondi.

Finché la tua funzione continua a ricevere un numero crescente di richieste, Lambda si dimensionerà alla massima velocità disponibile, fino al limite di simultaneità del tuo account. Puoi limitare la quantità di simultaneità che le singole funzioni possono utilizzare configurando la simultaneità riservata. Quando le richieste arrivano più velocemente della capacità di dimensionamento della funzione, oppure quando la funzione ha raggiunto la simultaneità massima, le altre richieste restituiscono esito negativo con un errore di limitazione della larghezza di banda della rete (429).

Monitoraggio della simultaneità

Lambda emette parametri di Amazon CloudWatch utili per monitorare la simultaneità delle funzioni. Questo argomento illustra questi parametri e spiega come interpretarli.

Sezioni

- Parametri generici di simultaneità
- Parametri della simultaneità fornita
- Lavorare con il parametro ClaimedAccountConcurrency

Parametri generici di simultaneità

Per monitorare la simultaneità delle funzioni Lambda, utilizza i seguenti parametri. La granularità di ogni parametro è di 1 minuto.

- `ConcurrentExecutions`: il numero di chiamate simultanee attive in un determinato momento. Lambda emette questo parametro per tutte le funzioni, gli alias e le versioni. Per qualsiasi funzione nella console Lambda, Lambda visualizza il grafico di `ConcurrentExecutions` in modo nativo nella scheda Monitoraggio, sotto Parametri. Visualizza questo parametro utilizzando MAX.
- `UnreservedConcurrentExecutions`: il numero di chiamate simultanee attive che utilizzano la simultaneità non riservata. Lambda emette questo parametro per tutte le funzioni in una regione. Visualizza questo parametro utilizzando MAX.
- `ClaimedAccountConcurrency`: la quantità di simultaneità che non è disponibile per le invocazioni on demand. `ClaimedAccountConcurrency` corrisponde a `UnreservedConcurrentExecutions` più la quantità di simultaneità allocata (ovvero la simultaneità totale riservata più la simultaneità totale fornita). Se `ClaimedAccountConcurrency` supera il limite di simultaneità dell'account, puoi richiedere un limite di simultaneità più elevato. Visualizza questo parametro utilizzando MAX. Per ulteriori informazioni, consulta Lavorare con il parametro `ClaimedAccountConcurrency`.

Parametri della simultaneità fornita

Per monitorare le funzioni Lambda che utilizzano la simultaneità fornita, utilizza i seguenti parametri. La granularità di ogni parametro è di 1 minuto.

- `ProvisionedConcurrentExecutions`: il numero di istanze dell'ambiente di esecuzione che stanno attivamente elaborando una chiamata con la simultaneità fornita. Lambda emette questo parametro per ogni versione e alias di funzione per cui sia configurata la simultaneità fornita. Visualizza questo parametro utilizzando MAX.

`ProvisionedConcurrentExecutions` non è uguale al numero totale di unità di simultaneità fornita allocate. Ad esempio, si supponga di allocare 100 unità di simultaneità fornita a una versione di funzione. In un dato minuto, se al massimo 50 di questi 100 ambienti di esecuzione gestivano le chiamate simultaneamente, il valore di MAX (`ProvisionedConcurrentExecutions`) è 50.

- `ProvisionedConcurrencyInvocations`: il numero di volte in cui Lambda richiama il codice di funzione tramite la simultaneità fornita. Lambda emette questo parametro per ogni versione e alias di funzione per cui sia configurata la simultaneità fornita. Visualizza questo parametro utilizzando SUM.

`ProvisionedConcurrencyInvocations` differisce da `ProvisionedConcurrentExecutions` per il fatto che `ProvisionedConcurrencyInvocations` conta il numero totale di chiamate, mentre `ProvisionedConcurrentExecutions` conta il numero di ambienti attivi. Per comprendere questa distinzione, esamina i seguenti scenari:



In questo esempio, supponiamo di ricevere 1 chiamata al minuto e che ogni chiamata richieda 2 minuti per essere completata. Ogni barra orizzontale arancione rappresenta una singola richiesta. Si supponga di allocare 10 unità di simultaneità fornita a questa funzione, in modo che ogni richiesta venga eseguita in base alla simultaneità fornita.

Tra i minuti 0 e 1, arriva la `Request 1`. Al minuto 1, il valore di MAX (`ProvisionedConcurrentExecutions`) è 1, poiché nell'ultimo minuto era attivo al massimo 1 ambiente di esecuzione. Anche il valore di SUM (`ProvisionedConcurrencyInvocations`) è 1, poiché nell'ultimo minuto è arrivata una nuova richiesta.

Tra i minuti 1 e 2, arriva la `Request 2` mentre la `Request 1` è ancora in esecuzione. Al minuto 2, il valore di MAX (`ProvisionedConcurrentExecutions`) è 2, poiché nell'ultimo minuto erano attivi al massimo 2 ambienti di esecuzione. Tuttavia, il valore di SUM (`ProvisionedConcurrencyInvocations`) è 1, poiché nell'ultimo minuto è arrivata solo una nuova richiesta. I parametri continuano a comportarsi in questo modo fino alla fine dell'esempio.

- `ProvisionedConcurrencySpilloverInvocations`: il numero di volte in cui Lambda richiama la funzione con la simultaneità (riservata o fornita) standard quando è in uso tutta la simultaneità fornita. Lambda emette questo parametro per ogni versione e alias di funzione per cui sia configurata la simultaneità fornita. Visualizza questo parametro utilizzando SUM. Il valore di `ProvisionedConcurrencyInvocations` + `ProvisionedConcurrencySpilloverInvocations` deve essere uguale al numero totale di chiamate delle funzioni (cioè il parametro `Invocations`).

  `ProvisionedConcurrencyUtilization`: la percentuale di simultaneità fornita in uso (ossia il valore di `ProvisionedConcurrentExecutions` diviso per la quantità totale di simultaneità fornita allocata). Lambda emette questo parametro per ogni versione e alias di funzione per cui sia configurata la simultaneità fornita. Visualizza questo parametro utilizzando MAX.

Ad esempio, si supponga di allocare 100 unità di simultaneità fornita a una versione di funzione. In un dato minuto, se al massimo 60 di questi 100 ambienti di esecuzione gestivano le chiamate contemporaneamente, il valore di MAX (`ProvisionedConcurrentExecutions`) è 60 e il valore di MAX (`ProvisionedConcurrencyUtilization`) è 0,6.

Un valore elevato di `ProvisionedConcurrencySpilloverInvocations` può indicare che è necessario allocare simultaneità fornita aggiuntiva per la funzione. In alternativa, è possibile configurare Application Auto Scaling per gestire il dimensionamento automatico della simultaneità fornita in base a soglie predefinite.

Viceversa, valori costantemente bassi di `ProvisionedConcurrencyUtilization` possono indicare che hai allocato una quantità eccessiva di simultaneità fornita per la funzione.

Lavorare con il parametro `ClaimedAccountConcurrency`

Lambda utilizza il parametro `ClaimedAccountConcurrency` per determinare la quantità di simultaneità disponibile nell'account per le invocazioni on demand. Lambda calcola `ClaimedAccountConcurrency` utilizzando la formula seguente:

```
ClaimedAccountConcurrency = UnreservedConcurrentExecutions + (allocated concurrency)
```

`UnreservedConcurrentExecutions` corrisponde al numero di invocazioni simultanee attive che utilizzano la simultaneità non riservata. La simultaneità allocata è la somma delle due parti seguenti (sostituendo RC come "simultaneità riservata" e PC come "simultaneità fornita"):

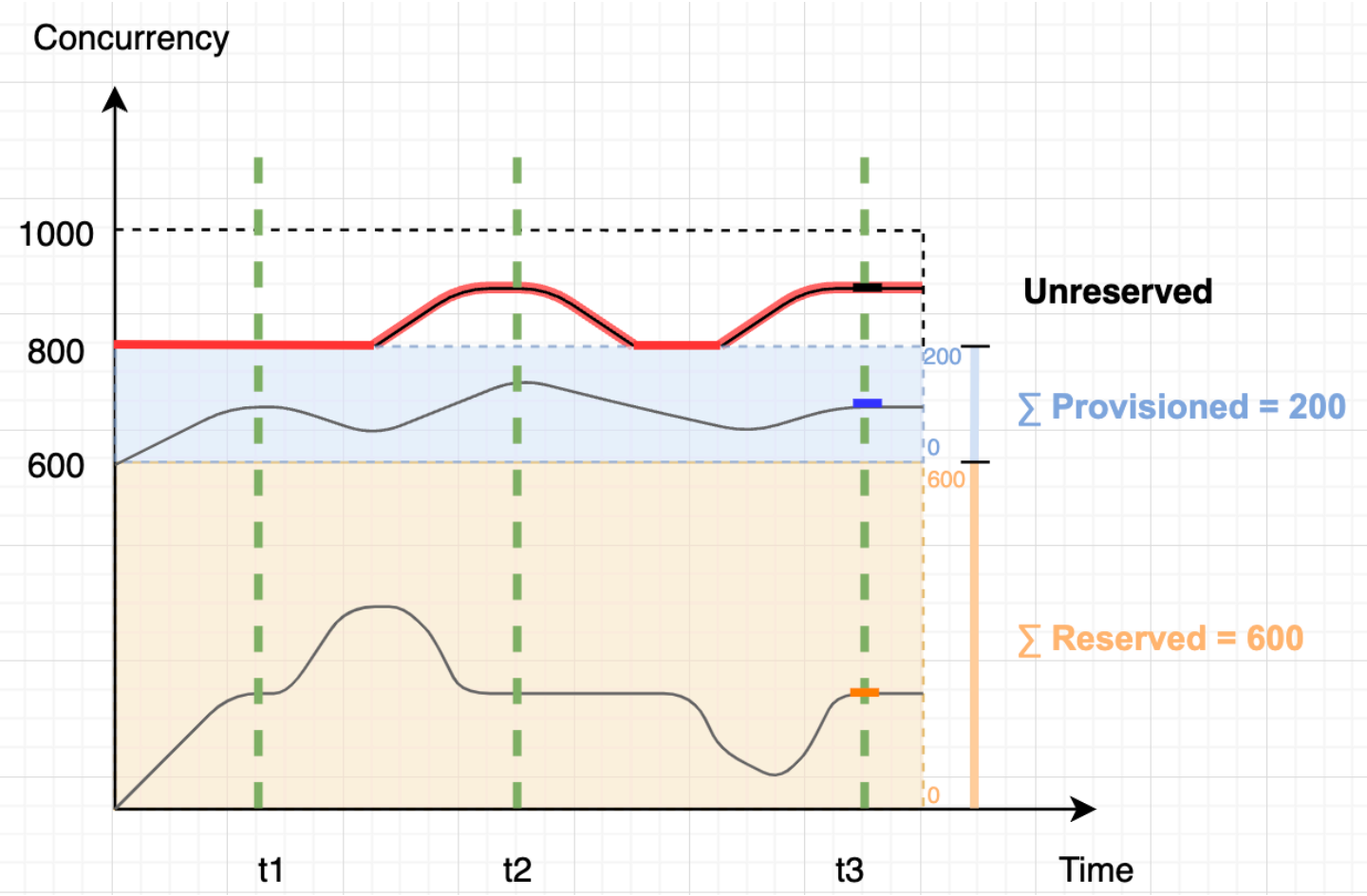
- Il valore RC totale in tutte le funzioni di una Regione.
- Il valore PC totale in tutte le funzioni di una Regione che utilizzanoPC, escluse le funzioni che utilizzano RC.

> **Note**
>
> Non puoi allocare più PC di RC per una funzione. Pertanto, il valore RC di una funzione è sempre maggiore o uguale al suo valore PC. Per calcolare il contributo alla simultaneità allocata per tali funzioni con sia PC che RC, Lambda considera solo RC, che è il valore massimo tra i due.

Per determinare la quantità di simultaneità disponibile per le invocazioni on demand, Lambda utilizza il parametro `ClaimedAccountConcurrency` anziché `ConcurrentExecutions`. Sebbene sia utile per tenere traccia del numero di invocazioni simultanee attive, il parametro `ConcurrentExecutions` non sempre riflette la disponibilità di simultaneità effettiva. Questo perché Lambda considera anche la simultaneità riservata e la simultaneità fornita per determinare la disponibilità.

Per illustrare il funzionamento di `ClaimedAccountConcurrency`, immaginiamo uno scenario in cui configuri una grande quantità di simultaneità riservata e simultaneità fornita nelle funzioni che rimane per buona parte inutilizzata. Nell'esempio seguente, supponiamo che il limite di simultaneità dell'account sia 1.000 e che vi siano due funzioni principali nell'account: `function-orange` e `function-blue`. Assegni 600 unità di simultaneità riservata per `function-orange`. Assegni 200

unità di simultaneità fornita per `function-blue`. Supponiamo che, nel corso del tempo, implementi ulteriori funzioni e osservi lo schema di traffico seguente:



Nel diagramma precedente, le linee nere indicano l'uso effettivo della simultaneità nel tempo, mentre la linea rossa indica il valore di `ClaimedAccountConcurrency` nel tempo. In tutto lo scenario, il valore minimo di `ClaimedAccountConcurrency` è 800, nonostante lo scarso utilizzo effettivo di simultaneità nelle funzioni. Questo perché hai assegnato 800 unità totali di simultaneità per `function-orange` e `function-blue`. Dal punto di vista di Lambda, hai "rivendicato" questa simultaneità per l'uso, quindi ti rimangono in effetti solo 200 unità di simultaneità per le altre funzioni.

In questo scenario, la concorrenza allocata è 800 nella formula `ClaimedAccountConcurrency`. Possiamo quindi ricavare il valore di `ClaimedAccountConcurrency` in corrispondenza di vari punti del diagramma:

- In `t1`, `ClaimedAccountConcurrency` è 800 (800 + 0 `UnreservedConcurrentExecutions`).
- In `t2`, `ClaimedAccountConcurrency` è 900 (800 + 100 `UnreservedConcurrentExecutions`).

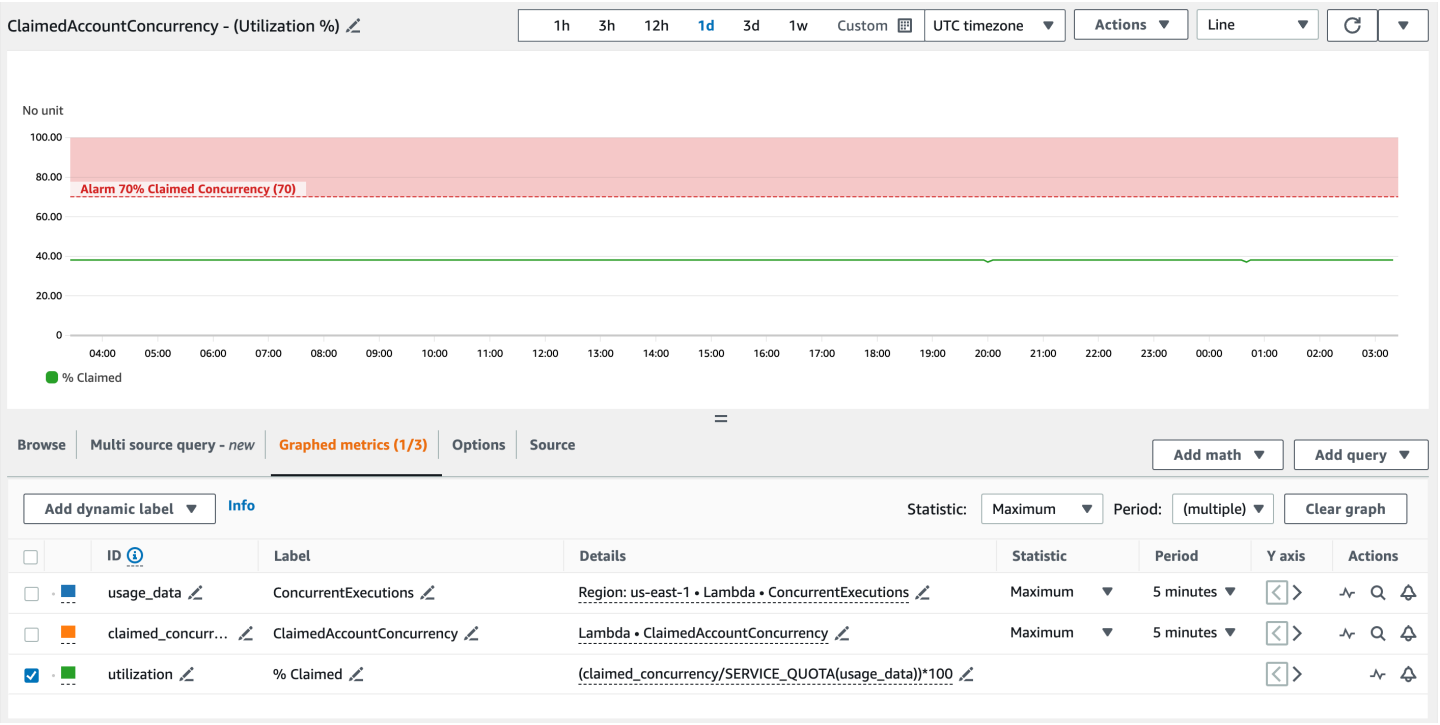
- In `t3`, `ClaimedAccountConcurrency` è di nuovo 900 (800 + 100 `UnreservedConcurrentExecutions`).

Configurazione del parametro **`ClaimedAccountConcurrency`** in CloudWatch

Lambda emette il parametro `ClaimedAccountConcurrency` in CloudWatch. Utilizza questo parametro insieme al valore di `SERVICE_QUOTA(ConcurrentExecutions)` per ottenere la percentuale di utilizzo della simultaneità nel tuo account, come mostrato nella formula seguente:

```
Utilization = (ClaimedAccountConcurrency/SERVICE_QUOTA(ConcurrentExecutions)) * 100%
```

La schermata seguente illustra come è possibile rappresentare graficamente la formula in CloudWatch. La linea `claim_utilization` verde rappresenta l'utilizzo della simultaneità nell'account, che è pari a circa il 40%:



La schermata precedente include anche un allarme CloudWatch che entra nello stato `ALARM` quando l'utilizzo della simultaneità supera il 70%. Puoi utilizzare il parametro `ClaimedAccountConcurrency` insieme ad allarmi simili per determinare in modo proattivo quando potrebbe essere necessario richiedere un limite di simultaneità dell'account più elevato.

Compilazione di funzioni Lambda con Node.js

Puoi eseguire il JavaScript codice con Node.js in. AWS Lambda Lambda fornisce Runtime per Node.js che eseguono il tuo codice per elaborare gli eventi. Il codice viene eseguito in un ambiente che include AWS SDK per JavaScript, con le credenziali di un ruolo AWS Identity and Access Management (IAM) gestito dall'utente. Per ulteriori informazioni sulle versioni SDK incluse nei runtime di Node.js, consulta the section called “Versioni SDK incluse nel runtime”.

Lambda supporta i seguenti runtime di Node.js.

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Node.js 24	`nodejs24.x`	Amazon Linux 2023	30 aprile 2028	1 giugno 2028	1 luglio 2028
Node.js 22	`nodejs22.x`	Amazon Linux 2023	30 aprile 2027	1 giugno 2027	1 luglio 2027
Node.js 20	`nodejs20.x`	Amazon Linux 2023	30 aprile 2026	31 agosto 2026	30 settembre 2026

Per creare una funzione Node.js.

1. Aprire la console Lambda.
2. Scegli Crea funzione.
3. Configura le impostazioni seguenti:
   - Nome della funzione: inserisci il nome della funzione.
   - Runtime: scegli Node.js 24.x.
4. Scegli Crea funzione.

La console crea una funzione Lambda con un singolo file di origine denominato `index.mjs`. È possibile modificare questo file e aggiungere altri file nell'editor di codice predefinito. Nella sezione

DEPLOY, scegli Implementa per aggiornare il codice della tua funzione. Quindi, per eseguire il codice, scegli Crea evento di test nella sezione EVENTI DI TEST.

Il file `index.mjs` esporta una funzione denominata `handler` che richiede un oggetto evento e un oggetto contesto. Questa è la funzione del gestore chiamata da Lambda quando la funzione viene richiamata. Il runtime della funzione Node.js riceve gli eventi di chiamata da Lambda e li passa al gestore. Nella configurazione della funzione il valore del gestore è `index.handler`.

Quando si salva il codice funzione, la console Lambda crea un pacchetto di implementazione dell'archivio di file .zip. Quando sviluppi il codice funzione al di fuori della console (utilizzando un IDE) devi creare un pacchetto di implementazione per caricare il codice nella funzione Lambda.

Il runtime della funzione passa un oggetto contesto al gestore, oltre all'evento di chiamata. L'oggetto contesto contiene ulteriori informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. Altre informazioni sono disponibili con le variabili di ambiente.

La funzione Lambda include un gruppo di CloudWatch log Logs. Il runtime della funzione invia i dettagli su ogni chiamata a Logs. CloudWatch Si trasmette qualsiasi log che la tua funzione emette durante la chiamata. Se la funzione restituisce un errore, Lambda formatta l'errore e lo restituisce al chiamante.

Argomenti

- Versioni SDK incluse nel runtime
- Utilizzo di keep-alive per le connessioni TCP
- Caricamento di certificati CA
- Funzionalità sperimentali di Node.js
- Definire l'handler delle funzioni Lambda in Node.js
- Distribuisci funzioni Lambda in Node.js con archivi di file .zip
- Distribuisci funzioni Lambda in Node.js con immagini di container
- Utilizzo dei livelli per le funzioni Lambda Node.js
- Utilizzo dell'oggetto di contesto Lambda per recuperare le informazioni sulla funzione Node.js
- Registrare e monitorare funzioni Lambda in Node.js
- Strumentazione del codice Node.js in AWS Lambda

Versioni SDK incluse nel runtime

Tutti i runtime Lambda Node.js supportati includono una versione secondaria specifica della AWS SDK per JavaScript v3, non la versione più recente. La versione secondaria specifica inclusa nel runtime dipende dalla versione di runtime e dalla tua. Regione AWS Per trovare la versione specifica dell'SDK inclusa nel runtime che stai utilizzando, crea una funzione Lambda con il codice seguente.

Example index.mjs

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

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

Restituisce una risposta nel seguente formato:

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

Per ulteriori informazioni, consulta Utilizzo dell'SDK per JavaScript v3 nel gestore.

Utilizzo di keep-alive per le connessioni TCP

L' HTTP/HTTPS agente Node.js predefinito crea una nuova connessione TCP per ogni nuova richiesta. Per evitare il costo della creazione di nuove connessioni, keep-alive è abilitato per impostazione predefinita in tutti i runtime Node.js supportati. Il keep-alive può ridurre i tempi di richiesta per le funzioni Lambda che effettuano più chiamate API utilizzando l'SDK.

Per disabilitare keep-alive, consulta Riutilizzo delle connessioni con keep-alive in Node.js nella Guida per sviluppatori SDK per 3.x.AWS JavaScript Per ulteriori informazioni sull'utilizzo di keep-alive, consulta HTTP keep-alive è attivo di default nell'SDK modulare o sul blog Developer Tools. AWS JavaScript AWS

Caricamento di certificati CA

Per le versioni di runtime di Node.js fino a Node.js 18, Lambda carica automaticamente i certificati CA (autorità di certificazione) specifici di Amazon per semplificare la creazione di funzioni che interagiscono con altri Servizi AWS. Ad esempio, Lambda include i certificati Amazon RDS necessari

per convalidare il certificato di identità del server installato sul tuo database Amazon RDS. Questo comportamento può avere un impatto sulle prestazioni durante gli avvii a freddo.

A partire da Node.js 20, Lambda non carica più certificati CA aggiuntivi per impostazione predefinita. Il runtime Node.js 20 contiene un file di certificato con tutti i certificati Amazon CA nella posizione `/var/runtime/ca-cert.pem`. Per ripristinare lo stesso comportamento da Node.js 18 e runtime precedenti, imposta la variabile di ambiente `NODE_EXTRA_CA_CERTS` su `/var/runtime/ca-cert.pem`.

Per prestazioni ottimali, consigliamo di raggruppare nel pacchetto di implementazione solo i certificati necessari e di caricarli tramite la variabile di ambiente `NODE_EXTRA_CA_CERTS`. Il file dei certificati deve essere composto da uno o più certificati CA root o intermedi affidabili in formato PEM. Ad esempio, per RDS, includi i certificati richiesti insieme al codice come `certificates/rds.pem`. Quindi, carica i certificati impostando `NODE_EXTRA_CA_CERTS` su `/var/task/certificates/rds.pem`.

Funzionalità sperimentali di Node.js

Le versioni originali del linguaggio Node.js abilitano alcune funzionalità sperimentali per impostazione predefinita. Lambda disabilita queste funzionalità per garantire stabilità di runtime e prestazioni costanti. La tabella seguente elenca le funzionalità sperimentali che Lambda disabilita.

Funzionalità sperimentale	Versioni supportate di Node.js	Bandiera Node.js applicata da Lambda	Bandiera Lambda da riattivare
Support per l'importazione di moduli utilizzando require in ES modules	Node.js 20, Node.js 22	`--no-experimental-require-module`	`--experimental-require-module`
Support per il rilevamento automatico dei moduli ES vs CommonJS	Node.js 22	`--no-experimental-detect-module`	`--experimental-detect-module`

Per abilitare una funzionalità sperimentale disabilitata, imposta il flag re-enable nella variabile di ambiente. `NODE_OPTIONS` Ad esempio, per abilitare il modulo ES require support, imposta

su`NODE_OPTIONS`. `--experimental-require-module` Lambda rileva questo override e rimuove il flag di disabilitazione corrispondente.

> **Important**
>
> L'utilizzo di funzionalità sperimentali può portare a problemi di instabilità e prestazioni. Queste funzionalità potrebbero essere modificate o rimosse nelle future versioni di Node.js. Le funzioni che utilizzano funzionalità sperimentali non sono idonee per il Lambda Service Level Agreement (SLA) o. Supporto AWS

Definire l'handler delle funzioni Lambda in Node.js

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Questa pagina descrive come utilizzare i gestori di funzioni Lambda in Node.js, incluse le opzioni per la configurazione del progetto, le convenzioni di denominazione e le migliori pratiche. Questa pagina include anche un esempio di funzione Lambda di Node.js che raccoglie informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon Simple Storage Service (Amazon S3). Per informazioni su come distribuire una funzione dopo averla scritta, consulta o. the section called “Implementazione di archivi di file .zip” the section called “Implementazione di immagini di container”

Argomenti

- Configurazione del progetto Node.js handler
- Esempio di codice della funzione Lambda Node.js
- Moduli CommonJS ed ES
- Inizializzazione di Node.js
- Convenzioni di denominazione dei gestori
- Definizione e accesso all'oggetto evento di input
- Modelli di gestione validi per le funzioni Node.js
- Utilizzo dell'SDK per JavaScript v3 nel gestore
- Accesso alle variabili d'ambiente
- Utilizzo dello stato globale
- Best practice di codice per funzioni Lambda in Node.js

Configurazione del progetto Node.js handler

Esistono diversi modi per inizializzare un progetto Lambda Node.js. Ad esempio, è possibile creare un progetto Node.js standard utilizzandonpm, creare un'AWS SAM applicazione o creare un'AWS CDK applicazione.

Per creare il progetto utilizzandonpm:

```
npm init
```

Questo comando inizializza il progetto e genera un `package.json` file che gestisce i metadati e le dipendenze del progetto.

Il codice della funzione risiede in un file or. `.js` `.mjs` JavaScript Nell'esempio seguente, diamo un nome a questo file `index.mjs` perché utilizza un gestore di moduli ES. Lambda supporta sia il modulo ES che i gestori CommonJS. Per ulteriori informazioni, consulta Moduli CommonJS ed ES.

Un tipico progetto di funzione Lambda di Node.js segue questa struttura generale:

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

Esempio di codice della funzione Lambda Node.js

Il seguente codice di funzione Lambda di esempio raccoglie le informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon S3.

Example index.mjs, funzione 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}`);
    }
}
```

Questo file `index.mjs` contiene le sezioni seguenti:

- `import`block: usa questo blocco per includere le librerie richieste dalla tua funzione Lambda, come i client AWS SDK.
- `const s3Client`dichiarazione: inizializza un client Amazon S3 al di fuori della funzione di gestione. Ciò fa sì che Lambda esegua questo codice durante la fase di inizializzazione e il client viene preservato per il riutilizzo su più chiamate.
- JSDoc blocco di commenti: definisci i tipi di input e output per il tuo gestore utilizzando le annotazioni. JSDoc
- `export const handler`: Questa è la funzione di gestione principale richiamata da Lambda. Quando distribuisci la funzione, specifica `index.handler` la proprietà Handler. Il valore della `Handler` proprietà è il nome del file e il nome del metodo del gestore esportato, separati da un punto.
- `uploadReceiptToS3`funzione: Questa è una funzione di supporto a cui fa riferimento la funzione di gestione principale.

Affinché questa funzione funzioni correttamente, il suo ruolo di esecuzione deve consentire l'azione. `s3:PutObject` Inoltre, assicuratevi di definire la variabile di `RECEIPT_BUCKET` ambiente. Dopo una chiamata riuscita, il bucket Amazon S3 dovrebbe contenere un file di ricevuta.

Moduli CommonJS ed ES

Node.js supporta due sistemi di moduli: CommonJS e ECMAScript modules (moduli ES). Lambda consiglia di utilizzare i moduli ES in quanto supportano l'attesa di primo livello, che consente il completamento di attività asincrone durante l'inizializzazione dell'ambiente di esecuzione.

Node.js tratta i file con estensione come moduli CommonJS mentre un'`.cjs`estensione indica i moduli ES. `.mjs` Per impostazione predefinita, Node.js tratta i file con l'estensione del nome `.js` file come moduli CommonJS. È possibile configurare Node.js per trattare `.js` i file come moduli ES specificando l'`type`as `module` nel file della funzione. `package.json` Puoi configurare Node.js in Lambda per rilevare automaticamente se un `.js` file deve essere trattato come CommonJS o come modulo ES aggiungendo il `–experimental-detect-module` flag alla variabile di ambiente. `NODE_OPTIONS` Per ulteriori informazioni, consulta le funzionalità di Experimental Node.js.

Gli esempi seguenti mostrano gestori di funzioni scritti utilizzando sia i moduli ES che i moduli CommonJS. Gli esempi rimanenti in questa pagina utilizzano tutti i moduli ES.

ES module example

Example — Gestore del modulo 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 — Gestore del modulo 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;
};
```

Inizializzazione di Node.js

Node.js utilizza un I/O modello non bloccante che supporta operazioni asincrone efficienti utilizzando un ciclo di eventi. Ad esempio, se Node.js effettua una chiamata di rete, la funzione continua a elaborare altre operazioni senza bloccare una risposta di rete. Quando viene ricevuta, la risposta di rete viene inserita nella coda di callback. Le attività della coda vengono elaborate al termine dell'operazione corrente.

Lambda consiglia di utilizzare l'await di primo livello in modo che le attività asincrone avviate durante l'inizializzazione dell'ambiente di esecuzione vengano completate durante l'inizializzazione. Le attività asincrone che non vengono completate durante l'inizializzazione vengono in genere eseguite durante la prima chiamata della funzione. Ciò può causare comportamenti o errori imprevisti. Ad esempio, l'inizializzazione della funzione può effettuare una chiamata di rete per recuperare un AWS parametro da Parameter Store. Se questa attività non viene completata durante l'inizializzazione, il valore può essere nullo durante una chiamata. Può inoltre verificarsi un ritardo tra l'inizializzazione e l'invoca, che può causare errori nelle operazioni che richiedono un intervallo di tempo. In particolare, le chiamate AWS di servizio possono basarsi su firme di richiesta sensibili al fattore tempo, con conseguenti errori nelle chiamate di servizio se la chiamata non viene completata durante la fase di inizializzazione. Il completamento delle attività durante l'inizializzazione in genere migliora le prestazioni di avvio a freddo e richiama innanzitutto le prestazioni quando si utilizza Provisioned Concurrency. Per ulteriori informazioni, consulta il nostro post sul blog Using Node.js ES modules and top-level await in. AWS Lambda

Convenzioni di denominazione dei gestori

Quando si configura una funzione, il valore dell'impostazione Handler è il nome del file e il nome del metodo del gestore esportato, separati da un punto. L'impostazione predefinita per le funzioni create nella console e negli esempi in questa guida è `index.handler`. Questo indica il metodo `handler` che viene esportato dal file `index.js` o `index.mjs`.

Se si crea una funzione nella console utilizzando un nome di file o un nome del gestore di funzione diverso, è necessario modificare il nome del gestore predefinito.

Modifica del nome del gestore funzioni (console)

1. Apri la pagina Funzioni della console Lambda e scegli la tua funzione.
2. Scegli la scheda Codice.
3. Scorri verso il basso fino al riquadro Impostazioni di runtime e scegli Modifica.

4. In Gestore, inserisci il nuovo nome per il tuo gestore di funzioni.
5. Scegli Save (Salva).

Definizione e accesso all'oggetto evento di input

JSON è il formato di input più comune e standard per le funzioni Lambda. In questo esempio, la funzione prevede un input simile a quanto segue:

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

Quando si utilizzano le funzioni Lambda in Node.js, è possibile definire la forma prevista dell'evento di input utilizzando JSDoc le annotazioni. In questo esempio, definiamo la struttura di input nel commento del gestore: JSDoc

```
/**
 * 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
 */
```

Dopo aver definito questi tipi nel JSDoc commento, potete accedere ai campi dell'oggetto evento direttamente nel codice. Ad esempio, `event.order_id` recupera il valore di `order_id` dall'input originale.

Modelli di gestione validi per le funzioni Node.js

Si consiglia di utilizzare async/await per dichiarare il gestore della funzione invece di utilizzare i callback. Async/await è un modo conciso e leggibile per scrivere codice asincrono, senza la necessità di callback annidati o di concatenare promesse. Con async/await puoi scrivere codice che si legga come codice sincrono, pur rimanendo asincrono e privo di blocchi.

gestori di funzioni asincrone (consigliati)

La parola chiave `async` contrassegna una funzione come asincrona, mentre la parola chiave `await` mette in pausa l'esecuzione della funzione fino alla risoluzione di `Promise`. Il gestore accetta i seguenti argomenti:

- `event`: contiene i dati di input passati alla funzione.
- `context`: contiene informazioni sull'invocazione, sulla funzione e sull'ambiente di esecuzione. Per ulteriori informazioni, consulta Utilizzo dell'oggetto di contesto Lambda per recuperare le informazioni sulla funzione Node.js.

Ecco le firme valide per il pattern: async/await

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

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

Gestori di funzioni sincroni

Se la funzione non esegue attività asincrone, è possibile utilizzare un gestore di funzioni sincrono, utilizzando una delle seguenti firme di funzione:

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

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

Gestori di funzioni di streaming delle risposte

Lambda supporta lo streaming delle risposte con Node.js. I gestori delle funzioni di streaming delle risposte utilizzano il `awslambda.streamifyResponse()` decoratore e accettano 3 parametri:`event`, e. `responseStream` `context` La firma della funzione è:

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

Per ulteriori informazioni, consulta Streaming di risposte per le funzioni Lambda.

gestori di funzioni basati su callback

> **Note**
>
> I gestori di funzioni basati su callback sono supportati solo fino a Node.js 22. A partire da Node.js 24, le attività asincrone devono essere implementate utilizzando gestori di funzioni asincrone.

I gestori di funzioni basati sul callback devono utilizzare gli argomenti event, context e callback. Esempio:

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

La funzione di callback prevede una risposta e una, che deve essere serializzabile in `Error` formato JSON. La funzione continua a essere eseguita fino a quando il ciclo di eventi non è vuoto o fino al timeout della funzione. La risposta non viene inviata all'invoker finché tutte le attività del ciclo di eventi non giungono a termine. Se la funzione va in timeout, viene invece restituito un errore. È possibile configurare il runtime per inviare immediatamente la risposta impostando il contesto. callbackWaitsForEmptyEventLoopa false.

Example – richiesta HTTP con callback

La seguente funzione di esempio controlla un URL e restituisce il codice di stato all'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));
  });
};
```

Utilizzo dell'SDK per JavaScript v3 nel gestore

Spesso, utilizzerai le funzioni Lambda per interagire o aggiornare altre AWS risorse. Il modo più semplice per interfacciarsi con queste risorse consiste nell'utilizzare il AWS SDK per JavaScript.

Tutti i runtime Lambda Node.js supportati includono l'SDK per la versione 3. JavaScript Tuttavia, consigliamo vivamente di includere i client AWS SDK necessari nel pacchetto di distribuzione. Ciò massimizza la compatibilità con le versioni precedenti durante i futuri aggiornamenti del runtime Lambda. Affidati all'SDK fornito in fase di esecuzione solo quando non puoi includere pacchetti aggiuntivi (ad esempio, quando usi l'editor di codice della console Lambda o il codice in linea in un modello). AWS CloudFormation

Per aggiungere dipendenze SDK alla tua funzione, usa il `npm install` comando per i client SDK specifici di cui hai bisogno. Nel codice di esempio, abbiamo usato il client Amazon S3. Aggiungi questa dipendenza eseguendo il comando seguente nella directory che contiene il `package.json` file:

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

Nel codice della funzione, importate il client e i comandi necessari, come dimostra la funzione di esempio:

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

Quindi, inizializza un client Amazon S3:

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

In questo esempio, abbiamo inizializzato il nostro client Amazon S3 all'esterno della funzione del gestore principale per evitare di doverlo inizializzare ogni volta che invochiamo la nostra funzione. Dopo aver inizializzato il client SDK, puoi utilizzarlo per effettuare chiamate API per quel servizio. AWS Il codice di esempio richiama l'azione dell'PutObjectAPI Amazon S3 nel modo seguente:

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

Accesso alle variabili d'ambiente

Nel codice del gestore, puoi fare riferimento a qualsiasi variabile di ambiente utilizzando. `process.env` In questo esempio, facciamo riferimento alla variabile di ambiente `RECEIPT_BUCKET` definita utilizzando le seguenti righe di codice:

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

Utilizzo dello stato globale

Lambda esegue il codice statico durante la fase di inizializzazione prima di richiamare la funzione per la prima volta. Le risorse create durante l'inizializzazione restano in memoria tra le invocazioni, in modo da evitare di doverle creare ogni volta che si invoca la funzione.

Nel codice di esempio, il codice di inizializzazione del client S3 è esterno al gestore. Il runtime inizializza il client prima che la funzione gestisca il primo evento e il client rimane disponibile per il riutilizzo in tutte le chiamate.

Best practice di codice per funzioni Lambda in Node.js

Segui queste linee guida durante la creazione di funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. In questo modo è possibile creare una funzione di cui è più semplice eseguire l'unit test.
- Controllare le dipendenze nel pacchetto di distribuzione della funzione. L'ambiente di AWS Lambda esecuzione contiene diverse librerie. Per i runtime Node.js e Python, questi includono. AWS SDKs Per abilitare il set di caratteristiche e aggiornamenti della sicurezza più recenti, Lambda aggiorna periodicamente tali librerie. Tali aggiornamenti possono introdurre lievi modifiche al comportamento della funzione Lambda. Per mantenere il controllo completo delle dipendenze utilizzate dalla funzione, inserire tutte le dipendenze nel pacchetto di implementazione.
- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del contesto di esecuzione.
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. In questo modo viene ridotta la quantità di tempo necessaria per il download del pacchetto e per la relativa decompressione prima dell'invocazione.

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive

elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedere Riutilizzo delle connessioni con Keep-Alive in Node.js.

Utilizzare le variabili di ambiente per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su `0` per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare documenti non documentati e non pubblici APIs nel codice della funzione Lambda. Per i runtime AWS Lambda gestiti, Lambda applica periodicamente aggiornamenti di sicurezza e funzionalità all'interno di Lambda. APIs Questi aggiornamenti delle API interne possono essere incompatibili con le versioni precedenti e portare a conseguenze indesiderate, come errori di chiamata se la funzione dipende da questi elementi non pubblici. APIs Consulta il riferimento alle API per un elenco di quelle disponibili al pubblico. APIs

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda?.

Distribuisci funzioni Lambda in Node.js con archivi di file .zip

Il codice della AWS Lambda funzione comprende un file .js o .mjs contenente il codice del gestore della funzione, insieme a tutti i pacchetti e moduli aggiuntivi da cui dipende il codice. Per implementare questo codice della funzione in Lambda, utilizza un pacchetto di implementazione. Questo pacchetto può essere un archivio di file .zip o un'immagine di container. Per ulteriori informazioni sull'uso delle immagini di container con Node.js, consulta la pagina Implementazione di funzioni Lambda in Node.js con immagini di container.

Per creare un pacchetto di implementazione come archivio di file .zip, puoi utilizzare l'utilità di archiviazione di file .zip incorporata del tuo strumento della linea di comando o qualsiasi altra utilità file .zip, come ad esempio 7zip. Gli esempi mostrati nelle sezioni seguenti presuppongono che tu stia utilizzando uno strumento della linea di comando `zip` in un ambiente Linux o MacOS. Per utilizzare gli stessi comandi in Windows, puoi installare il sottosistema Windows per Linux per ottenere una versione di Ubuntu e Bash integrata con Windows.

Nota che Lambda utilizza le autorizzazioni dei file POSIX, quindi potresti aver bisogno di impostare le autorizzazioni per la cartella del pacchetto di implementazione prima di creare l'archivio di file .zip.

Argomenti

- Dipendenze di runtime in Node.js
- Creazione di un pacchetto di implementazione .zip senza dipendenze
- Creazione di un pacchetto di implementazione .zip con dipendenze
- Creazione di un livello Node.js per dipendenze
- Percorso di ricerca delle dipendenze e librerie incluse nel runtime
- Creazione e aggiornamento delle funzioni Lambda di Node.js utilizzando file .zip

Dipendenze di runtime in Node.js

Per le funzioni Lambda che utilizzano il runtime di Node.js, una dipendenza può essere qualsiasi modulo Node.js. Il runtime di Node.js include una serie di librerie comuni, oltre a una versione di AWS SDK per JavaScript. Tutti i runtime Node.js supportati includono la versione 3 dell'SDK. Per utilizzare la versione 2 dell'SDK, è necessario aggiungere l'SDK al pacchetto di distribuzione del file.zip. Per trovare la versione specifica dell'SDK inclusa nel runtime che stai utilizzando, consulta. the section called “Versioni SDK incluse nel runtime”

Lambda aggiorna periodicamente le librerie SDK nel runtime di Node.js per includere le funzionalità e gli aggiornamenti di sicurezza più recenti. Lambda applica anche patch di sicurezza e aggiornamenti alle altre librerie incluse nel runtime. Per avere il pieno controllo delle dipendenze del pacchetto, puoi aggiungere la tua versione preferita di qualsiasi dipendenza inclusa nel runtime al tuo pacchetto di implementazione. Ad esempio, se desideri utilizzare una versione particolare dell'SDK per JavaScript, puoi includerla nel tuo file.zip come dipendenza. Per ulteriori informazioni sull'aggiunta di dipendenze incluse nel runtime al tuo file .zip, consulta la pagina Percorso di ricerca delle dipendenze e librerie incluse nel runtime.

In base al modello di responsabilità condivisa di AWS, è tua responsabilità gestire eventuali dipendenze nei pacchetti di implementazione delle tue funzioni. Ciò include l'applicazione di aggiornamenti e patch di sicurezza. Per aggiornare le dipendenze nel pacchetto di implementazione della funzione, crea prima un nuovo file .zip e poi caricalo su Lambda. Per ulteriori informazioni, consulta Creazione di un pacchetto di implementazione .zip con dipendenze e Creazione e aggiornamento delle funzioni Lambda di Node.js utilizzando file .zip.

Creazione di un pacchetto di implementazione .zip senza dipendenze

Se il codice della funzione non ha dipendenze oltre alle librerie incluse nel runtime Lambda, il file .zip contiene solo il file `index.js` o `index.mjs` con il codice del gestore della funzione. Utilizza il tuo strumento di compressione preferito per creare un file .zip con il file `index.js` o `index.mjs` nella directory principale. Se il file contenente il codice del gestore della funzione non si trova nella directory principale del file .zip, Lambda non è in grado di eseguire il codice.

Per informazioni su come implementare il file .zip per creare una nuova funzione Lambda o aggiornarne una esistente, consulta la sezione Creazione e aggiornamento delle funzioni Lambda di Node.js utilizzando file .zip.

Creazione di un pacchetto di implementazione .zip con dipendenze

Se il codice della funzione dipende da pacchetti o moduli non inclusi nel runtime Node.js di Lambda, puoi aggiungere queste dipendenze al file .zip con il codice della funzione oppure utilizzare un livello Lambda. Le istruzioni in questa sezione mostrano come includere le dipendenze nel pacchetto di implementazione .zip. Per istruzioni sull'inclusione di dipendenze in un livello, consulta the section called “Creazione di un livello Node.js per dipendenze”.

I seguenti comandi della CLI di esempio creano un file .zip denominato `my_deployment_package.zip` contenente il file `index.js` o `index.mjs` con il codice del

gestore della funzione e le relative dipendenze. Nell'esempio, installi le dipendenze utilizzando il gestore di pacchetti npm.

Per creare il pacchetto di implementazione

1. Passa alla directory del progetto contenente il file del codice sorgente `index.js` o `index.mjs`. In questo esempio, la directory è denominata `my_function`.

   ```
   cd my_function
   ```

2. Installa le librerie richieste dalla tua funzione nella directory `node_modules` utilizzando il comando `npm install`. In questo esempio installi l' SDK AWS X-Ray per Node.js.

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

   Questo crea una struttura di cartelle simile alla seguente:

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

   Al tuo pacchetto di implementazione puoi anche aggiungere moduli personalizzati che crei in autonomia. Crea una directory in `node_modules` con il nome del tuo modulo e salva lì i tuoi pacchetti personalizzati.

3. Crea un file .zip dei contenuti della cartella di progetto della directory principale. Utilizza l'opzione (ricorsiva) `r` per garantire che lo zip comprima le sottocartelle.

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

Creazione di un livello Node.js per dipendenze

Le istruzioni in questa sezione spiegano come includere dipendenze in un livello. Per istruzioni sull'inclusione di dipendenze in un pacchetto di implementazione, consulta the section called “Creazione di un pacchetto di implementazione .zip con dipendenze”.

Quando si aggiunge un livello a una funzione, Lambda carica il contenuto del livello nella directory `/opt` di quell'ambiente di esecuzione. Per ogni runtime Lambda, la variabile `PATH` include percorsi di cartelle specifici nella directory `/opt`. Per garantire che Lambda raccolga il contenuto del layer, il file.zip del layer deve avere le sue dipendenze in uno dei seguenti percorsi di cartella:

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

Ad esempio, la struttura del file .zip del livello potrebbe essere simile alla seguente:

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

Lambda rileva automaticamente tutte le librerie nella directory `/opt/lib` e tutti i file binari nella directory `/opt/bin`. Per accertarti che Lambda trovi correttamente il contenuto del tuo livello, crea un livello con la seguente struttura:

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

Dopo aver creato un pacchetto del livello, consulta the section called “Creazione ed eliminazione di livelli” e the section called “Aggiunta di livelli” per completare l'impostazione del livello.

Percorso di ricerca delle dipendenze e librerie incluse nel runtime

Il runtime di Node.js include una serie di librerie comuni, oltre a una versione di AWS SDK per JavaScript. Se desideri utilizzare una versione diversa di una libreria inclusa nel runtime, puoi farlo raggruppandola con la tua funzione o aggiungendola come dipendenza nel tuo pacchetto di implementazione. Ad esempio, puoi utilizzare una versione diversa dell'SDK aggiungendola al tuo pacchetto di implementazione .zip. Puoi anche includerlo in un livello Lambda per la tua funzione.

Quando utilizzi un'istruzione `import` o `require` nel codice, il runtime di Node.js cerca nelle directory nel percorso `NODE_PATH` finché non trova il modulo. Per impostazione predefinita, la prima posizione cercata dal runtime è la directory in cui il pacchetto di implementazione .zip viene decompresso e montato (`/var/task`). Se includi una versione di una libreria inclusa nel runtime nel tuo pacchetto di implementazione, questa versione avrà la precedenza sulla versione inclusa nel runtime. Le dipendenze nel pacchetto di implementazione hanno la precedenza anche sulle dipendenze nei livelli.

Quando aggiungi una dipendenza a un livello, Lambda la estrae in `/opt/nodejs/nodexx/node_modules`, dove `nodexx` rappresenta la versione del runtime che stai utilizzando. Nel percorso di ricerca, questa directory ha la precedenza sulla directory contenente le librerie incluse nel runtime (`/var/lang/lib/node_modules`). Le librerie nei livelli di funzione hanno quindi la precedenza sulle versioni incluse nel runtime.

Puoi visualizzare il percorso di ricerca completo per la tua funzione Lambda aggiungendo la seguente riga di codice.

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

Puoi anche aggiungere dipendenze in una cartella separata all'interno del tuo pacchetto .zip. Ad esempio, potresti aggiungere un modulo personalizzato a una cartella del tuo pacchetto .zip denominata `common`. Quando il pacchetto .zip viene decompresso e montato, questa cartella viene inserita nella directory `/var/task`. Per utilizzare una dipendenza da una cartella del pacchetto di implementazione .zip nel codice, utilizza un'istruzione `import { } from` o `const { } = require()`, a seconda che tu stia utilizzando la risoluzione del modulo CJS o ESM. Esempio:

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

Se raggruppi il codice con `esbuild`, `rollup` o qualcosa di simile, le dipendenze utilizzate dalla funzione vengono raggruppate in uno o più file. Si consiglia di utilizzare questo metodo per eliminare

le dipendenze ogni volta che è possibile. Rispetto all'aggiunta di dipendenze al pacchetto di distribuzione, il raggruppamento del codice consente di migliorare le prestazioni grazie alla riduzione delle operazioni. I/O

Creazione e aggiornamento delle funzioni Lambda di Node.js utilizzando file .zip

Dopo aver creato il pacchetto di implementazione .zip, puoi utilizzarlo per creare una nuova funzione Lambda o aggiornarne una esistente. Puoi distribuire il tuo pacchetto.zip utilizzando la console Lambda, l'API Lambda AWS Command Line Interface e l'API Lambda. Puoi anche creare e aggiornare le funzioni Lambda usando AWS Serverless Application Model (AWS SAM) e CloudFormation.

La dimensione massima per un pacchetto di implementazione .zip per Lambda è di 250 MB (dopo l'estrazione). Nota che questo limite si applica alla dimensione combinata di tutti i file caricati, inclusi eventuali livelli Lambda.

Il runtime Lambda necessita dell'autorizzazione per leggere i file nel pacchetto di distribuzione. Nella notazione ottale delle autorizzazioni Linux, Lambda richiede 644 permessi per i file non eseguibili (rw-r--r--) e 755 permessi () per le directory e i file eseguibili. rwxr-xr-x

In Linux e macOS, utilizza il comando `chmod` per modificare le autorizzazioni file su file e directory nel pacchetto di implementazione. Ad esempio, per assegnare a un file non eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 644 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta Set, View, Change, or Remove Permissions on an Object nella documentazione di Microsoft Windows.

> **Note**
>
> Se non concedi a Lambda le autorizzazioni necessarie per accedere alle directory nel pacchetto di distribuzione, Lambda imposta le autorizzazioni per tali directory su 755 (). rwxr-xr-x

Creazione e aggiornamento delle funzioni con file .zip utilizzando la console

Per creare una nuova funzione, devi prima creare la funzione nella console, quindi devi caricare il tuo archivio .zip. Per aggiornare una funzione esistente, apri la pagina relativa alla funzione, quindi segui la stessa procedura per aggiungere il file .zip aggiornato.

Se il file .zip ha dimensioni inferiori a 50 MB, è possibile creare o aggiornare una funzione caricando il file direttamente dal computer locale. Per i file .zip di dimensioni superiori a 50 MB, prima è necessario caricare il pacchetto in un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando Console di gestione AWS, consulta la Guida introduttiva ad Amazon S3. Per caricare file utilizzando la AWS CLI, consulta Move objects nella Guida per l'AWS CLI utente.

> **Note**
>
> Non è possibile modificare il tipo di pacchetto di implementazione (.zip o immagine di container) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine di container esistente per utilizzare un archivio di file .zip. È necessario creare una nuova funzione.

Creazione di una nuova funzione (console)

1. Apri la pagina Funzioni della console Lambda e scegli Crea funzione.
2. Scegli Author from scratch (Crea da zero).
3. In Basic information (Informazioni di base) eseguire queste operazioni:
   a. In Nome funzione, inserisci il nome della funzione.
   b. Per Runtime, seleziona il runtime che desideri utilizzare.
   c. (Facoltativo) Per Architettura, scegli l'architettura del set di istruzioni per la funzione. L'architettura predefinita è x86_64. Assicurati che il pacchetto di implementazione per la tua funzione sia compatibile con l'architettura del set di istruzioni scelta.
4. (Opzionale) In Autorizzazioni espandere Modifica ruolo di esecuzione predefinito. Puoi creare un nuovo ruolo di esecuzione o utilizzare un ruolo esistente.
5. Scegli Crea funzione. Lambda crea una funzione di base "Hello world" utilizzando il runtime scelto.

Caricamento di un archivio .zip dal computer locale (console)

1. Nella pagina Funzioni della console Lambda, scegli la funzione per cui vuoi caricare il file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli File .zip.
5. Per caricare il file .zip, procedi come segue:
    a. Seleziona Carica, quindi seleziona il tuo file .zip nel selettore di file.
    b. Seleziona Apri.
    c. Scegli Save (Salva).

Caricamento di un archivio .zip da un bucket Amazon S3 (console)

1. Nella pagina Funzioni della console Lambda, scegli la funzione per cui vuoi caricare un nuovo file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli Posizione Amazon S3.
5. Incolla l'URL del link Amazon S3 del tuo file .zip e scegli Salva.

Aggiornamento delle funzioni dei file .zip tramite l'editor di codice della console

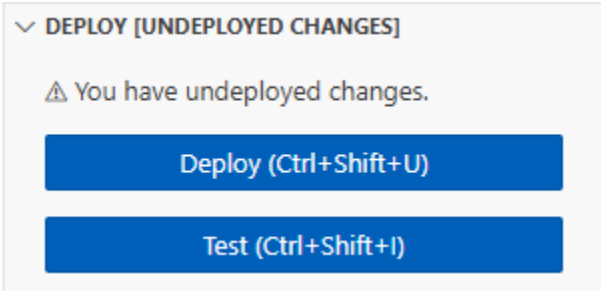
Per alcune funzioni con pacchetti di implementazione .zip, puoi utilizzare l'editor di codice integrato nella console Lambda per aggiornare direttamente il codice della funzione. Per utilizzare questa funzione, la funzione deve soddisfare i seguenti criteri:

- La funzione deve utilizzare uno dei runtime del linguaggio interpretato (Python, Node.js o Ruby)
- Il pacchetto di implementazione della funzione deve avere dimensioni inferiori a 50 MB (non compresso).

Il codice della funzione per le funzioni con pacchetti di implementazione di immagini di container non può essere modificato direttamente nella console.

Aggiornamento del codice della funzione utilizzando l'editor di codice della console

1. Apri la pagina Funzioni della console Lambda e scegli la tua funzione.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, seleziona il tuo file di codice sorgente e modificalo nell'editor di codice integrato.
4. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



Creazione e aggiornamento di funzioni con file.zip utilizzando il AWS CLI

È possibile utilizzare la AWS CLI per creare una nuova funzione o aggiornare una funzione esistente mediante un file .zip. Usa la funzione create-function e update-function-codei comandi per distribuire il tuo pacchetto .zip. Se il file .zip ha dimensioni inferiori a 50 MB, è possibile caricare il pacchetto .zip da una posizione di file nella macchina di compilazione locale. Per i file di dimensioni maggiori, è necessario caricare il pacchetto .zip da un bucket Amazon S3. Per istruzioni su come caricare un file su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

> **Note**
>
> Se carichi il tuo file.zip da un bucket Amazon S3 utilizzando AWS CLI il, il bucket deve trovarsi nella stessa posizione della Regione AWS tua funzione.

Per creare una nuova funzione utilizzando un file.zip con AWS CLI, devi specificare quanto segue:

- Il nome della funzione (`--function-name`)
- Il runtime della tua funzione (`--runtime`)
- Il nome della risorsa Amazon (ARN) del ruolo di esecuzione della funzione (`--role`)
- Il nome del metodo del gestore nel codice della funzione (`--handler`)

È inoltre necessario specificare la posizione del file .zip. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
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
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza l'opzione `--code` illustrata nel seguente comando di esempio. È necessario utilizzare il parametro `S3ObjectVersion` solo per gli oggetti con controllo delle versioni.

```
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
```

Per aggiornare una funzione esistente mediante la CLI, specifica il nome della funzione utilizzando il parametro `--function-name`. È inoltre necessario specificare la posizione del file .zip che desideri utilizzare per aggiornare il codice della funzione. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza le opzioni `--s3-bucket` e `--s3-key` come illustrato nel seguente comando di esempio. È necessario utilizzare il parametro `--s3-object-version` solo per gli oggetti con controllo delle versioni.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Creazione e aggiornamento delle funzioni con file .zip utilizzando l'API Lambda

Per creare e aggiornare le funzioni mediante un archivio di file .zip, utilizza le seguenti operazioni API:

- CreateFunction
- UpdateFunctionCode

Creazione e aggiornamento di funzioni con file.zip utilizzando AWS SAM

Il AWS Serverless Application Model (AWS SAM) è un toolkit che aiuta a semplificare il processo di creazione ed esecuzione di applicazioni serverless su. AWS Definisci le risorse per la tua applicazione in un modello YAML o JSON e utilizzi l'interfaccia a riga di AWS SAM comando (AWS SAM CLI) per creare, impacchettare e distribuire le tue applicazioni. Quando crei una funzione Lambda da un AWS SAM modello, crea AWS SAM automaticamente un pacchetto di distribuzione.zip o un'immagine del contenitore con il codice della funzione e le eventuali dipendenze specificate. Per ulteriori informazioni sull'utilizzo AWS SAM per creare e distribuire funzioni Lambda, consulta la Guida introduttiva AWS Serverless Application Model alla AWS SAM Developer Guide.

È inoltre possibile utilizzare AWS SAM per creare una funzione Lambda utilizzando un archivio di file.zip esistente. Per creare una funzione Lambda utilizzando AWS SAM, puoi salvare il tuo file.zip in un bucket Amazon S3 o in una cartella locale sulla tua macchina di compilazione. Per istruzioni su come caricare un file su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

Nel AWS SAM modello, la `AWS::Serverless::Function` risorsa specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `CodeUri`- impostato sull'URI Amazon S3 del codice della funzione, sul percorso della cartella locale o sull'oggetto FunctionCode
- `Runtime`: imposta il runtime prescelto

Inoltre AWS SAM, se il tuo file.zip è più grande di 50 MB, non è necessario caricarlo prima in un bucket Amazon S3. AWS SAM puoi caricare pacchetti.zip fino alla dimensione massima consentita di 250 MB (decompressi) da una posizione sulla macchina di compilazione locale.

Per ulteriori informazioni sulla distribuzione delle funzioni utilizzando il file.zip in AWS SAM, consulta la Guida per gli sviluppatori. AWS::Serverless::FunctionAWS SAM

Creazione e aggiornamento di funzioni con file.zip utilizzando CloudFormation

È possibile utilizzare CloudFormation per creare una funzione Lambda utilizzando un archivio di file.zip. Per creare una funzione Lambda da un file .zip, devi prima caricare il file su un bucket Amazon S3. Per istruzioni su come caricare un file su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

Nel CloudFormation modello, la `AWS::Lambda::Function` risorsa specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `Code`: inserisci il nome del bucket Amazon S3 e il nome del file .zip nei campi `S3Bucket` e `S3Key`
- `Runtime`: imposta il runtime prescelto

Il file.zip che CloudFormation genera non può superare i 4 MB. Per ulteriori informazioni sulla distribuzione delle funzioni utilizzando il file.zip in CloudFormation, AWS::Lambda::Functionconsultate la Guida per l'utente.CloudFormation

Distribuisci funzioni Lambda in Node.js con immagini di container

Esistono tre modi per creare un'immagine di container per una funzione Lambda in Node.js:

- Utilizzo di un'immagine di base per Node.js AWS

  Le immagini di base AWS sono precaricate con un runtime in linguaggio, un client di interfaccia di runtime per gestire l'interazione tra Lambda e il codice della funzione e un emulatore di interfaccia di runtime per i test locali.
- Utilizzo di un'immagine di AWS base solo per il sistema operativo

  AWS Le immagini di base solo per il sistema operativo contengono una distribuzione Amazon Linux e l'emulatore di interfaccia di runtime. Queste immagini vengono comunemente utilizzate per creare immagini di container per linguaggi compilati, come Go e Rust, e per un linguaggio o una versione di linguaggio per cui Lambda non fornisce un'immagine di base, come Node.js 19. Puoi anche utilizzare immagini di base solo per il sistema operativo al fine di implementare un runtime personalizzato. Per rendere l'immagine compatibile con Lambda, devi includere il client di interfaccia di runtime per Node.js nell'immagine.
- Utilizzo di un'immagine non di base AWS

  È possibile utilizzare un'immagine di base alternativa da un altro registro del container, come ad esempio Alpine Linux o Debian. Puoi anche utilizzare un'immagine personalizzata creata dalla tua organizzazione. Per rendere l'immagine compatibile con Lambda, devi includere il client di interfaccia di runtime per Node.js nell'immagine.

> **Tip**
>
> Per ridurre il tempo necessario all'attivazione delle funzioni del container Lambda, consulta Utilizzo di compilazioni a più fasi nella documentazione Docker. Per creare immagini di container efficienti, segui le best practice per scrivere file Docker.

Questa pagina spiega come creare, testare e implementare le immagini di container per Lambda.

Argomenti

- AWS immagini di base per Node.js
- Utilizzo di un'immagine di base per Node.js AWS

- Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

AWS immagini di base per Node.js

AWS fornisce le seguenti immagini di base per Node.js:

Tag	Runtime	Sistema operativo	Dockerfile	Raggiunta obsolescenza
24	Node.js 24	Amazon Linux 2023	Dockerfile per Node.js 24 ore su 24 GitHub	30 aprile 2028
22	Node.js 22	Amazon Linux 2023	Dockerfile per Node.js dalla 22 in poi GitHub	30 aprile 2027
20	Node.js 20	Amazon Linux 2023	Dockerfile per Node.js dalla versione 20 in poi GitHub	30 aprile 2026

Archivio Amazon ECR: gallery.ecr. aws/lambda/nodejs

Le immagini di base Node.js 20 e successive si basano sull'immagine minima del contenitore Amazon Linux 2023. Le immagini di base precedenti utilizzavano Amazon Linux 2. AL2023 offre diversi vantaggi rispetto ad Amazon Linux 2, tra cui un ingombro di distribuzione ridotto e versioni aggiornate di librerie come. `glibc`

AL2Le immagini basate su 023 utilizzano `microdnf` (symlinked as`dnf`) come gestore di pacchetti anzichéyum, che è il gestore di pacchetti predefinito in Amazon Linux 2. `microdnf`è un'implementazione autonoma di. `dnf` Per un elenco dei pacchetti inclusi nelle immagini AL2 basate su 023, consulta le colonne Minimal Container in Confronto dei pacchetti installati su Amazon Linux 2023 Container Images. Per ulteriori informazioni sulle differenze tra AL2 023 e Amazon Linux 2, consulta la sezione Introduzione al runtime di Amazon Linux 2023 AWS Lambda sul AWS Compute Blog.

> **Note**
>
> Per eseguire immagini AL2 basate su 023 localmente, incluso with AWS Serverless Application Model (AWS SAM), devi usare Docker versione 20.10.10 o successiva.

Utilizzo di un'immagine di base per Node.js AWS

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS CLI versione 2
- Docker (versione minima 25.0.0)
- Il plugin buildx di Docker.
- Node.js

Creazione di un'immagine da un'immagine di base

Per creare un'immagine del contenitore da un'immagine di AWS base per Node.js

1. Crea una directory per il progetto, quindi passa a quella directory.

   ```
   mkdir example
   cd example
   ```

2. Crea un nuovo progetto Node.js con `npm`. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi `Enter`.

   ```
   npm init
   ```

3. Crea un nuovo file denominato `index.js`. A fini di test, puoi utilizzare il codice della funzione di esempio seguente o sostituirlo con il tuo codice personalizzato.

   Example Gestore CommonJS

   ```
   exports.handler = async (event) => {
       const response = {
           statusCode: 200,
           body: JSON.stringify('Hello from Lambda!'),
       };
       return response;
   };
   ```

4. Se la tua funzione dipende da librerie diverse da AWS SDK per JavaScript, usa npm per aggiungerle al tuo pacchetto.

5. Crea un nuovo Dockerfile con la seguente configurazione:

   - Imposta la proprietà `FROM` sull'URI dell'immagine di base.
   - Utilizza il comando COPY per copiare il codice della funzione e le dipendenze di runtime in `{LAMBDA_TASK_ROOT}`, una variabile d'ambiente definita da Lambda.
   - Imposta l'argomento `CMD` specificando il gestore della funzione Lambda.

   Nota che l'esempio Dockerfile non include un'istruzione USER. Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

   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. Crea l'immagine Docker con il comando docker build. L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il tag `test`. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'`--provenance=false`opzione.

   ```
   docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
    .
   ```

   > **Note**
   >
   > Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Se intendi creare una funzione Lambda utilizzando l'architettura del set di ARM64 istruzioni, assicurati di modificare il comando per utilizzare invece l'`--platform linux/arm64`opzione.

(Facoltativo) Test dell'immagine in locale

1. Avvia l'immagine Docker con il comando docker run. In questo esempio, `docker-image` è il nome dell'immagine e `test` è il tag.

   ```
   docker run --platform linux/amd64 -p 9000:8080 docker-image:test
   ```

   Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Se hai creato l'immagine Docker per l'architettura del set di ARM64 istruzioni, assicurati di utilizzare l'`--platform linux/arm64`opzione invece di. `--platform linux/amd64`

2. Da una nuova finestra di terminale, invia un evento all'endpoint locale.

   Linux/macOS

   Su MacOS o Linux, esegui il comando seguente `curl`:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   In PowerShell, esegui il seguente `Invoke-WebRequest` comando:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Ottieni l'ID del container.

```
docker ps
```

4. Utilizza il comando docker kill per arrestare il container. In questo comando, sostituisci `3766c4ab331c` con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

Implementazione dell'immagine

Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Esegui il get-login-passwordcomando per autenticare la CLI Docker nel tuo registro Amazon ECR.
   - Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
   - Sostituiscilo `111122223333` con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il commando create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
    "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. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando docker tag per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:

   - `docker-image:test` è il nome e tag dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
   - Sostituisci l'`<ECRrepositoryUri>` con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando docker push per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crea un ruolo di esecuzione per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'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**
>
> È possibile creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta Autorizzazioni multiaccount Amazon ECR.

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nell'archivio Amazon ECR e quindi utilizzare il update-function-codecomando per distribuire l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per distribuire la nuova immagine nella stessa funzione Lambda, è necessario utilizzare update-function-codeil comando, anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

Se utilizzi un'immagine di base solo per il sistema operativo o un'immagine di base alternativa, devi includere il client dell'interfaccia di runtime nell'immagine. Il client dell'interfaccia di runtime estende l'API Runtime, che gestisce l'interazione tra Lambda e il codice della funzione.

Installa il client di interfaccia di runtime per Node.js utilizzando il gestore di pacchetti npm:

```
npm install aws-lambda-ric
```

È inoltre possibile scaricare il client dell'interfaccia di runtime Node.js da GitHub.

L'esempio seguente dimostra come creare un'immagine contenitore per Node.js utilizzando un'immagine non di AWS base. Il Dockerfile di esempio utilizza l'immagine di base `bookworm`. Il Dockerfile include il client di interfaccia di runtime.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS CLI versione 2

- Docker (versione minima 25.0.0)
- Il plugin buildx di Docker.
- Node.js

Creazione di un'immagine da un'immagine di base alternativa

Per creare un'immagine contenitore da un'immagine non di AWS base

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir example
cd example
```

2. Crea un nuovo progetto Node.js con `npm`. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi `Enter`.

```
npm init
```

3. Crea un nuovo file denominato `index.js`. A fini di test, puoi utilizzare il codice della funzione di esempio seguente o sostituirlo con il tuo codice personalizzato.

   Example Gestore CommonJS

```
exports.handler = async (event) => {
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
    return response;
};
```

4. Crea un nuovo Dockerfile. Il seguente Dockerfile utilizza un'immagine di base `bookworm` anziché un'immagine di base AWS. Il Dockerfile include il client di interfaccia di runtime, che rende l'immagine compatibile con Lambda. Il seguente Dockerfile utilizza una build multi-fase. Nella prima fase viene creata un'immagine della build, che è un ambiente Node.js standard in cui sono installate le dipendenze della funzione. Nella seconda fase viene creata un'immagine più leggera con il codice della funzione e le relative dipendenze. Ciò riduce la dimensione finale dell'immagine.

   - Imposta la proprietà `FROM` sull'identificativo dell'immagine di base.

- Utilizza il comando `COPY` per copiare il codice della funzione e le dipendenze di runtime.
- Imposta l'`ENTRYPOINT` sul modulo su cui desideri che il container Docker venga eseguito all'avvio. In questo caso, il modulo è il client di interfaccia di runtime.
- Imposta l'argomento `CMD` specificando il gestore della funzione Lambda.

Nota che l'esempio Dockerfile non include un'istruzione USER. Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

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. Crea l'immagine Docker con il comando docker build. L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il tag `test`. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'`--provenance=false`opzione.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Se intendi creare una funzione Lambda utilizzando l'architettura del set di ARM64 istruzioni, assicurati di modificare il comando per utilizzare invece l'`--platform linux/arm64`opzione.

(Facoltativo) Test dell'immagine in locale

Usa il simulatore dell'interfaccia di runtime per testare localmente l'immagine. Puoi creare l'emulatore nella tua immagine o seguire la procedura riportata e installarlo sul tuo computer locale.

Installazione ed esecuzione dell'emulatore di interfaccia di runtime sul computer locale

1. Dalla directory del progetto, esegui il comando seguente per scaricare l'emulatore di interfaccia di runtime (architettura x86-64) GitHub e installarlo sul computer locale.

   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
   ```

   Per installare l'emulatore arm64, sostituisci l'URL del GitHub repository nel comando precedente con il seguente:

   ```
   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
   ```

   Per installare l'emulatore arm64, sostituisci `$downloadLink` con quanto segue:

   ```
   https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/download/aws-lambda-rie-arm64
   ```

2. Avvia l'immagine Docker con il comando docker run. Tenere presente quanto segue:

   - `docker-image` è il nome dell'immagine e `test` è il tag.
   - `/usr/local/bin/npx aws-lambda-ric index.handler` è l'`ENTRYPOINT` seguito dal `CMD` del Dockerfile.

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
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Se hai creato l'immagine Docker per l'architettura del set di ARM64 istruzioni, assicurati di utilizzare l'`--platform linux/arm64`opzione invece di. `--platform linux/amd64`

3. Pubblica un evento nell'endpoint locale.

Linux/macOS

Su MacOS o Linux, esegui il comando seguente `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

In PowerShell, esegui il seguente `Invoke-WebRequest` comando:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Ottieni l'ID del container.

```
docker ps
```

5. Utilizza il comando docker kill per arrestare il container. In questo comando, sostituisci `3766c4ab331c` con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

Implementazione dell'immagine

Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Esegui il get-login-passwordcomando per autenticare la CLI Docker nel tuo registro Amazon ECR.
   - Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
   - Sostituiscilo `111122223333` con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
    "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. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando docker tag per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:

- `docker-image:test` è il nome e tag dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
- Sostituisci l'`<ECRrepositoryUri>` con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando docker push per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crea un ruolo di esecuzione per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'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**
>
> È possibile creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta Autorizzazioni multiaccount Amazon ECR.

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nell'archivio Amazon ECR e quindi utilizzare il update-function-codecomando per distribuire l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per distribuire la nuova immagine nella stessa funzione Lambda, è necessario utilizzare update-function-codeil comando, anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Utilizzo dei livelli per le funzioni Lambda Node.js

Usa i livelli Lambda per impacchettare codice e dipendenze che desideri riutilizzare in più funzioni. I livelli di solito contengono dipendenze dalla libreria, un runtime personalizzato o file di configurazione. La creazione di un livello prevede tre passaggi generali:

1. Crea un pacchetto per il contenuto del livello. Ciò significa creare un archivio di file con estensione .zip che contiene le dipendenze che desideri utilizzare nelle funzioni.
2. Crea il livello in Lambda.
3. Aggiungi il livello alle tue funzioni.

Argomenti

- Crea un pacchetto per il contenuto del livello
- Crea il layer in Lambda
- Aggiungete il layer alla vostra funzione
- App di esempio

Crea un pacchetto per il contenuto del livello

Per creare un layer, raggruppa i pacchetti in un archivio di file.zip che soddisfi i seguenti requisiti:

- Create il layer utilizzando la stessa versione di Node.js che intendete utilizzare per la funzione Lambda. Ad esempio, se create il layer utilizzando Node.js 24, utilizzate il runtime Node.js 24 per la funzione.
- Il file.zip del livello deve utilizzare una di queste strutture di directory:
  - `nodejs/node_modules`
  - `nodejs/node`*X*`/node_modules`(*X*dov'è la versione di Node.js, ad esempio`node22`)

  Per ulteriori informazioni, consulta Percorsi dei livelli per ciascun runtime Lambda.
- I pacchetti del livello devono essere compatibili con Linux. Le funzioni Lambda vengono eseguite su Amazon Linux.

Puoi creare livelli che contengono librerie Node.js di terze parti installate con `npm` (come `axios` o`lodash`) o JavaScript moduli personalizzati.

Dipendenze da terze parti

Per creare un livello utilizzando i pacchetti npm

1. Crea la struttura di directory richiesta e installa i pacchetti direttamente al suo interno:

```
mkdir -p nodejs
npm install --prefix nodejs lodash axios
```

Questo comando installa i pacchetti direttamente nella `nodejs/node_modules` directory, che è la struttura richiesta da Lambda.

> **Note**
>
> Per i pacchetti con dipendenze native o componenti binari (come sharp o bcrypt), assicurati che siano compatibili con l'ambiente Lambda Linux e l'architettura della tua funzione. Potrebbe essere necessario utilizzare il flag: `--platform`
>
> ```
> npm install --prefix nodejs --platform=linux --arch=x64 sharp
> ```
>
> Per dipendenze native più complesse, potrebbe essere necessario compilarle in un ambiente Linux che corrisponda al runtime Lambda. Puoi usare Docker per questo scopo.

2. Comprimi il contenuto del livello:

Linux/macOS

```
zip -r layer.zip nodejs/
```

PowerShell

```
Compress-Archive -Path .\nodejs -DestinationPath .\layer.zip
```

La struttura delle cartelle del file.zip dovrebbe essere la seguente:

```
nodejs/
### package.json
```

```
### package-lock.json
### node_modules/
    ### lodash/
    ### axios/
    ### (dependencies of the other packages)
```

> **Note**
>
> - Assicurati che il tuo file.zip includa la `nodejs` directory a livello principale al `node_modules` suo interno. Questa struttura assicura che Lambda possa localizzare e importare i tuoi pacchetti.
> - `package-lock.json`Il file `package.json` and nella `nodejs/` directory vengono utilizzati da npm per la gestione delle dipendenze ma non sono richiesti da Lambda per la funzionalità dei livelli. Ogni pacchetto installato contiene già il proprio `package.json` file che definisce il modo in cui Lambda importa il pacchetto.

Moduli personalizzati JavaScript

Per creare un livello utilizzando il proprio codice

1. Crea la struttura di directory richiesta per il tuo livello:

   ```
   mkdir -p nodejs/node_modules/validator
   cd nodejs/node_modules/validator
   ```

2. Crea un `package.json` file per il tuo modulo personalizzato per definire come deve essere importato:

   Example nodejs/node_modules/validator/package.json

   ```
   {
     "name": "validator",
     "version": "1.0.0",
     "type": "module",
     "main": "index.mjs"
   }
   ```

3. Crea il tuo file del JavaScript modulo:

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. Comprimi il contenuto del livello:

   Linux/macOS

   ```
   zip -r layer.zip nodejs/
   ```

PowerShell

```
Compress-Archive -Path .\nodejs -DestinationPath .\layer.zip
```

La struttura delle cartelle del file.zip dovrebbe essere la seguente:

```
nodejs/
### node_modules/
    ### validator/
        ### package.json
        ### index.mjs
```

5. Nella tua funzione, importa e usa i moduli. Esempio:

```
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'
    });
  }
```

```
};
```

È possibile utilizzare il seguente evento di test per richiamare la funzione:

```
{
    "body": "{\"productId\": \"ABC123\", \"quantity\": 2, \"priority\": \"express
\"}"
}
```

Risposta prevista:

```
{
  "statusCode": 200,
  "body": "{\"message\":\"Order validated successfully\",\"order\":{\"productId\":
\"ABC123\",\"quantity\":2,\"shippingPriority\":\"express\"}}"
}
```

Crea il layer in Lambda

È possibile pubblicare il layer utilizzando la console AWS CLI o la console Lambda.

AWS CLI

Esegui il publish-layer-version AWS CLI comando per creare il livello Lambda:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip
 --compatible-runtimes nodejs24.x
```

Il parametro di runtime compatibile è facoltativo. Quando specificato, Lambda utilizza questo parametro per filtrare i livelli nella console Lambda.

Console

Creazione di un livello (console)

1. Apri la pagina Layers (Livelli) nella console Lambda.
2. Scegli Create layer (Crea livello).
3. Scegli Carica un file.zip, quindi carica l'archivio.zip che hai creato in precedenza.

4. (Facoltativo) Per i runtime compatibili, scegliete il runtime Node.js che corrisponde alla versione Node.js utilizzata per creare il layer.
5. Scegli Create (Crea).

Aggiungete il layer alla vostra funzione

AWS CLI

Per collegare il layer alla tua funzione, esegui il update-function-configuration AWS CLI comando. Per il `--layers` parametro, utilizzate il livello ARN. L'ARN deve specificare la versione (ad esempio,`arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`). Per ulteriori informazioni, consulta Livelli e versioni di livelli.

```
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"
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Console

Aggiunta di un livello a una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la funzione.
3. Scorri verso il basso fino alla sezione Livelli, quindi scegli Aggiungi un livello.
4. In Scegli un livello, seleziona Livelli personalizzati, quindi scegli il tuo livello.

> **Note**
>
> Se non hai aggiunto un runtime compatibile quando hai creato il layer, il tuo layer non verrà elencato qui. È possibile specificare invece il livello ARN.

5. Scegliere Aggiungi.

App di esempio

Per altri esempi di utilizzo dei layer Lambda, consulta l'applicazione di esempio layer-nodejs nel repository Developer Guide. AWS Lambda GitHub Questa applicazione include un livello che contiene la libreria lodash. Dopo aver creato il layer, è possibile implementare e richiamare la funzione corrispondente per confermare che il layer funzioni come previsto.

Utilizzo dell'oggetto di contesto Lambda per recuperare le informazioni sulla funzione Node.js

Quando Lambda esegue la funzione, passa un oggetto Context al gestore. Questo oggetto fornisce i metodi e le proprietà che forniscono le informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

Metodi del contesto

- `getRemainingTimeInMillis()`: restituisce il numero di millisecondi rimasti prima del timeout dell'esecuzione.

Proprietà del contesto

- `functionName`: il nome della funzione Lambda.
- `functionVersion`: la versione della funzione.
- `invokedFunctionArn`: l'Amazon Resource Name (ARN) utilizzato per richiamare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `memoryLimitInMB`: la quantità di memoria allocata per la funzione.
- `awsRequestId`: l'identificatore della richiesta di invocazione.
- `logGroupName`: il gruppo di log per la funzione.
- `logStreamName`: il flusso di log per l'istanza della funzione.
- `identity`: (app per dispositivi mobili) Informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
  - `cognitoIdentityId`: l'identità autenticata di Amazon Cognito.
  - `cognitoIdentityPoolId`: il pool di identità Amazon Cognito che ha autorizzato l'invocazione.
- `clientContext`: (app per dispositivi mobili) Contesto client fornito a Lambda dall'applicazione client.
  - `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`: valori personalizzati impostati dall'applicazione mobile.
- `callbackWaitsForEmptyEventLoop`— Per impostazione predefinita (`true`), quando si utilizza un gestore di funzioni basato sul callback, Lambda attende che il ciclo di eventi sia vuoto dopo l'esecuzione del callback prima di terminare l'invoca della funzione. Imposta su `false` per inviare la risposta e terminare la chiamata immediatamente dopo l'esecuzione del callback anziché attendere che il ciclo di eventi sia vuoto. Gli eventi eccezionali continuano a verificarsi durante la chiamata successiva. Nota che Lambda supporta gestori di funzioni basati su callback solo per Node.js 22 e runtime precedenti.

La seguente funzione di esempio registra le informazioni di contesto e restituisce la posizione dei log.

Example File index.js

```
exports.handler = async function(event, context) {
  console.log('Remaining time: ', context.getRemainingTimeInMillis())
  console.log('Function name: ', context.functionName)
  return context.logStreamName
}
```

Registrare e monitorare funzioni Lambda in Node.js

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della funzione. L'ambiente del runtime Lambda invia i dettagli su ogni richiamo al flusso di log e inoltra i log e l'output del codice della funzione. Per ulteriori informazioni, consulta Lambda invia automaticamente i log delle funzioni a CloudWatch Logs..

Questa pagina descrive come produrre un output di registro dal codice della funzione Lambda e accedere ai log utilizzando AWS Command Line Interface la console Lambda o la console. CloudWatch

Sections

- Creazione di una funzione che restituisce i registri
- Utilizzo dei controlli di registrazione avanzati di Lambda con Node.js
- Visualizzazione dei log nella console Lambda
- Visualizzazione dei log nella console CloudWatch
- Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI
- Eliminazione dei log

Creazione di una funzione che restituisce i registri

Per generare i log dal codice della funzione, è possibile utilizzare i metodi dell'oggetto console o di qualsiasi libreria di registrazione che scriva su `stdout` o `stderr`. L'esempio seguente registra i valori delle variabili di ambiente e l'oggetto evento.

> **Note**
>
> Si consiglia di utilizzare tecniche come la convalida dell'input e la codifica dell'output durante la registrazione degli input. Se registri direttamente i dati di input, un utente malintenzionato potrebbe essere in grado di utilizzare il codice per rendere difficile l'individuazione delle manomissioni, falsificare le voci di log o aggirare i monitor dei log. Per ulteriori informazioni, consulta Neutralizzazione di output non corretta per i log in Enumerazione delle debolezze comuni.

Example File index.js – Registrazione dei log

```
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 Formato dei 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
```

Il runtime di Node.js registra le voci `START`, `END` e `REPORT` per ogni chiamata. A ogni voce registrata dalla funzione aggiunge un timestamp, l'ID della richiesta e il livello di log. La riga del report fornisce i seguenti dettagli.

Campi dati della riga REPORT

- RequestId— L'ID univoco della richiesta per la chiamata.
- Durata – La quantità di tempo che il metodo del gestore della funzione impiega durante l'elaborazione dell'evento.
- Durata fatturata – La quantità di tempo fatturata per la chiamata.
- Dimensioni memoria – La quantità di memoria allocata per la funzione.
- Quantità max utilizzata – La quantità di memoria utilizzata dalla funzione. Quando le invocazioni condividono un ambiente di esecuzione, Lambda riporta la memoria massima utilizzata in tutte le invocazioni. Questo comportamento potrebbe comportare un valore riportato superiore al previsto.
- Durata Init – Per la prima richiesta servita, la quantità di tempo impiegato dal runtime per caricare la funzione ed eseguire il codice al di fuori del metodo del gestore.
- XRAY TraceId — Per le richieste tracciate, l'ID di traccia.AWS X-Ray
- SegmentId— Per le richieste tracciate, l'ID del segmento X-Ray.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

Puoi visualizzare i log nella console Lambda, nella CloudWatch console Logs o dalla riga di comando.

Utilizzo dei controlli di registrazione avanzati di Lambda con Node.js

Per avere un maggiore controllo sul modo in cui i log delle tue funzioni vengono acquisiti, elaborati e utilizzati, puoi configurare le seguenti opzioni di registrazione per i runtime Node.js supportati:

- Formato di log: scegli tra i formati di testo normale e JSON strutturato per i log della funzione
- Livello di log: per i log in formato JSON, scegli il livello di dettaglio dei log che Lambda invia ad CloudWatch Amazon, come ERROR, DEBUG o INFO
- Gruppo di log: scegli il gruppo di log a cui la CloudWatch funzione invia i log

Per ulteriori informazioni su queste opzioni di registrazione e istruzioni su come configurare la funzione per utilizzarle, consulta la pagina the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”.

Per utilizzare le opzioni del formato di log e del livello di log con le funzioni Lambda in Node.js, consulta le istruzioni nelle sezioni seguenti.

Utilizzo di log JSON strutturati con Node.js

Se si seleziona JSON per il formato di registro della funzione, Lambda invierà l'output dei log utilizzando i metodi della console `console.trace` di`console.debug`,,, `console.log` `console.info`o`console.error`, `console.warn` e CloudWatch a come JSON strutturato. Ogni oggetto di log JSON contiene almeno quattro coppie chiave-valore con le seguenti chiavi:

- `"timestamp"`: l'ora in cui è stato generato il messaggio di log
- `"level"`: il livello di log assegnato al messaggio
- `"message"`: il contenuto del messaggio di log
- `"requestId"`: l'ID di richiesta univoco dell'invocazione alla funzione

A seconda del metodo di registrazione di log utilizzato dalla funzione, questo oggetto JSON può anche contenere coppie di chiavi aggiuntive. Ad esempio, se la funzione utilizza metodi della `console` per registrare i log degli oggetti di errore con più argomenti, l'oggetto JSON conterrà coppie chiave-valore aggiuntive con le chiavi `errorMessage`, `errorType` e `stackTrace`.

Se il codice utilizza già un'altra libreria di registrazione, come Powertools for AWS Lambda, per produrre log strutturati JSON, non è necessario apportare alcuna modifica. Lambda non codifica due volte i log che sono già codificati in JSON, quindi i log delle applicazioni della funzione continueranno a essere acquisiti come prima.

Per ulteriori informazioni sull'utilizzo del pacchetto Powertools for AWS Lambda logging per creare log strutturati JSON nel runtime Node.js, vedere. the section called “Registrazione dei log”

Esempi di output di log in formato JSON

Gli esempi seguenti mostrano come i vari output di log generati utilizzando i `console` metodi con argomenti singoli e multipli vengono acquisiti in CloudWatch Logs quando si imposta il formato di registro della funzione su JSON.

Il primo esempio utilizza il metodo `console.error` per generare una stringa semplice.

Example Codice di registrazione di Node.js

```
export const handler = async (event) => {
  console.error("This is a warning message");
  ...
}
```

Example Record di log JSON

```
{
    "timestamp":"2025-11-01T00:21:51.358Z",
    "level":"ERROR",
    "message":"This is a warning message",
    "requestId":"93f25699-2cbf-4976-8f94-336a0aa98c6f"
}
```

Con i metodi della `console`, puoi anche generare messaggi di log strutturati più complessi utilizzando argomenti singoli o multipli. Nel prossimo esempio, viene utilizzato `console.log` per generare due coppie chiave-valore con un singolo argomento. Nota che il `"message"` campo nell'oggetto JSON che Lambda invia CloudWatch ai log non è stringato.

Example Codice di registrazione di Node.js

```
export const handler = async (event) => {
  console.log({data: 12.3, flag: false});
  ...
}
```

Example Record di log JSON

```
{
    "timestamp": "2025-12-08T23:21:04.664Z",
    "level": "INFO",
    "requestId": "405a4537-9226-4216-ac59-64381ec8654a",
    "message": {
        "data": 12.3,
        "flag": false
    }
}
```

Nel prossimo esempio, viene utilizzato ancora una volta il metodo `console.log` per creare un output di log. Nella fattispecie, il metodo richiede due argomenti, una mappa contenente due coppie chiave-valore e una stringa identificativa. Tieni presente che in questo caso, poiché hai fornito due argomenti, Lambda rende il campo `"message"` in formato stringa.

Example Codice di registrazione di Node.js

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

```
  console.log('Some object - ', {data: 12.3, flag: false});
  ...
}
```

Example Record di 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 assegna gli output generati utilizzando il livello di log INFO della `console.log`.

L'ultimo esempio mostra come gli oggetti di errore possono essere inviati ai log utilizzando i metodi. CloudWatch `console` Tieni presente che quando esegui il log di oggetti di errore utilizzando più argomenti, Lambda aggiunge i campi `errorMessage`, `errorType` e `stackTrace` all'output del log.

Example Codice di registrazione di Node.js

```
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 Record di 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)"
    ]
}
```

Quando si registrano i log di più tipi di errore, i campi aggiuntivi `errorMessage`, `errorType` e `stackTrace` vengono estratti dal primo tipo di errore fornito al metodo della `console`.

Utilizzo di librerie client formato del parametro incorporato (EMF) con log JSON strutturati

AWS fornisce librerie client open source per Node.js che è possibile utilizzare per creare log in formato metrico incorporato (EMF). Se disponi di funzioni esistenti che utilizzano queste librerie e modifichi il formato di registro della funzione in JSON, CloudWatch potresti non riconoscere più le metriche emesse dal tuo codice.

Se attualmente il codice emette i log EMF direttamente utilizzando `console.log` o utilizzando Powertools for AWS Lambda (TypeScript), non CloudWatch sarà inoltre in grado di analizzarli se modificate il formato di registro della funzione in JSON.

> **Important**
>
> Per assicurarvi che i log EMF delle vostre funzioni continuino ad essere analizzati correttamente CloudWatch, aggiornate le librerie EMF e Powertools for alle versioni più recenti. AWS Lambda Se passi al formato di log JSON, ti consigliamo di eseguire dei test anche per garantire la compatibilità con i parametri incorporati della tua funzione. Se il tuo codice emette i log EMF direttamente utilizzando `console.log`, modifica il codice in modo che emetta tali parametri direttamente a `stdout`, come mostrato nel seguente esempio di codice.

Example codice che emette parametri incorporati a **`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")
```

Utilizzo del filtraggio a livello di log con Node.js

Per AWS Lambda filtrare i log delle applicazioni in base al loro livello di registro, la funzione deve utilizzare log in formato JSON. Puoi farlo in due modi:

- Crea output di log utilizzando i metodi standard della console e configura la tua funzione per utilizzare la formattazione dei log JSON. AWS Lambda quindi filtra gli output di log utilizzando

la coppia chiave-valore «level» nell'oggetto JSON descritto in. the section called “Utilizzo di log JSON strutturati con Node.js” Per informazioni su come configurare il formato di log della funzione, consulta la pagina the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”.

- Utilizza un'altra libreria o metodo di registrazione per creare nel codice dei log JSON strutturati che includono una coppia chiave-valore "livello" che definisce il livello dell'output log. Ad esempio, puoi utilizzare Powertools per generare output AWS Lambda di log strutturati JSON dal tuo codice. Per ulteriori informazioni sull'utilizzo di Powertools con il runtime Node.js, consulta la pagina the section called “Registrazione dei log”.

  Per consentire a Lambda di filtrare i log della funzione, è necessario includere anche una coppia chiave-valore `"timestamp"` nell'output log JSON. L'ora deve essere specificata in un formato di timestamp RFC 3339 valido. Se non fornisci un timestamp valido, Lambda assegnerà al log il livello INFO e aggiungerà un timestamp per tuo conto.

Quando configuri la tua funzione per utilizzare il filtraggio a livello di log, selezioni il livello di log che desideri inviare AWS Lambda a Logs tra le seguenti opzioni: CloudWatch

Livello di log	Utilizzo standard
TRACE (dettaglio massimo)	Le informazioni più dettagliate utilizzate per tracciare il percorso di esecuzione del codice
DEBUG	Informazioni dettagliate per il debug del sistema
INFO	Messaggi che registrano il normale funzionam ento della funzione
WARN	Messaggi relativi a potenziali errori che possono portare a comportamenti imprevisti se non risolti
ERRORE	Messaggi relativi a problemi che impediscono al codice di funzionare come previsto
FATAL (dettaglio minimo)	Messaggi relativi a errori gravi che causano l'interruzione del funzionamento dell'appl icazione

Lambda invia i log del livello selezionato e inferiore a. CloudWatch Ad esempio, se configuri un livello di log WARN, Lambda invierà i log corrispondenti ai livelli WARN, ERROR e FATAL.

Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor del codice incorporato, troverai i log nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l'output del log nella sezione Dettagli.

Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (***your-function-name***/aws/lambda/).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Viene visualizzato un flusso di log quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire le chiamate simultanee. Per trovare i log per una chiamata specifica, consigliamo di strumentare la funzione con. AWS X-Ray X-Ray registra i dettagli sulla richiesta e il flusso di log nella traccia.

Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della AWS CLI versione 2.

È possibile utilizzare AWS CLI per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'`base64` utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `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
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
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
```

L'utilità `base64` è disponibile su Linux, macOS e Ubuntu su Windows. Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

Example Script get-logs.sh

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza `sed` per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.

L'cli-binary-formatopzione è obbligatoria se utilizzi la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 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 (solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il seguente comando per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
    "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"
}
```

Eliminazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, eliminare il gruppo di log o configurare un periodo di conservazione trascorso il quale i log vengono eliminati automaticamente.

Strumentazione del codice Node.js in AWS Lambda

Lambda si integra con AWS X-Ray per aiutarti a tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzare X-Ray per tracciare una richiesta mentre attraversa le risorse nell'applicazione, che possono includere funzioni Lambda e altri servizi AWS .

Per inviare dati di tracciamento a X-Ray, è possibile utilizzare una delle due librerie SDK:

- AWS Distro for OpenTelemetry (ADOT): una distribuzione sicura, pronta per la produzione e supportata dell'SDK (). AWS OpenTelemetry OTel
- SDK AWS X-Ray per Node.js: un SDK per generare e inviare i dati di traccia su X-Ray.

Ciascuno di essi SDKs offre modi per inviare i dati di telemetria al servizio X-Ray. Puoi quindi utilizzare X-Ray per visualizzare, filtrare e analizzare le metriche delle prestazioni dell'applicazione per identificare i problemi e le opportunità di ottimizzazione.

> **Important**
>
> X-Ray e Powertools per AWS Lambda SDKs fanno parte di una soluzione di strumentazione strettamente integrata offerta da. AWS I livelli Lambda ADOT fanno parte di uno standard di settore per la strumentazione di tracciamento che in generale raccoglie più dati, ma potrebbero non essere adatti a tutti i casi d'uso. È possibile implementare il end-to-end tracciamento in X-Ray utilizzando entrambe le soluzioni. Per saperne di più sulla scelta tra di esse, consulta Scelta tra AWS Distro for Open Telemetry e X-Ray. SDKs

Sections

- Utilizzo di ADOT per strumentare le funzioni Node.js
- Utilizzo dell'SDK X-Ray per strumentare le funzioni Node.js
- Attivazione del tracciamento con la console Lambda
- Attivazione del tracciamento con l'API Lambda
- Attivazione del tracciamento con CloudFormation
- Interpretazione di una traccia X-Ray
- Memorizzazione delle dipendenze di runtime in un livello (SDK X-Ray)

Utilizzo di ADOT per strumentare le funzioni Node.js

ADOT fornisce layer Lambda completamente gestiti che racchiudono tutto il necessario per raccogliere dati di telemetria utilizzando l'SDK. OTel Usando questo livello, è possibile strumentare le funzioni Lambda senza dover modificare alcun codice funzione. Puoi anche configurare il tuo layer per eseguire l'inizializzazione personalizzata di. OTel Per ulteriori informazioni, consulta la sezione relativa alla configurazione personalizzata per ADOT Collector su Lambda nella documentazione di ADOT.

Per i runtime Node.js, puoi aggiungere il livello Lambda gestito da AWS per ADOT Javascript per strumentare automaticamente le tue funzioni. Per istruzioni dettagliate su come aggiungere questo layer, consulta AWS Distro for OpenTelemetry Lambda JavaScript Support nella documentazione ADOT.

Utilizzo dell'SDK X-Ray per strumentare le funzioni Node.js

Per registrare i dettagli sulle chiamate effettuate dalla funzione Lambda ad altre risorse nell'applicazione, è anche possibile utilizzare il SDK AWS X-Ray per Node.js. Per ottenere l'SDK, aggiungere il pacchetto `aws-xray-sdk-core` alle dipendenze dell'applicazione.

Example blank-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"
  }
}
```

Per strumentare i client AWS SDK nella versione AWS SDK per JavaScript v3, avvolgete l'istanza del client con il metodo. `captureAWSv3Client`

Example blank- nodejs/function/index .js — Tracciamento di un client SDK AWS

```
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 => {
  ...
```

Il runtime Lambda imposta alcune variabili di ambiente per configurare l'SDK X-Ray. Ad esempio, Lambda imposta `AWS_XRAY_CONTEXT_MISSING` su `LOG_ERROR` per evitare di generare errori di runtime dall'SDK X-Ray. Per impostare una strategia mancante di contesto personalizzato, sovrascrivi la variabile di ambiente nella configurazione della funzione in modo da non avere alcun valore, quindi puoi impostare la strategia mancante di contesto a livello di programmazione.

Example Esempio di codice di inizializzazione

```
const AWSXRay = require('aws-xray-sdk-core');

// Configure the context missing strategy to do nothing
AWSXRay.setContextMissingStrategy(() => {});
```

Per ulteriori informazioni, consulta the section called “Variabili di ambiente”.

Dopo aver aggiunto le dipendenze corrette e aver apportato le modifiche necessarie al codice, attivare il tracciamento nella configurazione della funzione tramite la console Lambda o l'API.

Attivazione del tracciamento con la console Lambda

Per attivare il tracciamento attivo sulla funzione Lambda con la console, attenersi alla seguente procedura:

Per attivare il tracciamento attivo

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.

3. Scegliere Configuration (Configurazione) e quindi Monitoring and operations tools (Strumenti di monitoraggio e operazioni).
4. In Strumenti di monitoraggio aggiuntivi scegli Modifica.
5. In CloudWatch Application Signals e AWS X-Ray, scegli Enable for Lambda service trace.
6. Scegli Save (Salva).

Attivazione del tracciamento con l'API Lambda

Configura il tracciamento sulla tua funzione Lambda con AWS o SDK, utilizza AWS CLI le seguenti operazioni API:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Il AWS CLI comando di esempio seguente abilita il tracciamento attivo su una funzione denominata my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

La modalità di tracciamento fa parte della configurazione specifica della versione quando si pubblica una versione della funzione. Non è possibile modificare la modalità di tracciamento in una versione pubblicata.

Attivazione del tracciamento con CloudFormation

Per attivare il tracciamento su una `AWS::Lambda::Function` risorsa in un CloudFormation modello, utilizzate la proprietà. `TracingConfig`

Example function-inline.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
```

```
      ...
```

Per una `AWS::Serverless::Function` risorsa AWS Serverless Application Model (AWS SAM), utilizzate la `Tracing` proprietà.

Example template.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretazione di una traccia X-Ray

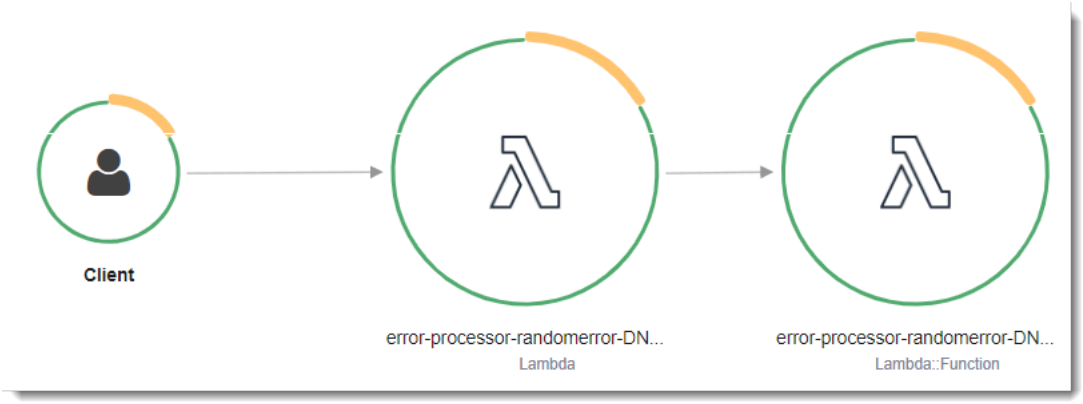
La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al ruolo di esecuzione della funzione. Altrimenti, aggiungete la AWSXRayDaemonWriteAccesspolitica al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico dei servizi X-Ray mostra informazioni sull'applicazione e tutti i suoi componenti. Il seguente esempio mostra un'applicazione con due funzioni. La funzione principale elabora gli eventi e talvolta restituisce errori. La seconda funzione in alto elabora gli errori che compaiono nel gruppo di log della prima e utilizza l' AWS SDK per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon Logs. CloudWatch

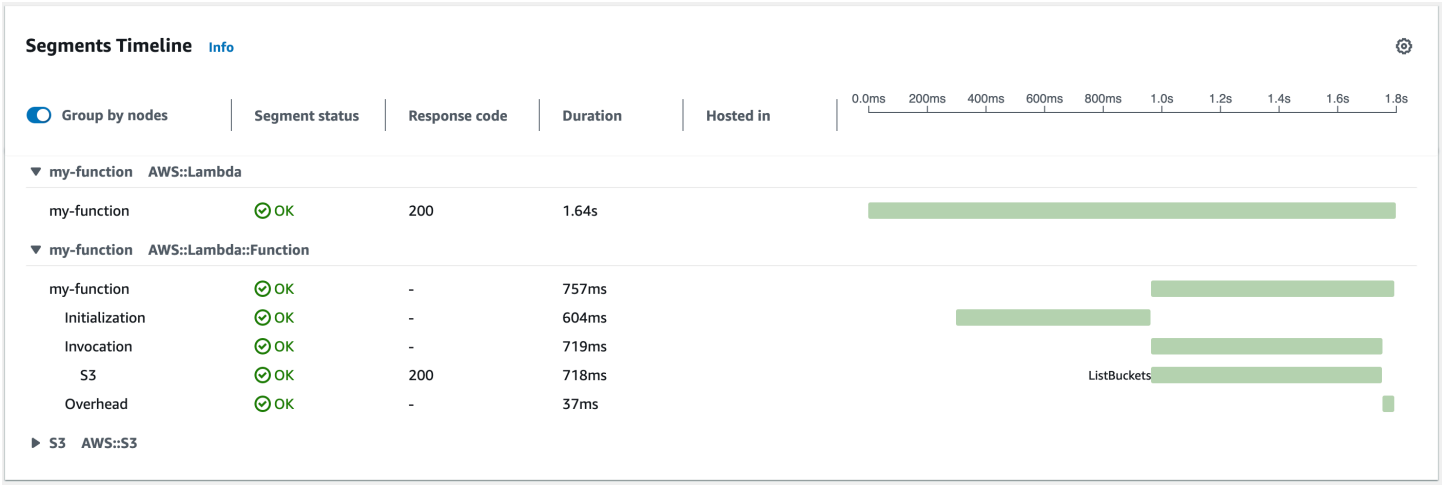


X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non è possibile configurare la frequenza di campionamento di X-Ray per le funzioni.

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda. L'esempio seguente mostra una traccia con questi 2 segmenti. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.



Questo esempio espande il segmento `AWS::Lambda::Function` per visualizzare i relativi tre sottosegmenti.

> **Note**
>
> AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. La traccia di esempio mostrata qui illustra il segmento di funzione vecchio stile. Le differenze tra i segmenti vecchio e nuovo stile sono descritte nei paragrafi seguenti.
> Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regioni, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia di nuovo formato.

Il segmento di funzioni vecchio stile contiene i seguenti sottosegmenti:

- Inizializzazione – Rappresenta il tempo trascorso a caricare la funzione e ad eseguire il codice di inizializzazione. Questo sottosegmento viene visualizzato solo per il primo evento che viene elaborato da ogni istanza della funzione.
- Chiamata: rappresenta il tempo impiegato per eseguire il codice del gestore.
- Overhead: rappresenta il tempo impiegato dal runtime Lambda per prepararsi a gestire l'evento successivo.

Il segmento di funzione di nuovo stile non contiene un sottosegmento `Invocation`. I sottosegmenti dei clienti sono invece collegati direttamente al segmento di funzioni. Per ulteriori informazioni sulla struttura dei segmenti di funzioni vecchio e nuovo stile, consulta the section called “Informazioni sui monitoraggi di X-Ray”.

È inoltre possibile strumentare i client HTTP, registrare query SQL e creare segmenti secondari personalizzati con annotazioni e metadati. Per ulteriori informazioni, consulta SDK AWS X-Ray per Node.js nella Guida per gli sviluppatori di AWS X-Ray .

> **Prezzi**
>
> Puoi utilizzare il tracciamento X-Ray gratuitamente ogni mese fino a un determinato limite come parte del AWS piano gratuito. Oltre la soglia, X-Ray addebita lo storage di traccia e il recupero. Per ulteriori informazioni, consultare Prezzi di AWS X-Ray.

Memorizzazione delle dipendenze di runtime in un livello (SDK X-Ray)

Se utilizzate X-Ray SDK per strumentare i client AWS SDK del codice della funzione, il pacchetto di distribuzione può diventare piuttosto grande. Per evitare di caricare dipendenze di runtime ogni volta che si aggiorna il codice della funzione, includere l'SDK X-Ray in un livello Lambda.

L'esempio seguente mostra una risorsa `AWS::Serverless::LayerVersion` che memorizza SDK AWS X-Ray per Node.js.

Example template.yml – Livello delle dipendenze

```
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
```

Con questa configurazione, si aggiorna il livello della libreria solo se si modificano le dipendenze di runtime. Poiché il pacchetto di implementazione della funzione contiene solo il codice, questo può contribuire a ridurre i tempi di caricamento.

La creazione di un layer per le dipendenze richiede modifiche alla compilazione per generare l'archivio dei layer prima della distribuzione. Per un esempio funzionante, vedere l'applicazione di esempio blank-nodejs .

Creazione di funzioni Lambda con TypeScript

È possibile utilizzare il runtime Node.js per eseguire il TypeScript codice. AWS Lambda Poiché Node.js non esegue il TypeScript codice in modo nativo, è necessario prima traspilare il TypeScript codice in. JavaScript Quindi, usa i JavaScript file per distribuire il codice della funzione in Lambda. Il codice viene eseguito in un ambiente che include l' AWS SDK for JavaScript, con le credenziali di un ruolo AWS Identity and Access Management (IAM) che gestisci. Per ulteriori informazioni sulle versioni SDK incluse nei runtime di Node.js, consulta the section called “Versioni SDK incluse nel runtime”.

Lambda supporta i seguenti runtime di Node.js.

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Node.js 24	`nodejs24.x`	Amazon Linux 2023	30 aprile 2028	1 giugno 2028	1 luglio 2028
Node.js 22	`nodejs22.x`	Amazon Linux 2023	30 aprile 2027	1 giugno 2027	1 luglio 2027
Node.js 20	`nodejs20.x`	Amazon Linux 2023	30 aprile 2026	31 agosto 2026	30 settembre 2026

Argomenti

- Configurazione di un TypeScript ambiente di sviluppo
- Definizioni dei tipi per Lambda
- Definisci il gestore di funzioni Lambda in TypeScript
- Implementa codice trascritto TypeScript in Lambda con archivi di file.zip
- Distribuire codice TypeScript trasformato in Lambda con immagini di container
- Utilizzo dell'oggetto contestuale Lambda per recuperare informazioni TypeScript sulla funzione
- Registra e monitora le funzioni TypeScript Lambda
- TypeScript Codice di tracciamento in AWS Lambda

Configurazione di un TypeScript ambiente di sviluppo

Utilizzate un ambiente di sviluppo integrato locale (IDE) o un editor di testo per scrivere il codice TypeScript della funzione. Non puoi creare TypeScript codice sulla console Lambda.

Puoi usare esbuild o il TypeScript compiler (`tsc`) di Microsoft per trasporre il codice in. TypeScript JavaScript AWS Serverless Application Model (AWS SAM) e AWS Cloud Development Kit (AWS CDK) usano entrambi esbuild.

Durante l'utilizzo di esbuild, considera quanto segue:

- Ci sono diverse avvertenze. TypeScript
- È necessario configurare le impostazioni di TypeScript traspilazione in modo che corrispondano al runtime di Node.js che si intende utilizzare. Per ulteriori informazioni, consulta Target nella documentazione esbuild. Per un esempio di file tsconfig.json che dimostra come indirizzare una versione specifica di Node.js supportata da Lambda, fai riferimento al repository. TypeScript GitHub
- esbuild non esegue controlli di tipo. Per controllare i tipi, utilizza il compilatore `tsc`. Esegui `tsc --noEmit` oppure aggiungi un parametro `"noEmit"` al file tsconfig.json come mostrato nell'esempio seguente. Questo configura per non emettere file. `tsc` JavaScript Dopo aver controllato i tipi, usa esbuild per convertire i TypeScript file in. 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"]
}
```

Definizioni dei tipi per Lambda

Il pacchetto @types/aws-lambda fornisce definizioni dei tipi per le funzioni Lambda. Installa questo pacchetto quando la tua funzione utilizza uno dei seguenti elementi:

- Fonti di AWS eventi comuni, come:
  - `APIGatewayProxyEvent`: per integrazioni proxy Amazon API Gateway
  - `SNSEvent`: per notifiche Amazon Simple Notification Service
  - `SQSEvent`: per messaggi Amazon Simple Queue Service
  - `S3Event`: per eventi di trigger S3
  - `DynamoDBStreamEvent`: per flussi Amazon DynamoDB
- L'oggetto contestuale Lambda
- Il modello di gestore di callback

Per aggiungere le definizioni dei tipi Lambda alla tua funzione, installa `@types/aws-lambda` come dipendenza di sviluppo:

```
npm install -D @types/aws-lambda
```

Quindi, importa i tipi da `aws-lambda`:

```
import { Context, S3Event, APIGatewayProxyEvent } from 'aws-lambda';

export const handler = async (event: S3Event, context: Context) => {
    // Function code
};
```

L'istruzione `import ... from 'aws-lambda'` importa le definizioni dei tipi. Non importa il pacchetto npm `aws-lambda`, che è uno strumento di terzi non correlato. Per ulteriori informazioni, consulta aws-lambda nel repository. DefinitelyTyped GitHub

> **Note**
>
> Non è necessario @types/aws-lambda quando si utilizzano definizioni di tipo personalizzate. Per una funzione di esempio che definisce il proprio tipo per un oggetto evento, vedi Esempio di codice della TypeScript funzione Lambda.

Definisci il gestore di funzioni Lambda in TypeScript

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Questa pagina descrive come utilizzare i gestori di funzioni Lambda in TypeScript, incluse le opzioni per la configurazione del progetto, le convenzioni di denominazione e le migliori pratiche. Questa pagina include anche un esempio di funzione TypeScript Lambda che raccoglie informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon Simple Storage Service (Amazon S3). Per informazioni su come distribuire una funzione dopo averla scritta, consulta o. the section called “Implementare archivi di file .zip” the section called “Implementazione di immagini di container”

Argomenti

- Configurazione del progetto TypeScript
- Esempio di codice della TypeScript funzione Lambda
- Moduli CommonJS ed ES
- Inizializzazione di Node.js
- Convenzioni di denominazione dei gestori
- Definizione e accesso all'oggetto evento di input
- Modelli di gestione validi per le funzioni TypeScript
- Utilizzo dell'SDK per la JavaScript versione 3 nel gestore
- Accesso alle variabili d'ambiente
- Utilizzo dello stato globale
- Procedure consigliate di codice per le TypeScript funzioni Lambda

Configurazione del progetto TypeScript

Utilizzate un ambiente di sviluppo integrato locale (IDE) o un editor di testo per scrivere il codice TypeScript della funzione. Non puoi creare TypeScript codice sulla console Lambda.

Esistono diversi modi per inizializzare un progetto TypeScript Lambda. Ad esempio, puoi creare un progetto utilizzandonpm, creare un'AWS SAM applicazione o creare un'AWS CDK applicazione. Per creare il progetto utilizzandonpm:

```
npm init
```

Il codice della funzione risiede in un `.ts` file, che trascrivi in un JavaScript file in fase di compilazione. Puoi usare esbuild o il TypeScript compiler (`tsc`) di Microsoft in cui trasporre il codice. TypeScript JavaScript Per usare esbuild, aggiungilo come dipendenza di sviluppo:

```
npm install -D esbuild
```

Un tipico progetto di funzione TypeScript Lambda segue questa struttura generale:

```
/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
```

Esempio di codice della TypeScript funzione Lambda

Il seguente codice di funzione Lambda di esempio raccoglie le informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon S3. Questo esempio definisce un tipo di evento personalizzato (). `OrderEvent` Per informazioni su come importare le definizioni dei tipi per le sorgenti di AWS eventi, consulta Definizioni dei tipi per Lambda.

> **Note**
>
> Questo esempio utilizza un gestore di moduli ES. Lambda supporta sia il modulo ES che i gestori CommonJS. Per ulteriori informazioni, consulta Moduli CommonJS ed ES.

Example funzione Lambda index.ts

```
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'}`);
    }
}
```

Questo file `index.ts` contiene le sezioni seguenti:

- `import`block: usa questo blocco per includere le librerie richieste dalla tua funzione Lambda, come i client AWS SDK.
- `const s3Client`dichiarazione: inizializza un client Amazon S3 al di fuori della funzione di gestione. Ciò fa sì che Lambda esegua questo codice durante la fase di inizializzazione e il client viene preservato per il riutilizzo su più chiamate.
- `type OrderEvent`: definisce la struttura dell'evento di input previsto.
- `export const handler`: Questa è la funzione di gestione principale richiamata da Lambda. Quando distribuisci la funzione, specifica `index.handler` la proprietà Handler. Il valore della `Handler` proprietà è il nome del file e il nome del metodo del gestore esportato, separati da un punto.
- `uploadReceiptToS3`funzione: Questa è una funzione di supporto a cui fa riferimento la funzione di gestione principale.

Affinché questa funzione funzioni correttamente, il suo ruolo di esecuzione deve consentire l'azione. `s3:PutObject` Inoltre, assicuratevi di definire la variabile di `RECEIPT_BUCKET` ambiente. Dopo una chiamata riuscita, il bucket Amazon S3 dovrebbe contenere un file di ricevuta.

Moduli CommonJS ed ES

Node.js supporta due sistemi di moduli: CommonJS e ECMAScript modules (moduli ES). Lambda consiglia di utilizzare i moduli ES in quanto supportano l'attesa di primo livello, che consente il completamento di attività asincrone durante l'inizializzazione dell'ambiente di esecuzione.

Node.js tratta i file con estensione come moduli CommonJS, mentre un'`.cjs`estensione indica i moduli ES. `.mjs` Per impostazione predefinita, Node.js tratta i file con l'estensione del nome `.js` file come moduli CommonJS. È possibile configurare Node.js per trattare `.js` i file come moduli ES specificando l'`type`as `module` nel file della funzione. `package.json` Puoi configurare Node.js

in Lambda per rilevare automaticamente se un `.js` file deve essere trattato come CommonJS o come modulo ES aggiungendo il `–experimental-detect-module` flag alla variabile di ambiente. `NODE_OPTIONS` Per ulteriori informazioni, consulta le funzionalità di Experimental Node.js.

Gli esempi seguenti mostrano gestori di funzioni scritti utilizzando sia i moduli ES che i moduli CommonJS. Gli esempi rimanenti in questa pagina utilizzano tutti i moduli ES.

Inizializzazione di Node.js

Node.js utilizza un I/O modello non bloccante che supporta operazioni asincrone efficienti utilizzando un ciclo di eventi. Ad esempio, se Node.js effettua una chiamata di rete, la funzione continua a elaborare altre operazioni senza bloccare una risposta di rete. Quando viene ricevuta, la risposta di rete viene inserita nella coda di callback. Le attività della coda vengono elaborate al termine dell'operazione corrente.

Lambda consiglia di utilizzare l'await di primo livello in modo che le attività asincrone avviate durante l'inizializzazione dell'ambiente di esecuzione vengano completate durante l'inizializzazione. Le attività asincrone che non vengono completate durante l'inizializzazione vengono in genere eseguite durante la prima chiamata della funzione. Ciò può causare comportamenti o errori imprevisti. Ad esempio, l'inizializzazione della funzione può effettuare una chiamata di rete per recuperare un AWS parametro da Parameter Store. Se questa attività non viene completata durante l'inizializzazione, il valore può essere nullo durante una chiamata. È inoltre possibile che si verifichi un ritardo tra l'inizializzazione e l'invoca, che può causare errori nelle operazioni che richiedono un fattore temporale. In particolare, le chiamate AWS di servizio possono basarsi su firme di richiesta sensibili al fattore tempo, con conseguenti errori nelle chiamate di servizio se la chiamata non viene completata durante la fase di inizializzazione. Il completamento delle attività durante l'inizializzazione in genere migliora le prestazioni di avvio a freddo e richiama innanzitutto le prestazioni quando si utilizza Provisioned Concurrency. Per ulteriori informazioni, consulta il nostro post sul blog Using Node.js ES modules and top-level await in. AWS Lambda

Convenzioni di denominazione dei gestori

Quando si configura una funzione, il valore dell'impostazione Handler è il nome del file e il nome del metodo del gestore esportato, separati da un punto. L'impostazione predefinita per le funzioni create nella console e negli esempi in questa guida è `index.handler`. Questo indica il metodo `handler` che viene esportato dal file `index.js` o `index.mjs`.

Se si crea una funzione nella console utilizzando un nome di file o un nome del gestore di funzione diverso, è necessario modificare il nome del gestore predefinito.

Modifica del nome del gestore funzioni (console)

1. Apri la pagina Funzioni della console Lambda e scegli la tua funzione.
2. Scegli la scheda Codice.
3. Scorri verso il basso fino al riquadro Impostazioni di runtime e scegli Modifica.
4. In Gestore, inserisci il nuovo nome per il tuo gestore di funzioni.
5. Scegli Save (Salva).

Definizione e accesso all'oggetto evento di input

JSON è il formato di input più comune e standard per le funzioni Lambda. In questo esempio, la funzione prevede un input simile a quanto segue:

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

Quando si utilizzano le funzioni Lambda in TypeScript, è possibile definire la forma dell'evento di input utilizzando un tipo o un'interfaccia. In questo esempio, definiamo la struttura dell'evento utilizzando un tipo:

```
type OrderEvent = {
    order_id: string;
    amount: number;
    item: string;
}
```

Dopo aver definito il tipo o l'interfaccia, usala nella firma del gestore per garantire la sicurezza del tipo:

```
export const handler = async (event: OrderEvent): Promise<string> => {
```

Durante la compilazione, TypeScript verifica che l'oggetto evento contenga i campi obbligatori con i tipi corretti. Ad esempio, il TypeScript compilatore segnala un errore se si tenta di utilizzare `event.order_id` come numero o `event.amount` come stringa.

Modelli di gestione validi per le funzioni TypeScript

Si consiglia di utilizzare async/await per dichiarare il gestore della funzione invece di utilizzare i callback. Async/await è un modo conciso e leggibile per scrivere codice asincrono, senza la necessità di callback annidati o di concatenare promesse. Con async/await puoi scrivere codice che si legga come codice sincrono, pur rimanendo asincrono e privo di blocchi.

Gli esempi in questa sezione utilizzano il tipo. `S3Event` Tuttavia, puoi utilizzare qualsiasi altro tipo di AWS evento nel pacchetto @types /aws-lambda o definire il tuo tipo di evento. Per utilizzare i tipi di: @types/aws-lambda

1. Aggiungi il @types/aws-lambda pacchetto come dipendenza di sviluppo:

   ```
   npm install -D @types/aws-lambda
   ```

2. Importa i tipi di cui hai bisogno, ad esempio `ContextS3Event`, o`Callback`.

gestori di funzioni asincrone (consigliati)

La parola chiave `async` contrassegna una funzione come asincrona, mentre la parola chiave `await` mette in pausa l'esecuzione della funzione fino alla risoluzione di `Promise`. Il gestore accetta i seguenti argomenti:

- `event`: contiene i dati di input passati alla funzione.
- `context`: contiene informazioni sull'invocazione, sulla funzione e sull'ambiente di esecuzione. Per ulteriori informazioni, consulta Utilizzo dell'oggetto contestuale Lambda per recuperare informazioni TypeScript sulla funzione.

Ecco le firme valide per il pattern: async/await

```
export const handler = async (event: S3Event): Promise<void> => { };
```

```
export const handler = async (event: S3Event, context: Context): Promise<void> => { };
```

> **Note**
>
> Quando elabori matrici di elementi in modo asincrono, assicurati di utilizzare await with `Promise.all` per garantire il completamento di tutte le operazioni. Metodi come `forEach`

don't wait che i callback asincroni vengano completati. Per ulteriori informazioni, consulta la sezione Array.prototype.forEach() nella documentazione di Mozilla.

Gestori di funzioni sincroni

Se la funzione non esegue attività asincrone, è possibile utilizzare un gestore di funzioni sincrono, utilizzando una delle seguenti firme di funzione:

```
export const handler = (event: S3Event): void => { };
```

```
export const handler = (event: S3Event, context: Context): void => { };
```

Gestori di funzioni di streaming delle risposte

Lambda supporta lo streaming delle risposte con Node.js. I gestori delle funzioni di streaming delle risposte utilizzano il decoratore AWSLambda.streamifyResponse () e accettano 3 parametri: event, responseStream e context. La firma della funzione è:

```
export const handler = awslambda.streamifyResponse(async (event: APIGatewayProxyEvent,
 responseStream: NodeJS.WritableStream, context: Context) => { });
```

Per ulteriori informazioni, consulta Response streaming per le funzioni Lambda.

Gestori di funzioni basati su callback

> **Note**
>
> I gestori di funzioni basati su callback sono supportati solo fino a Node.js 22. A partire da Node.js 24, le attività asincrone devono essere implementate utilizzando gestori di funzioni asincrone.

I gestori di funzioni basati su callback possono utilizzare gli argomenti event, context e callback. L'argomento callback prevede una risposta e una, che deve essere serializzabile in `Error` formato JSON.

Ecco la firma valida per il pattern del gestore di callback:

```
export const handler = (event: S3Event, context: Context, callback: Callback<void>):
 void => { };
```

La funzione continua a essere eseguita fino a quando il ciclo di eventi non è vuoto o la funzione scade. La risposta non viene inviata all'invoker finché tutte le attività del ciclo di eventi non giungono a termine. Se la funzione va in timeout, viene invece restituito un errore. È possibile configurare il runtime per inviare immediatamente la risposta impostando il contesto. callbackWaitsForEmptyEventLoopa false.

Example TypeScript funzione con callback

L'esempio seguente utilizzaAPIGatewayProxyCallback, che è un tipo di callback specializzato specifico per le integrazioni API Gateway. La maggior parte delle fonti di AWS eventi utilizza il `Callback` tipo generico mostrato nelle firme precedenti.

```
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',
        }),
    });
};
```

Utilizzo dell'SDK per la JavaScript versione 3 nel gestore

Spesso, utilizzerai le funzioni Lambda per interagire o aggiornare altre AWS risorse. Il modo più semplice per interfacciarsi con queste risorse consiste nell'utilizzare il AWS SDK per JavaScript. Tutti i runtime Lambda Node.js supportati includono l'SDK per la versione 3. JavaScript Tuttavia, consigliamo vivamente di includere i client AWS SDK necessari nel pacchetto di distribuzione. Ciò massimizza la compatibilità con le versioni precedenti durante i futuri aggiornamenti del runtime Lambda.

Per aggiungere dipendenze SDK alla tua funzione, usa il `npm install` comando per i client SDK specifici di cui hai bisogno. Nel codice di esempio, abbiamo usato il client Amazon S3. Aggiungi

questa dipendenza eseguendo il comando seguente nella directory che contiene il `package.json` file:

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

Nel codice della funzione, importate il client e i comandi necessari, come dimostra la funzione di esempio:

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

Quindi, inizializza un client Amazon S3:

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

In questo esempio, abbiamo inizializzato il nostro client Amazon S3 all'esterno della funzione del gestore principale per evitare di doverlo inizializzare ogni volta che invochiamo la nostra funzione. Dopo aver inizializzato il client SDK, puoi utilizzarlo per effettuare chiamate API per quel servizio. AWS Il codice di esempio richiama l'azione dell'PutObjectAPI Amazon S3 nel modo seguente:

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

Accesso alle variabili d'ambiente

Nel codice del gestore, puoi fare riferimento a qualsiasi variabile di ambiente utilizzando. `process.env` In questo esempio, facciamo riferimento alla variabile di ambiente `RECEIPT_BUCKET` definita utilizzando le seguenti righe di codice:

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

Utilizzo dello stato globale

Lambda esegue il codice statico durante la fase di inizializzazione prima di richiamare la funzione per la prima volta. Le risorse create durante l'inizializzazione restano in memoria tra le invocazioni, in modo da evitare di doverle creare ogni volta che si invoca la funzione.

Nel codice di esempio, il codice di inizializzazione del client S3 è esterno al gestore. Il runtime inizializza il client prima che la funzione gestisca il primo evento e il client rimane disponibile per il riutilizzo in tutte le chiamate.

Procedure consigliate di codice per le TypeScript funzioni Lambda

Segui queste linee guida durante la creazione di funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. In questo modo è possibile creare una funzione di cui è più semplice eseguire l'unit test.
- Controllare le dipendenze nel pacchetto di distribuzione della funzione. L'ambiente di AWS Lambda esecuzione contiene diverse librerie. Per i runtime Node.js e Python, questi includono. AWS SDKs Per abilitare il set di caratteristiche e aggiornamenti della sicurezza più recenti, Lambda aggiorna periodicamente tali librerie. Tali aggiornamenti possono introdurre lievi modifiche al comportamento della funzione Lambda. Per mantenere il controllo completo delle dipendenze utilizzate dalla funzione, inserire tutte le dipendenze nel pacchetto di implementazione.
- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del contesto di esecuzione.
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. In questo modo viene ridotta la quantità di tempo necessaria per il download del pacchetto e per la relativa decompressione prima dell'invocazione.

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedere Riutilizzo delle connessioni con Keep-Alive in Node.js.

Utilizzare le variabili di ambiente per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su `0` per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare documenti non documentati e non pubblici APIs nel codice della funzione Lambda. Per i runtime AWS Lambda gestiti, Lambda applica periodicamente aggiornamenti di sicurezza e funzionalità all'interno di Lambda. APIs Questi aggiornamenti delle API interne possono essere incompatibili con le versioni precedenti e portare a conseguenze indesiderate, come errori di chiamata se la funzione dipende da questi elementi non pubblici. APIs Consulta il riferimento alle API per un elenco di quelle disponibili al pubblico. APIs

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda?.

Implementa codice trascritto TypeScript in Lambda con archivi di file.zip

Prima di poter distribuire il TypeScript codice in AWS Lambda, devi trasporlo in. JavaScript Questa pagina spiega tre modi per creare e distribuire TypeScript codice in Lambda con archivi di file.zip:

- Usare () AWS Serverless Application ModelAWS SAM
- Usando il AWS Cloud Development Kit (AWS CDK)
- Usando AWS Command Line Interface (AWS CLI) ed esbuild

AWS SAM e AWS CDK semplifica la creazione e l'implementazione delle funzioni. TypeScript La specifica del AWS SAM modello fornisce una sintassi semplice e chiara per descrivere le funzioni APIs, le autorizzazioni, le configurazioni e gli eventi Lambda che costituiscono l'applicazione serverless. AWS CDK consente di creare applicazioni affidabili, scalabili e convenienti nel cloud con la notevole potenza espressiva di un linguaggio di programmazione. AWS CDK È destinato a utenti con esperienza da moderata a elevata. AWS Sia gli che AWS CDK i AWS SAM usano esbuild per trascrivere il codice TypeScript in. JavaScript

Utilizzo AWS SAM per distribuire TypeScript codice in Lambda

Segui i passaggi seguenti per scaricare, creare e distribuire un' TypeScript applicazione Hello World di esempio utilizzando. AWS SAM Questa applicazione implementa un backend API di base. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando si invia una richiesta GET all'endpoint API Gateway, viene richiamata la funzione Lambda. La funzione restituisce un messaggio `hello world`.

> **Note**
>
> AWS SAM usa esbuild per creare le funzioni Lambda di Node.js TypeScript dal codice. il supporto di esbuild è attualmente in anteprima pubblica. Durante l'anteprima pubblica, il supporto di esbuild potrebbe essere soggetto a modifiche incompatibili con il passato.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS CLI versione 2
- AWS SAM CLI versione 1.75 o successiva
- Node.js

Implementa un'applicazione di esempio AWS SAM

1. Inizializza l'applicazione utilizzando il modello Hello World. TypeScript

```
sam init --app-template hello-world-typescript --name sam-app --package-type Zip --runtime nodejs24.x
```

2. (Facoltativo) L'applicazione di esempio include configurazioni per strumenti di uso comune, come per il linting del codice e Jest ESLintper il test delle unità. Per eseguire i comandi lint e test:

```
cd sam-app/hello-world
npm install
npm run lint
npm run test
```

3. Costruisci l'app.

```
cd sam-app
sam build
```

4. Distribuire l'app.

```
sam deploy --guided
```

5. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, rispondi con `Enter`.
6. L'output mostra l'endpoint per la REST API. Apri l'endpoint in un browser per testare la funzione. Dovresti vedere questa risposta:

```
{"message":"hello world"}
```

7. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

Utilizzo di AWS CDK per distribuire il TypeScript codice in Lambda

Segui i passaggi seguenti per creare e distribuire un' TypeScript applicazione di esempio utilizzando. AWS CDK Questa applicazione implementa un backend API di base. È costituito da un endpoint API Gateway e da una funzione Lambda. Quando si invia una richiesta GET all'endpoint API Gateway, viene richiamata la funzione Lambda. La funzione restituisce un messaggio `hello world`.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS CLI versione 2
- AWS CDK versione 2
- Node.js
- Docker o esbuild

Implementa un'applicazione di esempio AWS CDK

1. Crea una directory di progetto per la nuova applicazione.

```
mkdir hello-world
cd hello-world
```

2. Inizializza l'app.

```
cdk init app --language typescript
```

3. Aggiungi il pacchetto @types /aws-lambda come dipendenza di sviluppo. Questo pacchetto contiene le definizioni dei tipi per Lambda.

```
npm install -D @types/aws-lambda
```

4. Apri la directory lib. Dovresti vedere un file chiamato hello-world-stack.ts. Crea due nuovi file in questa directory: hello-world.function.ts e hello-world.ts.
5. Apri hello-world.function.ts e aggiungi il seguente codice al file. Questo è il codice per la funzione Lambda.

> **Note**
>
> L'istruzione `import` importa le definizioni dei tipi da @types/aws-lambda. Non importa il pacchetto NPM `aws-lambda`, che è uno strumento di terzi non correlato. Per ulteriori informazioni, consulta aws-lambda nel repository. 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. Apri hello-world.ts e aggiungi il seguente codice al file. Contiene il NodejsFunction costrutto, che crea la funzione Lambda, e il costrutto, che crea LambdaRestApi l'API REST.

```
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,
    });
  }
}
```

Il costrutto `NodejsFunction` presuppone quanto segue per impostazione predefinita:

- Viene richiamato il gestore di funzioni `handler`.
- Il file .ts che contiene il codice funzione (hello-world.function.ts) si trova nella stessa directory del file .ts che contiene il costrutto (hello-world.ts). Il costrutto utilizza l'ID del costrutto ("hello-world") e il nome del file del gestore Lambda ("funzione") per trovare il codice funzione. Ad esempio, se il codice funzione si trova in un file chiamato hello-world.my-function.ts, il file hello-world.ts deve fare riferimento al codice funzione in questo modo:

```
const helloFunction = new NodejsFunction(this, 'my-function');
```

È possibile modificare questo comportamento e configurare altri parametri esbuild. Per ulteriori informazioni, consulta Configurazione di esbuild nel riferimento all'API. AWS CDK

7. Apri .tshello-world-stack. Questo è il codice che definisce lo stack AWS CDK. Sostituisci il codice con il seguente:

```
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. dalla directory `hello-world` contenente il file `cdk.json`, implementa l'applicazione.

```
cdk deploy
```

9. AWS CDK Compila e impacchetta la funzione Lambda utilizzando esbuild, quindi distribuisce la funzione nel runtime Lambda. L'output mostra l'endpoint per la REST API. Apri l'endpoint in un browser per testare la funzione. Dovresti vedere questa risposta:

```
{"message":"hello world"}
```

Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

Utilizzo di AWS CLI ed esbuild per distribuire il TypeScript codice in Lambda

L'esempio seguente dimostra come traspilare e distribuire codice in TypeScript Lambda utilizzando esbuild e. esbuild produce un file con tutte le AWS CLI dipendenze. JavaScript Questo è l'unico file che devi aggiungere all'archivio .zip.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS CLI versione 2
- Node.js
- Un ruolo di esecuzione per la funzione Lambda
- Per gli utenti Windows, un'utilità di file zip come 7zip.

Implementare una funzione di esempio

1. Sul tuo computer locale, crea una directory di progetto per la nuova funzione.
2. Crea un nuovo progetto Node.js con npm o un gestore di pacchetti di tua scelta.

   ```
   npm init
   ```

3. Aggiungi i pacchetti @types/aws-lambda e esbuild come dipendenze di sviluppo. Il pacchetto `@types/aws-lambda` contiene le definizioni dei tipi per Lambda.

   ```
   npm install -D @types/aws-lambda esbuild
   ```

4. Crea un nuovo file denominato index.ts. Aggiungi il seguente codice al nuovo file. Questo è il codice per la funzione Lambda. La funzione restituisce un messaggio `hello world`. La funzione non crea risorse API Gateway.

   > **Note**
   >
   > L'istruzione `import` importa le definizioni dei tipi da @types/aws-lambda. Non importa il pacchetto NPM `aws-lambda`, che è uno strumento di terzi non correlato. Per ulteriori informazioni, consulta aws-lambda nel repository. 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. Aggiungi uno script di compilazione al file package.json. esbuild viene configurato per creare automaticamente il pacchetto di implementazione .zip. Per ulteriori informazioni, consulta la sezione relativa alla compilazione degli script nella documentazione di 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

   In questo esempio, il comando `"postbuild"` utilizza l'utilità 7zip per creare il file con estensione .zip. Utilizza l'utilità zip di Windows preferita e modifica il comando secondo necessità.

   ```
   "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. Creare il pacchetto.

```
npm run build
```

7. Crea una funzione Lambda utilizzando il pacchetto di implementazione .zip. Sostituisci il testo evidenziato con l'Amazon Resource Name (ARN) del tuo ruolo di esecuzione.

```
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. Esegui un evento di test per confermare che la funzione restituisca la risposta seguente. Se desideri richiamare questa funzione utilizzando API Gateway, crea e configura una REST API.

```
{
  "statusCode": 200,
  "body": "{\"message\":\"hello world\"}"
}
```

Distribuire codice TypeScript trasformato in Lambda con immagini di container

È possibile distribuire il codice TypeScript in una funzione AWS Lambda come un'immagine di container Node.js. AWS fornisce immagini di base per Node.js per consentire la creazione dell'immagine di container. Queste immagini di base sono precaricate con un runtime di lingua e altri componenti necessari per eseguire l'immagine su Lambda. AWS fornisce un Dockerfile per ciascuna delle immagini di base per aiutare nella creazione dell'immagine di container.

Se si utilizza un'immagine di base di community o aziendale privata, è necessario aggiungere il client di interfaccia di runtime (RIC) Node.js all'immagine di base per renderla compatibile con Lambda.

Lambda fornisce un emulatore di interfaccia di runtime per i test in locale. Le immagini di base AWS per Node.js includono l'emulatore di interfaccia di runtime. Se utilizzi un'immagine di base alternativa, come Alpine Linux o Debian, puoi creare il simulatore nella tua immagine o installarlo sul tuo computer locale.

Utilizzo di un'immagine di base Node.js per creare e compilare il codice funzione TypeScript

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS CLI versione 2
- Docker (versione minima 25.0.0)
- Il plugin Docker buildx.
- Node.js 22.x

Creazione di un'immagine da un'immagine di base

Per creare un'immagine da un'immagine di base AWS per Lambda

1. Sul tuo computer locale, crea una directory di progetto per la nuova funzione.
2. Crea un nuovo progetto Node.js con `npm` o un gestore di pacchetti a tua scelta.

```
npm init
```

3. Aggiungi i pacchetti @types/aws-lambda e esbuild come dipendenze di sviluppo. Il pacchetto `@types/aws-lambda` contiene le definizioni dei tipi per Lambda.

```
npm install -D @types/aws-lambda esbuild
```

4. Aggiungi uno script di compilazione al file `package.json`.

```
  "scripts": {
  "build": "esbuild index.ts --bundle --minify --sourcemap --platform=node --
target=es2020 --outfile=dist/index.js"
}
```

5. Crea un nuovo file denominato `index.ts`. Aggiungi il codice di esempio seguente al nuovo file. Questo è il codice per la funzione Lambda. La funzione restituisce un messaggio `hello world`.

> **Note**
>
> L'istruzione `import` importa le definizioni dei tipi da @types/aws-lambda. Non importa il pacchetto NPM `aws-lambda`, che è uno strumento di terzi non correlato. Per ulteriori informazioni, consulta aws-lambda nel repository GitHub DefinitelyTyped.

```
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. Crea un nuovo Dockerfile con la seguente configurazione:

   - Imposta la proprietà `FROM` sull'URI dell'immagine di base.
   - Imposta l'argomento `CMD` per specificare il gestore della funzione Lambda.

Il seguente Dockerfile di esempio utilizza una build multi-fase. Il primo passaggio trasforma il codice TypeScript in JavaScript. Il secondo passaggio produce un'immagine di container che contiene solo file JavaScript e dipendenze di produzione.

Nota che l'esempio Dockerfile non include un'istruzione USER. Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

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. Crea l'immagine Docker con il comando docker build. L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il tag `test`. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'`--provenance=false`opzione.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Per creare una funzione Lambda utilizzando l'architettura del set di istruzioni ARM64, accertati di modificare il comando per utilizzare l'opzione `--platform linux/arm64`.

(Facoltativo) Test dell'immagine in locale

1. Avvia l'immagine Docker con il comando docker run. In questo esempio, `docker-image` è il nome dell'immagine e `test` è il tag.

   ```
   docker run --platform linux/amd64 -p 9000:8080 docker-image:test
   ```

   Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Per creare l'immagine Docker per l'architettura del set di istruzioni ARM64, assicurati di utilizzare l'opzione `--platform linux/arm64` anziché `--platform linux/amd64`.

2. Da una nuova finestra di terminale, invia un evento all'endpoint locale.

   Linux/macOS

   Su MacOS o Linux, esegui il comando seguente `curl`:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   In PowerShell, esegui il comando seguente `Invoke-WebRequest`:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Ottieni l'ID del container.

```
docker ps
```

4. Utilizza il comando docker kill per arrestare il container. In questo comando, sostituisci `3766c4ab331c` con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

Implementazione dell'immagine

Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Per autenticare la CLI Docker nel registro Amazon ECR, esegui il comando get-login-password.
   - Imposta il valore di `--region` sulla Regione AWS in cui desideri creare il repository Amazon ECR.
   - Sostituisci `111122223333` con l'ID dell'Account AWS.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il commando create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Il repository Amazon ECR deve essere nella stessa Regione AWS della funzione Lambda.

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
    "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. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando docker tag per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:

   - `docker-image:test` è il nome e tag dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
   - Sostituisci l'`<ECRrepositoryUri>` con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando docker push per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crea un ruolo di esecuzione per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'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**
>
> È possibile creare una funzione utilizzando un'immagine in un account AWS diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta Autorizzazioni multiaccount Amazon ECR.

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nel repository Amazon ECR e quindi utilizzare il comando update-function-code per implementare l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per implementare la nuova immagine nella stessa funzione Lambda, devi utilizzare il comando update-function-code anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Utilizzo dell'oggetto contestuale Lambda per recuperare informazioni TypeScript sulla funzione

Quando Lambda esegue la funzione, passa un oggetto Context al gestore. Questo oggetto fornisce i metodi e le proprietà che forniscono le informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

Per abilitare il controllo del tipo per l'oggetto di contesto, è necessario aggiungere il pacchetto @types /aws-lambda come dipendenza di sviluppo e importare il tipo. `Context` Per ulteriori informazioni, consulta Definizioni dei tipi per Lambda.

Metodi del contesto

- `getRemainingTimeInMillis()`: restituisce il numero di millisecondi rimasti prima del timeout dell'esecuzione.

Proprietà del contesto

- `functionName`: il nome della funzione Lambda.
- `functionVersion`: la versione della funzione.
- `invokedFunctionArn`: l'Amazon Resource Name (ARN) utilizzato per richiamare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `memoryLimitInMB`: la quantità di memoria allocata per la funzione.
- `awsRequestId`: l'identificatore della richiesta di invocazione.
- `logGroupName`: il gruppo di log per la funzione.
- `logStreamName`: il flusso di log per l'istanza della funzione.
- `identity`: (app per dispositivi mobili) Informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
  - `cognitoIdentityId`: l'identità autenticata di Amazon Cognito.
  - `cognitoIdentityPoolId`: il pool di identità Amazon Cognito che ha autorizzato l'invocazione.
- `clientContext`: (app per dispositivi mobili) Contesto client fornito a Lambda dall'applicazione client.
  - `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`: valori personalizzati impostati dall'applicazione mobile.
- `callbackWaitsForEmptyEventLoop`— Per impostazione predefinita (`true`), quando si utilizza un gestore di funzioni basato sul callback, Lambda attende che il ciclo di eventi sia vuoto dopo l'esecuzione del callback prima di terminare l'invoca della funzione. Imposta su `false` per inviare la risposta e terminare la chiamata immediatamente dopo l'esecuzione del callback anziché attendere che il ciclo di eventi sia vuoto. Gli eventi eccezionali continuano a verificarsi durante la chiamata successiva. Nota che Lambda supporta gestori di funzioni basati su callback solo per Node.js 22 e runtime precedenti.

Example File index.js

La seguente funzione di esempio registra le informazioni di contesto e restituisce la posizione dei log.

> **Note**
>
> Prima di utilizzare questo codice in una funzione Lambda, devi aggiungere il pacchetto @types/aws-lambda come dipendenza di sviluppo. Questo pacchetto contiene le definizioni dei tipi per Lambda. Per ulteriori informazioni, consulta Definizioni dei tipi per 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;
};
```

Registra e monitora le funzioni TypeScript Lambda

AWS Lambda monitora automaticamente le funzioni Lambda e invia le voci di registro ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della funzione. L'ambiente di runtime di Lambda invia al flusso di log i dettagli su ogni invocazione e altri output dal codice della funzione. Per ulteriori informazioni sui CloudWatch registri, consulta. Lambda invia automaticamente i log delle funzioni a CloudWatch Logs.

Per i log di output dal codice della funzione, puoi utilizzare i metodi nell'oggetto console. Per una registrazione più dettagliata, puoi utilizzare qualunque libreria di registrazione che scrive su `stdout` o `stderr`.

Sections

- Utilizzo di strumenti di registrazione e librerie
- Utilizzo di Powertools per AWS Lambda () e per la registrazione TypeScript strutturata AWS SAM
- Utilizzo di Powertools for AWS Lambda (TypeScript) e the AWS CDK per la registrazione strutturata
- Visualizzazione dei log nella console Lambda
- Visualizzazione dei log nella console CloudWatch

Utilizzo di strumenti di registrazione e librerie

Powertools for AWS Lambda (TypeScript) è un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori. L'utilità di registrazione fornisce un logger ottimizzato per Lambda che include informazioni aggiuntive sul contesto delle funzioni in tutte le funzioni con output strutturato come JSON. Utilizza l'utility per eseguire le seguenti operazioni:

- Acquisizione di campi essenziali dal contesto Lambda, avvii a freddo e output di registrazione della struttura come JSON
- Registrazione degli eventi di chiamata Lambda quando richiesto (disabilitata per impostazione predefinita)
- Stampa di tutti i log solo per una percentuale di chiamate tramite campionamento dei log (disabilitata per impostazione predefinita)
- Aggiunta di chiavi supplementari al log strutturato in qualsiasi momento
- Utilizzo di un formattatore di log personalizzato (Bring Your Own Formatter) per generare i log in una struttura compatibile con Logging RFC dell'organizzazione

Utilizzo di Powertools per AWS Lambda () e per la registrazione TypeScript strutturata AWS SAM

Segui i passaggi seguenti per scaricare, creare e distribuire un' TypeScript applicazione Hello World di esempio con i moduli Powertools for AWS Lambda (TypeScript) integrati utilizzando il. AWS SAM Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio `hello world`.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Node.js 2.0 o versione successiva
- AWS CLI versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS SAM

1. Inizializza l'applicazione utilizzando il modello Hello World. TypeScript

```
sam init --app-template hello-world-powertools-typescript --name sam-app --package-type Zip --runtime nodejs24.x
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi `Enter`.

> **Note**
>
> Perché HelloWorldFunction potrebbe non avere un'autorizzazione definita. Va bene? , assicurati di entrarey.

5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Richiama l'endpoint dell'API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere i log per la funzione, esegui sam logs. Per ulteriori informazioni, consulta l'argomento relativo all'utilizzo dei log nella Guida per sviluppatori AWS Serverless Application Model .

```
sam logs --stack-name sam-app
```

L'output del log ha la struttura seguente:

```
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. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

Gestione della conservazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, elimina il gruppo di log o configura un periodo di conservazione dopo il quale i log CloudWatch vengono eliminati automaticamente. Per configurare la conservazione dei log, aggiungi quanto segue al tuo modello: AWS SAM

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
    # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

Utilizzo di Powertools for AWS Lambda (TypeScript) e the AWS CDK per la registrazione strutturata

Segui i passaggi seguenti per scaricare, creare e distribuire un' TypeScript applicazione Hello World di esempio con moduli Powertools for AWS Lambda (TypeScript) integrati utilizzando. AWS CDK

Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio `hello world`.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Node.js 2.0 o versione successiva
- AWS CLI versione 2
- AWS CDK versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS CDK

1. Crea una directory di progetto per la nuova applicazione.

```
mkdir hello-world
cd hello-world
```

2. Inizializza l'app.

```
cdk init app --language typescript
```

3. Aggiungi il pacchetto @types /aws-lambda come dipendenza di sviluppo.

```
npm install -D @types/aws-lambda
```

4. Installa l'utilità Logger di Powertools.

```
npm install @aws-lambda-powertools/logger
```

5. Apri la directory lib. Dovresti vedere un file chiamato hello-world-stack.ts. Crea due nuovi file in questa directory: hello-world.function.ts e hello-world.ts.
6. Apri hello-world.function.ts e aggiungi il seguente codice al file. Questo è il codice per la funzione 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. Apri hello-world.ts e aggiungi il seguente codice al file. Contiene il NodejsFunction costrutto che crea la funzione Lambda, configura le variabili di ambiente per Powertools e imposta la conservazione dei log su una settimana. Include anche il LambdaRestApi costrutto, che crea l'API REST.

```
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. Apri hello-world-stack.ts. Questo è il codice che definisce lo stack AWS CDK. Sostituisci il codice con il seguente:

```
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. Passa alla directory del progetto.

```
cd hello-world
```

10. Distribuisci l'applicazione.

```
cdk deploy
```

11. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?ExportName==`apiUrl`].OutputValue' --output text
```

12. Richiama l'endpoint dell'API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

13. Per ottenere i log per la funzione, esegui sam logs. Per ulteriori informazioni, consulta l'argomento relativo all'utilizzo dei log nella Guida per sviluppatori AWS Serverless Application Model .

```
sam logs --stack-name HelloWorldStack
```

L'output del log ha la struttura seguente:

```
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. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
cdk destroy
```

Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor del codice incorporato, troverai i log nei risultati dell’esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l’output del log nella sezione Dettagli.

Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (***your-function-name***/aws/lambda/).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Viene visualizzato un flusso di log quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire le chiamate simultanee. Per trovare i log per una chiamata specifica, consigliamo di strumentare la funzione con. AWS X-Ray X-Ray registra i dettagli sulla richiesta e il flusso di log nella traccia.

TypeScript Codice di tracciamento in AWS Lambda

Lambda si integra con AWS X-Ray per aiutarti a tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzare X-Ray per tracciare una richiesta mentre attraversa le risorse nell'applicazione, che possono includere funzioni Lambda e altri servizi AWS .

Per inviare dati di tracciamento a X-Ray, è possibile utilizzare una delle tre librerie SDK:

- AWS Distro for OpenTelemetry (ADOT): una distribuzione sicura, pronta per la produzione e supportata dell'SDK (). AWS OpenTelemetry OTel
- AWS X-Ray SDK per Node.js: un SDK per la generazione e l'invio di dati di traccia a X-Ray.
- Powertools for AWS Lambda (TypeScript): un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori.

Ciascuno di essi SDKs offre modi per inviare i dati di telemetria al servizio X-Ray. Puoi quindi utilizzare X-Ray per visualizzare, filtrare e analizzare le metriche delle prestazioni dell'applicazione per identificare i problemi e le opportunità di ottimizzazione.

> **Important**
>
> X-Ray e Powertools per AWS Lambda SDKs fanno parte di una soluzione di strumentazione strettamente integrata offerta da. AWS I livelli Lambda ADOT fanno parte di uno standard di settore per la strumentazione di tracciamento che in generale raccoglie più dati, ma potrebbero non essere adatti a tutti i casi d'uso. È possibile implementare il end-to-end tracciamento in X-Ray utilizzando entrambe le soluzioni. Per saperne di più sulla scelta tra di esse, consulta Scelta tra AWS Distro for Open Telemetry e X-Ray. SDKs

Sections

- Utilizzo di Powertools per () e per il tracciamento AWS Lambda TypeScript AWS SAM
- Usare Powertools for AWS Lambda (TypeScript) e the AWS CDK per tracciare
- Interpretazione di una traccia X-Ray

Utilizzo di Powertools per () e per il tracciamento AWS Lambda TypeScript AWS SAM

Segui i passaggi seguenti per scaricare, creare e distribuire un' TypeScript applicazione Hello World di esempio con i moduli Powertools for AWS Lambda (TypeScript) integrati utilizzando il. AWS SAM Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio `hello world`.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Node.js
- AWS CLI versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS SAM

1. Inizializza l'applicazione utilizzando il modello Hello World. TypeScript

```
sam init --app-template hello-world-powertools-typescript --name sam-app --package-type Zip --runtime nodejs24.x --no-tracing
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi `Enter`.

> **Note**
>
> Perché HelloWorldFunction potrebbe non avere un'autorizzazione definita. Va bene? , assicurati di entrarey.

5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Richiama l'endpoint dell'API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere le tracce per la funzione, esegui sam traces.

```
sam traces
```

L'output della traccia ha il seguente aspetto:

```
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. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non è possibile configurare la frequenza di campionamento di X-Ray per le funzioni.

Usare Powertools for AWS Lambda (TypeScript) e the AWS CDK per tracciare

Segui i passaggi seguenti per scaricare, creare e distribuire un' TypeScript applicazione Hello World di esempio con moduli Powertools for AWS Lambda (TypeScript) integrati utilizzando. AWS CDK Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio `hello world`.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Node.js
- AWS CLI versione 2
- AWS CDK versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS Cloud Development Kit (AWS CDK)

1. Crea una directory di progetto per la nuova applicazione.

```
mkdir hello-world
cd hello-world
```

2. Inizializza l'app.

```
cdk init app --language typescript
```

3. Aggiungi il pacchetto @types /aws-lambda come dipendenza di sviluppo.

```
npm install -D @types/aws-lambda
```

4. Installa l'utilità Tracer di Powertools.

```
npm install @aws-lambda-powertools/tracer
```

5. Apri la directory lib. Dovresti vedere un file chiamato hello-world-stack.ts. Crea due nuovi file in questa directory: hello-world.function.ts e hello-world.ts.

6. Apri hello-world.function.ts e aggiungi il seguente codice al file. Questo è il codice per la funzione 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. Apri hello-world.ts e aggiungi il seguente codice al file. Contiene il NodejsFunction costrutto che crea la funzione Lambda, configura le variabili di ambiente per Powertools e imposta la conservazione dei log su una settimana. Include anche il LambdaRestApi costrutto, che crea l'API REST.

```
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. Apri hello-world-stack.ts. Questo è il codice che definisce lo stack AWS CDK. Sostituisci il codice con il seguente:

```
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. Distribuisci l'applicazione.

```
cd ..
cdk deploy
```

10. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?ExportName==`apiUrl`].OutputValue' --output text
```

11. Richiama l'endpoint dell'API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

12. Per ottenere le tracce per la funzione, esegui sam traces.

```
sam traces
```

L'output della traccia ha il seguente aspetto:

```
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]
```

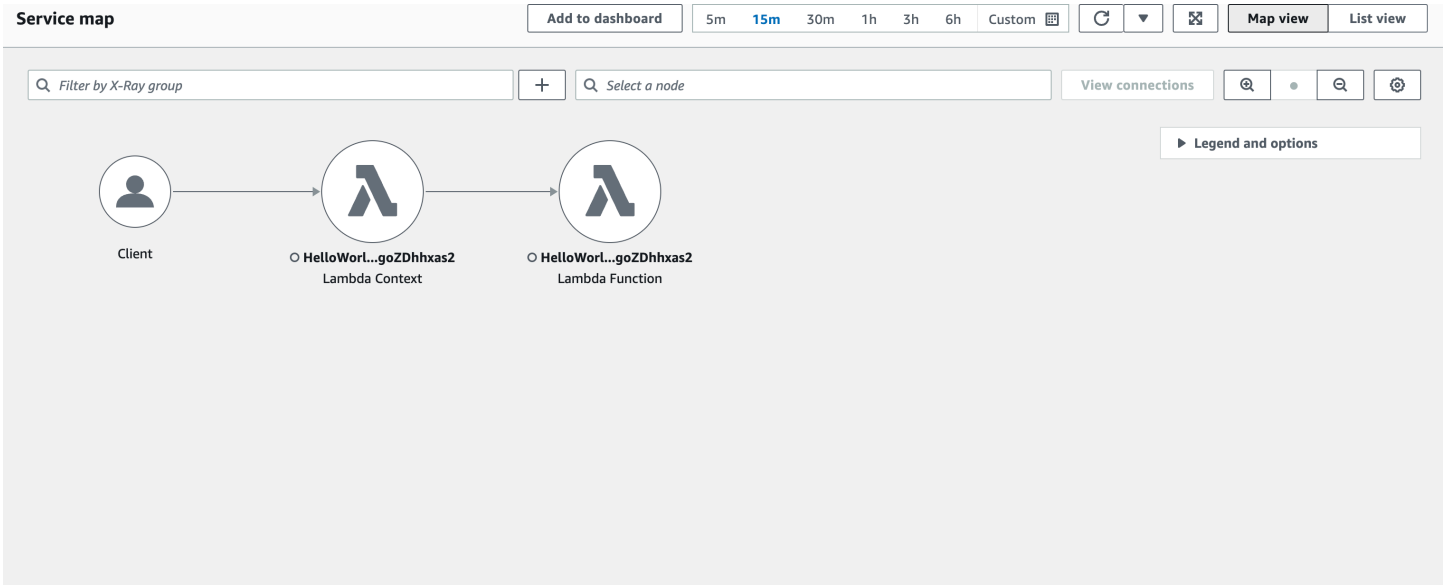
```
  - 0.157s - HelloWorldStack-helloworldfunction111A2BCD-Xyzv11a1bcde
    - 0.169s - Initialization
    - 0.058s - Invocation
      - 0.055s - ## index.handler
        - 0.000s - ### MySubSegment
    - 0.099s - Overhead
```

13. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
cdk destroy
```

Interpretazione di una traccia X-Ray

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. La mappa del tracciamento X-Ray fornisce informazioni sull'applicazione e su tutti i relativi componenti. L'esempio seguente mostra una traccia dall'applicazione di esempio:



Compilazione di funzioni Lambda con Python

Puoi eseguire il codice Python in AWS Lambda. Lambda fornisce Runtime per Python che eseguono il tuo codice per elaborare gli eventi. Il codice viene eseguito in un ambiente che include l'SDK for Python (Boto3), con le credenziali AWS Identity and Access Management di un ruolo (IAM) che gestisci. Per ulteriori informazioni sulle versioni SDK incluse nei runtime Python, consulta the section called “Versioni SDK incluse nel runtime”.

Lambda supporta i seguenti runtime di Python.

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Python 3.14	`python3.14`	Amazon Linux 2023	30 giugno 2029	31 luglio 2029	31 agosto 2029
Python 3.13	`python3.13`	Amazon Linux 2023	30 giugno 2029	31 luglio 2029	31 agosto 2029
Python 3.12	`python3.12`	Amazon Linux 2023	31 ottobre 2028	30 novembre 2028	10 gennaio 2029
Python 3.11	`python3.11`	Amazon Linux 2	30 giugno 2027	31 luglio 2027	31 agosto 2027
Python 3.10	`python3.10`	Amazon Linux 2	31 ottobre 2026	30 novembre 2026	15 gennaio 2027

Per creare una funzione Python

1. Aprire la console Lambda.
2. Scegli Crea funzione.
3. Configura le impostazioni seguenti:
   - Nome della funzione: inserisci il nome della funzione.
   - Runtime: scegli Python 3.14.

4. Scegli Crea funzione.

La console crea una funzione Lambda con un singolo file di origine denominato `lambda_function`. È possibile modificare questo file e aggiungere altri file nell'editor di codice predefinito. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione. Quindi, per eseguire il codice, scegli Crea evento di test nella sezione EVENTI DI TEST.

La funzione Lambda include un gruppo di CloudWatch log Logs. Il runtime della funzione invia i dettagli su ogni chiamata a Logs. CloudWatch Si trasmette qualsiasi log che la tua funzione emette durante la chiamata. Se la funzione restituisce un errore, Lambda formatta l'errore e lo restituisce al chiamante.

Argomenti

- Versioni SDK incluse nel runtime
- Funzionalità Python disattivate
- Formato della risposta
- Chiusura graduale per le estensioni
- Definire l'handler della funzione Lambda in Python
- Utilizzo di archivi di file .zip per le funzioni Lambda in Python
- Distribuisci funzioni Lambda per Python con immagini di container
- Utilizzo dei livelli per le funzioni Lambda in Python
- Utilizzo dell'oggetto del contesto Lambda per recuperare le informazioni sulla funzione Python
- Registrare e monitorare le funzioni Lambda con Python
- Test delle funzioni AWS Lambda in Python
- Strumentazione del codice Python in AWS Lambda

Versioni SDK incluse nel runtime

La versione dell' AWS SDK inclusa nel runtime di Python dipende dalla versione di runtime e dalla tua. Regione AWS Per trovare la versione dell'SDK incluso nel runtime che stai utilizzando, crea una funzione Lambda con il codice seguente.

```
import boto3
import botocore
```

```
def lambda_handler(event, context):
   print(f'boto3 version: {boto3.__version__}')
   print(f'botocore version: {botocore.__version__}')
```

Funzionalità Python disattivate

La tabella seguente elenca le funzionalità di Python che sono disabilitate nei runtime gestiti da Lambda e nelle immagini base dei contenitori per Python. Queste funzionalità devono essere abilitate quando l'eseguibile del runtime Python è compilato e non possono essere abilitate utilizzando un flag in fase di esecuzione. Per utilizzare queste funzionalità in Lambda, puoi distribuire la tua build di runtime Python con queste funzionalità abilitate, utilizzando un'immagine del contenitore o un runtime personalizzato.

Funzionalità Python	Versioni di Python interessate	Status
Just-in-Time compilatore (JIT)	Python 3.13 e versioni successive	Il compilatore JIT è sperimentale e non è consigliato per carichi di lavoro di produzione. È quindi disabilitato nei runtime Lambda Python.
Threading libero	Python 3.13 e versioni successive	Il threading libero (opzione per disabilitare il blocco globale dell'interprete) è disabilitato nelle build Lambda Python a causa dell'impatto sulle prestazioni sul codice a thread singolo.

Formato della risposta

Nei runtime Python 3.12 e successivi, le funzioni restituiscono caratteri Unicode come parte della loro risposta JSON. I runtime Python precedenti restituivano sequenze con escape per i caratteri Unicode nelle risposte. Ad esempio, in Python 3.11, se restituisci una stringa Unicode come "こんにちは", viene eseguito l'escape dei caratteri Unicode e restituito il valore "\u3053\u3093\u306b\u3061\u306f". Il runtime Python 3.12 restituisce il valore "こんにちは" originale.

L'utilizzo delle risposte Unicode riduce le dimensioni delle risposte Lambda e ciò facilita l'inserimento di risposte più grandi nella dimensione del payload massima di 6 MB per le funzioni sincrone. Nell'esempio precedente, la versione con escape è di 32 byte, rispetto ai 17 della stringa Unicode.

Quando esegui l'aggiornamento a Python 3.12 o a runtime Python successivi, potrebbe essere necessario modificare il codice tenendo conto del nuovo formato di risposta. Se il chiamante prevede un codice Unicode con escape, devi aggiungere manualmente il codice alla funzione restituita per eseguire l'escape dell'Unicode o regolare il chiamante per gestire la restituzione dell'Unicode.

Chiusura graduale per le estensioni

I runtime Python 3.12 e versioni successive offrono funzionalità di arresto graduale migliorate per funzioni con estensioni esterne. Quando chiude un ambiente di esecuzione, Lambda invia un segnale `SIGTERM` al runtime e quindi un evento `SHUTDOWN` a ogni estensione esterna registrata. È possibile catturare il segnale `SIGTERM` nella funzione Lambda e ripulire risorse come le connessioni al database create dalla funzione.

Per ulteriori informazioni sul ciclo di vita dell'ambiente di esecuzione, consulta Comprendere il ciclo di vita dell'ambiente di esecuzione Lambda. Per esempi di come usare Graceful Shutdown con le estensioni, consulta il repository Samples.AWS GitHub

Definire l'handler della funzione Lambda in Python

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Questa pagina descrive come lavorare con i gestori di funzioni Lambda in Python, incluse le convenzioni di denominazione, le firme degli handler e le migliori pratiche. Questa pagina include anche un esempio di funzione Go Lambda che raccoglie informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon Simple Storage Service (S3).

Argomenti

- Esempio di codice della funzione Lambda in Python
- Convenzioni di denominazione dei gestori
- L'oggetto evento Lambda:
- Accesso e utilizzo dell'oggetto contestuale Lambda
- Firme dell'handler valide per gli handler Python
- Restituzione di un valore
- Utilizzo della AWS SDK per Python (Boto3) versione v2 nell'handler
- Accesso alle variabili d'ambiente
- Best practice di codice per le funzioni Lambda con Python Lambda

Esempio di codice della funzione Lambda in Python

Il seguente esempio di codice della funzione Lambda in Python raccoglie le informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon S3.

Example funzione 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
```

Questo file contiene le sezioni seguenti:

- Blocco `import`: utilizza questo blocco per includere le librerie richieste dalla funzione Lambda.
- Inizializzazione globale del client e del logger SDK: l'inclusione del codice di inizializzazione all'esterno del gestore sfrutta il riutilizzo dell'ambiente di esecuzione per migliorare le prestazioni della funzione. Per ulteriori informazioni, consulta the section called “Best practice di codice per le funzioni Lambda con Python Lambda”.
- `def upload_receipt_to_s3(bucket_name, key, receipt_content):`Questa è una funzione di supporto chiamata dalla funzione principale. `lambda_handler`
- `def lambda_handler(event, context):`: questo è il metodo dell'handler principale, che contiene la logica principale dell'applicazione. Quando Lambda richiama il gestore di funzioni, il runtime Lambda passa due argomenti alla funzione, l'oggetto evento che contiene i dati che la funzione deve elaborare e l'oggetto context che contiene informazioni sulla chiamata della funzione.

Convenzioni di denominazione dei gestori

Il nome del gestore di funzioni Lambda specificato al momento della creazione di una funzione Lambda deriva da quanto segue:

- il nome del file in cui si trova la funzione del gestore Lambda.
- il nome della funzione del gestore Python.

Nell'esempio precedente, se il file è denominato`lambda_function.py`, il gestore viene specificato come. `lambda_function.lambda_handler` Questo è il nome del gestore predefinito assegnato alle funzioni create utilizzando la console Lambda.

Se si crea una funzione nella console utilizzando un nome di file o un nome del gestore di funzione diverso, è necessario modificare il nome del gestore predefinito.

Modifica del nome del gestore funzioni (console)

1. Apri la pagina Funzioni della console Lambda e scegli la tua funzione.
2. Scegli la scheda Codice.
3. Scorri verso il basso fino al riquadro Impostazioni di runtime e scegli Modifica.
4. In Gestore, inserisci il nuovo nome per il tuo gestore di funzioni.
5. Selezionare Salva.

L'oggetto evento Lambda:

Quando la funzione viene richiamata da Lambda, passa un argomento dell'oggetto evento al gestore di funzioni. Gli oggetti JSON sono il formato di eventi più comune per le funzioni Lambda. Nell'esempio di codice riportato nella sezione precedente, la funzione prevede un input nel seguente formato:

```
{
    "Order_id": "12345",
    "Amount": 199.99,
    "Item": "Wireless Headphones"
}
```

Se la funzione è richiamata da un altroServizio AWS, l'evento di input è anche un oggetto JSON. Il formato esatto dell'oggetto evento dipende dal servizio che richiama la funzione. Per vedere il formato dell'evento per un particolare servizio, consultate la pagina appropriata nel Integrazione con altri servizi capitolo.

Se l'evento di input ha la forma di un oggetto JSON, il runtime Lambda converte l'oggetto in un dizionario Python. Per assegnare valori nell'input JSON alle variabili del codice, usa i metodi del dizionario Python standard come illustrato nel codice di esempio.

Puoi anche passare dati alla tua funzione come array JSON o come qualsiasi altro tipo di dati JSON validi. La tabella seguente definisce come il runtime Python converte questi tipi JSON.

Tipo di dati JSON	Tipo di dati Python
oggetto	Dizionario`dict`
array	elenco`list`
number	numero intero (`int`) o numero in virgola mobile () `float`
string	Stringa (`str`)
Booleano	booleano (`bool`)
null	noeType () `NoneType`

Accesso e utilizzo dell'oggetto contestuale Lambda

L'oggetto contesto: contiene informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. Lambda passa l'oggetto contestuale alla tua funzione automaticamente. Puoi utilizzare l'oggetto contestuale per generare informazioni sull'invocazione della funzione a scopo di monitoraggio.

L'oggetto context è una classe Python definita nel client dell'interfaccia runtime Lambda. Per restituire il valore di una qualsiasi delle proprietà dell'oggetto contestuale, utilizzate il metodo corrispondente sull'oggetto context. Ad esempio, il frammento di codice seguente assegna il valore della `aws_request_id` proprietà (l'identificatore per la richiesta di chiamata) a una variabile denominata. `request`

```
request = context.aws_request_id
```

Per ulteriori informazioni sull'utilizzo dell'oggetto contestuale Lambda e per visualizzare un elenco completo dei metodi e delle proprietà disponibili, vedere. the section called “Contesto”

Firme dell'handler valide per gli handler Python

Quando si definisce la funzione di gestione in Python, la funzione deve accettare due argomenti. Il primo di questi argomenti è l'oggetto evento Lambda e il secondo è l'oggetto contesto Lambda. Per convenzione, questi argomenti di input sono generalmente denominati `event` `econtext`, ma puoi assegnare loro tutti i nomi che desideri. Se dichiari la tua funzione di gestione con un solo argomento di input, Lambda genererà un errore quando tenta di eseguire la funzione. Il modo più comune per dichiarare una funzione di gestione in Python è il seguente:

```
def lambda_handler(event, context):
```

È inoltre possibile utilizzare i suggerimenti di tipo Python nella dichiarazione di funzioni, come illustrato nell'esempio seguente:

```
from typing import Dict, Any

def lambda_handler(event: Dict[str, Any], context: Any) -> Dict[str, Any]:
```

Per utilizzare una AWS digitazione specifica per eventi generati da altri oggetti Servizi AWS e per l'oggetto di contesto, aggiungi il `aws-lambda-typing` pacchetto al pacchetto di distribuzione della funzione. Puoi installare questa libreria nel tuo ambiente di sviluppo **`pip install aws-lambda-typing`** eseguendo. Il seguente frammento di codice mostra come utilizzare suggerimenti di tipo AWS specifici. In questo esempio, l'evento previsto è un evento 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]:
```

Non puoi usare il tipo di funzione Python per la tua `async` funzione di gestione.

Restituzione di un valore

È inoltre possibile che il gestore restituisca un valore, che deve essere serializzabile in formato JSON. I tipi di restituzione più comuni includono`dict`,,`list`, `estr`. `int` `float` `bool`

Ciò che accade al valore restituito dipende dal tipo di chiamata e dal servizio che ha invocato la funzione. Ad esempio:

- Se si utilizza il tipo di invocazione `RequestResponse`, come , restituisce il risultato della chiamata della funzione Python al client che invoca la funzione Lambda (nella risposta HTTP alla richiesta di invocazione, serializzata in JSON). Ad esempio, la console AWS Lambda utilizza il tipo di invocazione `RequestResponse`; quindi, quando si invoca la funzione mediante la console, in quest'ultima verrà visualizzato il valore restituito.
- Se il gestore restituisce degli oggetti che non possono essere serializzati da `json.dumps`, il runtime restituisce un errore.
- Se il gestore restituisce `None`, come fanno implicitamente le funzioni Python senza un'istruzione `return`, il runtime restituisce `null`.
- Se utilizzi il tipo di invocazione `Event`, ovvero una invocazione asincrona, il valore viene ignorato.

Nel codice di esempio, il gestore restituisce il seguente dizionario Python:

```
{
  "statusCode": 200,
  "message": "Receipt processed successfully"
}
```

Il runtime Lambda serializza questo dizionario e lo restituisce al client che ha richiamato la funzione come stringa JSON.

> **Note**
>
> In Python 3.9 e nelle versioni successive, Lambda include il requestId dell'invocazione nella risposta di errore.

Utilizzo della AWS SDK per Python (Boto3) versione v2 nell'handler

Spesso, utilizzerai le funzioni Lambda per interagire o aggiornare altre Servizi AWS risorse. Il modo più semplice per interfacciarsi con queste risorse è usare la AWS SDK per Python (Boto3) v2. Tutti i runtime Lambda Python supportati includono una versione dell'SDK per Python. Tuttavia, consigliamo vivamente di includere l'SDK nel pacchetto di distribuzione della funzione se il codice deve utilizzarlo. L'inclusione dell'SDK nel pacchetto di distribuzione offre il pieno controllo sulle dipendenze e riduce il rischio di problemi di disallineamento delle versioni con altre librerie. Per ulteriori informazioni, consulta le pagine the section called “Dipendenze di runtime in Python” e the section called “Compatibilità con le versioni precedenti”.

Per usare l'SDK per Python nella tua funzione Lambda, aggiungi la seguente istruzione al blocco di importazione all'inizio del codice della funzione:

```
import boto3
```

Usa il `pip install` comando per aggiungere la `boto3` libreria al pacchetto di distribuzione della funzione. Per istruzioni dettagliate su come aggiungere dipendenze a un pacchetto di implementazione .zip, vedere. the section called “Creazione di un pacchetto di implementazione .zip con dipendenze” Per ulteriori informazioni sull'aggiunta di dipendenze alle funzioni Lambda distribuite come immagini di container, consulta o. the section called “Creazione di un'immagine da un'immagine di base” the section called “Creazione di un'immagine da un'immagine di base alternativa”

Quando `boto3` si utilizza il codice, non è necessario fornire alcuna credenziale per inizializzare un client. Ad esempio, nel codice di esempio, utilizziamo la seguente riga di codice per inizializzare un client Amazon S3:

```
# Initialize the S3 client outside of the handler
s3_client = boto3.client('s3')
```

Con Python, Lambda crea automaticamente variabili di ambiente con credenziali. L'`boto3`SDK controlla le variabili di ambiente della funzione per verificare la presenza di queste credenziali durante l'inizializzazione.

Accesso alle variabili d'ambiente

Nel codice dell'handler, puoi fare riferimento a qualsiasi variabile di ambiente utilizzando il metodo `os.environ.get`. In questo esempio, facciamo riferimento alla variabile di `RECEIPT_BUCKET` ambiente definita utilizzando la seguente riga di codice:

```
# Access environment variables
bucket_name = os.environ.get('RECEIPT_BUCKET')
```

Non dimenticate di includere un'`import os`istruzione nel blocco di importazione all'inizio del codice.

Best practice di codice per le funzioni Lambda con Python Lambda

Segui le linee guida riportate nell'elenco seguente per utilizzare le best practice di codifica durante la creazione delle funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. In questo modo è possibile creare una funzione di cui è più semplice eseguire l'unit test. Ad esempio, in Python, l'aspetto è analogo al seguente:

```
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
```

- Controllare le dipendenze nel pacchetto di distribuzione della funzione. L'ambiente di esecuzione AWS Lambda contiene diverse librerie. Per i runtime Node.js e Python, questi includono gli SDK AWS. Per abilitare il set di caratteristiche e aggiornamenti della sicurezza più recenti, Lambda aggiorna periodicamente tali librerie. Tali aggiornamenti possono introdurre lievi modifiche al comportamento della funzione Lambda. Per mantenere il controllo completo delle dipendenze utilizzate dalla funzione, inserire tutte le dipendenze nel pacchetto di implementazione.
- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del contesto di esecuzione.
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. In questo modo viene ridotta la quantità di tempo necessaria per il download del pacchetto e per la relativa decompressione prima dell'invocazione.

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente,

utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedere Riutilizzo delle connessioni con Keep-Alive in Node.js.

Utilizzare le variabili di ambiente per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su `0` per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare API non documentate e non pubbliche nel codice della funzione Lambda. Per i tempi di esecuzione gestiti AWS Lambda, Lambda applica periodicamente aggiornamenti di sicurezza e funzionalità alle API interne di Lambda. Questi aggiornamenti API interni potrebbero essere incompatibili con le versioni precedenti, causando conseguenze indesiderate come errori di chiamata se la funzione ha una dipendenza su queste API non pubbliche. Consulta il riferimento all'API per un elenco di API disponibili pubblicamente.

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda?.

Utilizzo di archivi di file .zip per le funzioni Lambda in Python

Il codice della AWS Lambda funzione comprende un file.py contenente il codice del gestore della funzione, insieme a tutti i pacchetti e i moduli aggiuntivi da cui dipende il codice. Per implementare questo codice della funzione in Lambda, utilizza un pacchetto di implementazione. Questo pacchetto può essere un archivio di file .zip o un'immagine di container. Per ulteriori informazioni sull'uso delle immagini di container con Python, consulta la sezione Implementazione di funzioni Lambda in Python con immagini di container.

Per creare un pacchetto di implementazione come archivio di file .zip, puoi utilizzare l'utilità di archiviazione di file .zip incorporata del tuo strumento della linea di comando o qualsiasi altra utilità file .zip, come ad esempio 7zip. Gli esempi mostrati nelle sezioni seguenti presuppongono che tu stia utilizzando uno strumento della linea di comando `zip` in un ambiente Linux o MacOS. Per utilizzare gli stessi comandi in Windows, puoi installare il sottosistema Windows per Linux per ottenere una versione di Ubuntu e Bash integrata con Windows.

Nota che Lambda utilizza le autorizzazioni dei file POSIX, quindi potresti aver bisogno di impostare le autorizzazioni per la cartella del pacchetto di implementazione prima di creare l'archivio di file .zip.

Argomenti

- Dipendenze di runtime in Python
- Creazione di un pacchetto di implementazione .zip senza dipendenze
- Creazione di un pacchetto di implementazione .zip con dipendenze
- Percorso di ricerca delle dipendenze e librerie incluse nel runtime
- Utilizzo delle cartelle __pycache__
- Creazione di un pacchetto di implementazione .zip con librerie native
- Creazione e aggiornamento delle funzioni Lambda di Python utilizzando file .zip

Dipendenze di runtime in Python

Per le funzioni Lambda che utilizzano il runtime Python, una dipendenza può essere qualsiasi pacchetto o modulo Python. Se implementi la funzione utilizzando un archivio .zip, puoi aggiungere queste dipendenze al file .zip con il tuo codice della funzione o utilizzare un livello Lambda. Un livello è un file .zip separato che può contenere codice aggiuntivo o altri contenuti. Per ulteriori informazioni sull'uso dei livelli Lambda in Python, consulta the section called “Livelli”.

I runtime Lambda Python includono e le relative dipendenze. AWS SDK per Python (Boto3) Lambda fornisce l'SDK nel runtime per scenari di implementazione in cui non è possibile aggiungere le proprie dipendenze. Questi scenari includono la creazione di funzioni nella console utilizzando l'editor di codice integrato o l'utilizzo di funzioni inline in AWS Serverless Application Model () o modelli.AWS SAM CloudFormation

Lambda aggiorna periodicamente le librerie nel runtime Python per includere gli aggiornamenti e le patch di sicurezza più recenti. Se la funzione utilizza la versione dell'SDK Boto3 inclusa nel runtime ma il pacchetto di implementazione include dipendenze SDK, ciò può causare problemi di disallineamento delle versioni. Ad esempio, il pacchetto di implementazione potrebbe includere la dipendenza SDK urllib3. Quando Lambda aggiorna l'SDK nel runtime, i problemi di compatibilità tra la nuova versione del runtime e la versione di urllib3 nel pacchetto di implementazione possono causare il fallimento della funzione.

> **Important**
>
> Per mantenere il pieno controllo sulle dipendenze ed evitare possibili problemi di disallineamento delle versioni, si consiglia di aggiungere tutte le dipendenze della funzione al pacchetto di implementazione, anche se le relative versioni sono incluse nel runtime Lambda. Ciò include l'SDK Boto3.

Per scoprire quale versione dell'SDK per Python (Boto3) è inclusa nel runtime che stai utilizzando, consulta the section called “Versioni SDK incluse nel runtime”.

In base al modello di responsabilità condivisa di AWS, è tua responsabilità gestire eventuali dipendenze nei pacchetti di implementazione delle tue funzioni. Ciò include l'applicazione di aggiornamenti e patch di sicurezza. Per aggiornare le dipendenze nel pacchetto di implementazione della funzione, crea prima un nuovo file .zip e poi caricalo su Lambda. Per ulteriori informazioni, consulta Creazione di un pacchetto di implementazione .zip con dipendenze e Creazione e aggiornamento delle funzioni Lambda di Python utilizzando file .zip.

Creazione di un pacchetto di implementazione .zip senza dipendenze

Se il codice della funzione non ha dipendenze, il file .zip contiene solo il file .py con il codice del gestore della funzione. Usa la tua utilità zip preferita per creare un file .zip con il file .py nella directory principale. Se il file .py non si trova nella directory principale del file .zip, Lambda non sarà in grado di eseguire il codice.

Per informazioni su come implementare il file .zip per creare una nuova funzione Lambda o aggiornarne una esistente, consulta la sezione Creazione e aggiornamento delle funzioni Lambda di Python utilizzando file .zip.

Creazione di un pacchetto di implementazione .zip con dipendenze

Se il tuo codice della funzione dipende da pacchetti o moduli aggiuntivi, puoi aggiungere queste dipendenze al file .zip con il codice della funzione oppure utilizzare un livello Lambda. Le istruzioni in questa sezione mostrano come includere le dipendenze nel pacchetto di implementazione .zip. Affinché Lambda esegua il codice, il file .py contenente il codice del gestore e tutte le dipendenze della funzione deve essere installato nella radice del file .zip.

Supponiamo che il codice della funzione sia salvato in un file denominato `lambda_function.py`. I seguenti comandi della CLI di esempio creano un file .zip denominato `my_deployment_package.zip` contenente il codice della funzione e le relative dipendenze. Puoi installare le tue dipendenze direttamente in una cartella nella directory del tuo progetto o utilizzare un ambiente virtuale Python.

Creazione del pacchetto di implementazione (directory del progetto)

1. Passa alla directory del progetto contenente il file del codice sorgente `lambda_function.py`. In questo esempio, la directory è denominata `my_function`.

   ```
   cd my_function
   ```

2. Crea una nuova directory denominata "pacchetto" in cui installare le tue dipendenze.

   ```
   mkdir package
   ```

   Tieni presente che per un pacchetto di implementazione .zip, Lambda prevede che il codice sorgente e le relative dipendenze siano tutti nella directory principale del file .zip. Tuttavia, l'installazione delle dipendenze direttamente nella directory del progetto può introdurre un gran numero di nuovi file e cartelle e rendere difficile la navigazione nell'IDE. Qui crei una directory `package` separata per mantenere le dipendenze separate dal codice sorgente.

3. Installa le dipendenze nella directory `package`. L'esempio seguente installa l'SDK Boto3 da Python Package Index utilizzando pip. Se il codice della funzione utilizza pacchetti Python creati da te, salvali nella directory `package`.

```
pip install --target ./package boto3
```

4. Crea un file .zip con le librerie installate nella directory principale.

```
cd package
zip -r ../my_deployment_package.zip .
```

Verrà generato un file `my_deployment_package.zip` nella directory del progetto.

5. Aggiunta del file lambda_function.py alla directory principale del file .zip

```
cd ..
zip my_deployment_package.zip lambda_function.py
```

Il tuo file .zip dovrebbe avere una struttura di directory semplice, con il codice del gestore della funzione e tutte le cartelle delle dipendenze installate nella directory principale come segue.

```
my_deployment_package.zip
|- bin
|  |-jp.py
|- boto3
|  |-compat.py
|  |-data
|  |-docs
...
|- lambda_function.py
```

Se il file .py contenente il codice del gestore della funzione non si trova nella directory principale del file .zip, Lambda non sarà in grado di eseguire il codice.

Creazione del pacchetto di implementazione (ambiente virtuale)

1. Crea e attiva un ambiente virtuale nella directory del progetto. In questo esempio, la directory del progetto è denominata `my_function`.

```
~$ cd my_function
~/my_function$ python3.14 -m venv my_virtual_env
~/my_function$ source ./my_virtual_env/bin/activate
```

2. Installa le librerie richieste utilizzando pip. Nell'esempio seguente viene installato l'SDK Boto3

```
(my_virtual_env) ~/my_function$ pip install boto3
```

3. Utilizza `pip show` per trovare la posizione, all'interno del tuo ambiente virtuale, in cui pip ha installato le tue dipendenze.

```
(my_virtual_env) ~/my_function$ pip show <package_name>
```

   La cartella in cui pip installa le tue librerie può essere denominata `site-packages` oppure `dist-packages`. Questa cartella può trovarsi nella directory `lib/python3.x` oppure `lib64/python3.x` (dove python3.x rappresenta la versione di Python che stai utilizzando).

4. Disattivazione dell'ambiente virtuale

```
(my_virtual_env) ~/my_function$ deactivate
```

5. Accedi alla directory contenente le dipendenze che hai installato con pip e crea un file .zip nella directory principale del tuo progetto, nella quale sono installate le dipendenze. In questo esempio, pip ha installato le tue dipendenze nella directory `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. Vai alla directory principale del tuo progetto, in cui si trova il file .py contenente il codice del gestore, e aggiungi quel file alla directory principale del tuo pacchetto .zip. In questo esempio, il file di codice della funzione è denominato `lambda_function.py`.

```
~/my_function/my_virtual_env/lib/python3.14/site-packages$ cd ../../../../
~/my_function$ zip my_deployment_package.zip lambda_function.py
```

Percorso di ricerca delle dipendenze e librerie incluse nel runtime

Quando utilizzi un'istruzione `import` nel codice, il runtime Python cerca nelle directory del suo percorso di ricerca finché non trova il modulo o il pacchetto. Per impostazione predefinita, la prima posizione cercata dal runtime è la directory in cui il pacchetto di implementazione .zip viene decompresso e montato (`/var/task`). Se includi una versione di una libreria inclusa nel runtime

nel tuo pacchetto di implementazione, questa versione avrà la precedenza sulla versione inclusa nel runtime. Le dipendenze nel pacchetto di implementazione hanno la precedenza anche sulle dipendenze nei livelli.

Quando aggiungi una dipendenza a un livello, Lambda la estrae in `/opt/python/lib/python3.x/site-packages`, dove `python3.x` rappresenta la versione del runtime che stai utilizzando, o `/opt/python`. Nel percorso di ricerca, queste directory hanno la precedenza sulle directory contenenti le librerie incluse nel runtime e le librerie installate con pip (`/var/runtime` e `/var/lang/lib/python3.x/site-packages`). Le librerie nei livelli di funzione hanno quindi la precedenza sulle versioni incluse nel runtime.

> **Note**
>
> Nel runtime gestito e nell'immagine di base di Python 3.11, l' AWS SDK e le sue dipendenze sono installati nella directory. `/var/lang/lib/python3.11/site-packages`

Puoi visualizzare il percorso di ricerca completo per la tua funzione Lambda aggiungendo il seguente frammento di codice.

```
import sys

search_path = sys.path
print(search_path)
```

> **Note**
>
> Poiché le dipendenze nei livelli o nel pacchetto di implementazione hanno la precedenza sulle librerie incluse nel runtime, ciò può causare problemi di disallineamento delle versioni se si include una dipendenza SDK come urllib3 nel pacchetto senza includere anche l'SDK. Se implementi la tua versione di una dipendenza Boto3, devi anche implementare Boto3 come dipendenza nel tuo pacchetto di implementazione. Ti consigliamo di impacchettare tutte le dipendenze della tua funzione, anche se le rispettive versioni sono già incluse nel runtime.

Puoi anche aggiungere dipendenze in una cartella separata all'interno del tuo pacchetto .zip. Ad esempio, potresti aggiungere una versione dell'SDK Boto3 a una cartella del tuo pacchetto .zip chiamata `common`. Quando il pacchetto .zip viene decompresso e montato, questa cartella viene

inserita nella directory `/var/task`. Per utilizzare nel codice una dipendenza da una cartella del pacchetto di implementazione .zip, utilizza un'istruzione `import from`. Ad esempio, per utilizzare una versione di Boto3 da una cartella denominata `common` nel tuo pacchetto .zip, usa la seguente istruzione.

```
from common import boto3
```

Utilizzo delle cartelle __pycache__

È consigliabile non includere cartelle `__pycache__` nel pacchetto di implementazione della funzione. Il bytecode Python compilato su un computer di compilazione con un'architettura o un sistema operativo diverso potrebbe non essere compatibile con l'ambiente di esecuzione Lambda.

Creazione di un pacchetto di implementazione .zip con librerie native

Se la tua funzione utilizza solo pacchetti e moduli Python puri, puoi usare il comando `pip install` per installare le tue dipendenze su qualsiasi computer di compilazione locale e creare il file .zip. Molte librerie Python popolari, tra cui NumPy e Pandas, non sono Python puro e contengono codice scritto in C o C++. Quando aggiungi librerie contenenti codice C/C++ al pacchetto di implementazione, devi creare il pacchetto correttamente per assicurarti che sia compatibile con l'ambiente di esecuzione Lambda.

La maggior parte dei pacchetti disponibili nel Python Package Index (PyPI) sono disponibili come "wheel" (file .whl). Un file .whl è un tipo di file ZIP che contiene una distribuzione compilata con file binari precompilati per un particolare sistema operativo e un'architettura di set di istruzioni. Per rendere il pacchetto di implementazione compatibile con Lambda, è necessario installare il wheel per i sistemi operativi Linux e l'architettura del set di istruzioni della funzione.

Alcuni pacchetti possono essere disponibili solo come distribuzioni di origine. Per questi pacchetti, è necessario compilare e creare personalmente i componenti C/C++.

Per vedere quali distribuzioni sono disponibili per il pacchetto richiesto, procedi come segue:

1. Cerca il nome del pacchetto nella pagina principale di Python Package Index.
2. Seleziona la versione del pacchetto da utilizzare.
3. Scegli Scarica file.

Utilizzo di distribuzioni integrate (wheel)

Per scaricare un wheel compatibile con Lambda, utilizza l'opzione `--platform` in pip.

Se la tua funzione Lambda utilizza l'architettura del set di istruzioni x86_64, esegui il comando `pip install` seguente per installare un wheel compatibile nella tua directory `package`. Sostituisci `--python 3.x` con la versione del runtime Python che stai utilizzando.

```
pip install \
--platform manylinux2014_x86_64 \
--target=package \
--implementation cp \
--python-version 3.x \
--only-binary=:all: --upgrade \
<package_name>
```

Se la funzione utilizza l'architettura del set di istruzioni arm64, esegui il seguente comando. Sostituisci `--python 3.x` con la versione del runtime Python che stai utilizzando.

```
pip install \
--platform manylinux2014_aarch64 \
--target=package \
--implementation cp \
--python-version 3.x \
--only-binary=:all: --upgrade \
<package_name>
```

Utilizzo delle distribuzioni di origine

Se il tuo pacchetto è disponibile solo come distribuzione di origine, la creazione delle librerie C/C++ spetta a te. Per rendere il pacchetto compatibile con l'ambiente di esecuzione Lambda, è necessario crearlo in un ambiente che utilizzi lo stesso sistema operativo Amazon Linux. Puoi farlo creando il tuo pacchetto in un'istanza Amazon Elastic Compute Cloud (Amazon EC2) Linux.

Per informazioni su come avviare e connettersi a un'istanza Amazon EC2 Linux, consulta la Guida introduttiva ad Amazon EC2 nella Amazon EC2 User Guide.

Creazione e aggiornamento delle funzioni Lambda di Python utilizzando file .zip

Dopo aver creato il pacchetto di implementazione .zip, puoi utilizzarlo per creare una nuova funzione Lambda o aggiornarne una esistente. Puoi distribuire il tuo pacchetto.zip utilizzando la console Lambda, l'API Lambda AWS Command Line Interface e l'API Lambda. Puoi anche creare e aggiornare le funzioni Lambda usando AWS Serverless Application Model (AWS SAM) e CloudFormation.

La dimensione massima per un pacchetto di implementazione .zip per Lambda è di 250 MB (dopo l'estrazione). Nota che questo limite si applica alla dimensione combinata di tutti i file caricati, inclusi eventuali livelli Lambda.

Il runtime Lambda necessita dell'autorizzazione per leggere i file nel pacchetto di distribuzione. Nella notazione ottale delle autorizzazioni Linux, Lambda richiede 644 permessi per i file non eseguibili (rw-r--r--) e 755 permessi () per le directory e i file eseguibili. rwxr-xr-x

In Linux e macOS, utilizza il comando `chmod` per modificare le autorizzazioni file su file e directory nel pacchetto di implementazione. Ad esempio, per assegnare a un file non eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 644 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta Set, View, Change, or Remove Permissions on an Object nella documentazione di Microsoft Windows.

> **Note**
>
> Se non concedi a Lambda le autorizzazioni necessarie per accedere alle directory nel pacchetto di distribuzione, Lambda imposta le autorizzazioni per tali directory su 755 (). rwxr-xr-x

Creazione e aggiornamento delle funzioni con file .zip utilizzando la console

Per creare una nuova funzione, devi prima creare la funzione nella console, quindi devi caricare il tuo archivio .zip. Per aggiornare una funzione esistente, apri la pagina relativa alla funzione, quindi segui la stessa procedura per aggiungere il file .zip aggiornato.

Se il file .zip ha dimensioni inferiori a 50 MB, è possibile creare o aggiornare una funzione caricando il file direttamente dal computer locale. Per i file .zip di dimensioni superiori a 50 MB, prima è necessario caricare il pacchetto in un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il Console di gestione AWS, consulta la Guida introduttiva ad Amazon S3. Per caricare file utilizzando la AWS CLI, consulta Move objects nella Guida per l'AWS CLI utente.

> **Note**
>
> Non è possibile modificare il tipo di pacchetto di implementazione (.zip o immagine di container) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine di container esistente per utilizzare un archivio di file .zip. È necessario creare una nuova funzione.

Creazione di una nuova funzione (console)

1. Apri la pagina Funzioni della console Lambda e scegli Crea funzione.
2. Scegli Author from scratch (Crea da zero).
3. In Basic information (Informazioni di base) eseguire queste operazioni:
   a. In Nome funzione, inserisci il nome della funzione.
   b. Per Runtime, seleziona il runtime che desideri utilizzare.
   c. (Facoltativo) Per Architettura, scegli l'architettura del set di istruzioni per la funzione. L'architettura predefinita è x86_64. Assicurati che il pacchetto di implementazione per la tua funzione sia compatibile con l'architettura del set di istruzioni scelta.
4. (Opzionale) In Autorizzazioni espandere Modifica ruolo di esecuzione predefinito. Puoi creare un nuovo ruolo di esecuzione o utilizzare un ruolo esistente.
5. Scegli Crea funzione. Lambda crea una funzione di base "Hello world" utilizzando il runtime scelto.

Caricamento di un archivio .zip dal computer locale (console)

1. Nella pagina Funzioni della console Lambda, scegli la funzione per cui vuoi caricare il file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli File .zip.

5. Per caricare il file .zip, procedi come segue:

    a. Seleziona Carica, quindi seleziona il tuo file .zip nel selettore di file.

    b. Seleziona Apri.

    c. Scegli Save (Salva).

Caricamento di un archivio .zip da un bucket Amazon S3 (console)

1. Nella pagina Funzioni della console Lambda, scegli la funzione per cui vuoi caricare un nuovo file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli Posizione Amazon S3.
5. Incolla l'URL del link Amazon S3 del tuo file .zip e scegli Salva.

Aggiornamento delle funzioni dei file .zip tramite l'editor di codice della console

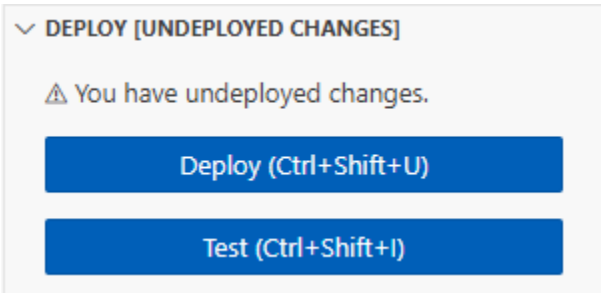
Per alcune funzioni con pacchetti di implementazione .zip, puoi utilizzare l'editor di codice integrato nella console Lambda per aggiornare direttamente il codice della funzione. Per utilizzare questa funzione, la funzione deve soddisfare i seguenti criteri:

- La funzione deve utilizzare uno dei runtime del linguaggio interpretato (Python, Node.js o Ruby)
- Il pacchetto di implementazione della funzione deve avere dimensioni inferiori a 50 MB (non compresso).

Il codice della funzione per le funzioni con pacchetti di implementazione di immagini di container non può essere modificato direttamente nella console.

Aggiornamento del codice della funzione utilizzando l'editor di codice della console

1. Apri la pagina Funzioni della console Lambda e scegli la tua funzione.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, seleziona il tuo file di codice sorgente e modificalo nell'editor di codice integrato.
4. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



Creazione e aggiornamento di funzioni con file.zip utilizzando AWS CLI

È possibile utilizzare la AWS CLI per creare una nuova funzione o aggiornare una funzione esistente mediante un file .zip. Usa la funzione create-function e update-function-codei comandi per distribuire il tuo pacchetto .zip. Se il file .zip ha dimensioni inferiori a 50 MB, è possibile caricare il pacchetto .zip da una posizione di file nella macchina di compilazione locale. Per i file di dimensioni maggiori, è necessario caricare il pacchetto .zip da un bucket Amazon S3. Per istruzioni su come caricare un file su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

> **Note**
>
> Se carichi il tuo file.zip da un bucket Amazon S3 utilizzando AWS CLI il, il bucket deve trovarsi nella stessa posizione della Regione AWS tua funzione.

Per creare una nuova funzione utilizzando un file.zip con AWS CLI, devi specificare quanto segue:

- Il nome della funzione (`--function-name`)
- Il runtime della tua funzione (`--runtime`)
- Il nome della risorsa Amazon (ARN) del ruolo di esecuzione della funzione (`--role`)
- Il nome del metodo del gestore nel codice della funzione (`--handler`)

È inoltre necessario specificare la posizione del file .zip. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
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
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza l'opzione `--code` illustrata nel seguente comando di esempio. È necessario utilizzare il parametro `S3ObjectVersion` solo per gli oggetti con controllo delle versioni.

```
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
```

Per aggiornare una funzione esistente mediante la CLI, specifica il nome della funzione utilizzando il parametro `--function-name`. È inoltre necessario specificare la posizione del file .zip che desideri utilizzare per aggiornare il codice della funzione. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza le opzioni `--s3-bucket` e `--s3-key` come illustrato nel seguente comando di esempio. È necessario utilizzare il parametro `--s3-object-version` solo per gli oggetti con controllo delle versioni.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Creazione e aggiornamento delle funzioni con file .zip utilizzando l'API Lambda

Per creare e aggiornare le funzioni mediante un archivio di file .zip, utilizza le seguenti operazioni API:

- CreateFunction
- UpdateFunctionCode

Creazione e aggiornamento di funzioni con file.zip utilizzando AWS SAM

Il AWS Serverless Application Model (AWS SAM) è un toolkit che aiuta a semplificare il processo di creazione ed esecuzione di applicazioni serverless su. AWS Definisci le risorse per la tua

applicazione in un modello YAML o JSON e utilizzi l'interfaccia a riga di AWS SAM comando (AWS SAM CLI) per creare, impacchettare e distribuire le tue applicazioni. Quando crei una funzione Lambda da un AWS SAM modello, crea AWS SAM automaticamente un pacchetto di distribuzione.zip o un'immagine del contenitore con il codice della funzione e le eventuali dipendenze specificate. Per ulteriori informazioni sull'utilizzo AWS SAM per creare e distribuire funzioni Lambda, consulta la Guida introduttiva AWS Serverless Application Model alla AWS SAM Developer Guide.

È inoltre possibile utilizzare AWS SAM per creare una funzione Lambda utilizzando un archivio di file.zip esistente. Per creare una funzione Lambda utilizzando AWS SAM, puoi salvare il tuo file.zip in un bucket Amazon S3 o in una cartella locale sulla tua macchina di compilazione. Per istruzioni su come caricare un file su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

Nel AWS SAM modello, la `AWS::Serverless::Function` risorsa specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `CodeUri`- impostato sull'URI Amazon S3 del codice della funzione, sul percorso della cartella locale o sull'oggetto FunctionCode
- `Runtime`: imposta il runtime prescelto

Inoltre AWS SAM, se il tuo file.zip è più grande di 50 MB, non è necessario caricarlo prima in un bucket Amazon S3. AWS SAM puoi caricare pacchetti.zip fino alla dimensione massima consentita di 250 MB (decompressi) da una posizione sulla macchina di compilazione locale.

Per ulteriori informazioni sulla distribuzione delle funzioni utilizzando il file.zip in AWS SAM, consulta la Guida per gli sviluppatori. AWS::Serverless::FunctionAWS SAM

Creazione e aggiornamento di funzioni con file.zip utilizzando CloudFormation

È possibile utilizzare CloudFormation per creare una funzione Lambda utilizzando un archivio di file.zip. Per creare una funzione Lambda da un file .zip, devi prima caricare il file su un bucket Amazon S3. Per istruzioni su come caricare un file su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

Per i runtime di Node.js e Python, puoi anche fornire codice sorgente in linea nel tuo modello. CloudFormation CloudFormation quindi crea un file.zip contenente il codice quando crei la funzione.

Utilizzo di un file .zip esistente

Nel CloudFormation modello, la `AWS::Lambda::Function` risorsa specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `Code`: inserisci il nome del bucket Amazon S3 e il nome del file .zip nei campi `S3Bucket` e `S3Key`
- `Runtime`: imposta il runtime prescelto

Creazione di un file .zip da codice inline

È possibile dichiarare semplici funzioni scritte in Python o Node.js in linea in un modello. CloudFormation Poiché il codice è incorporato in YAML o JSON, non puoi aggiungere dipendenze esterne al tuo pacchetto di implementazione. Ciò significa che la funzione deve utilizzare la versione dell' AWS SDK inclusa nel runtime. I requisiti del modello, come la necessità di evitare determinati caratteri, rendono anche più difficile l'utilizzo delle funzionalità di controllo della sintassi e di completamento del codice dell'IDE. Ciò significa che il tuo modello potrebbe richiedere test aggiuntivi. A causa di queste limitazioni, la dichiarazione di funzioni in linea è più adatta per codice molto semplice che cambia raramente.

Per creare un file .zip dal codice inline per i runtime Node.js e Python, imposta le seguenti proprietà nella risorsa del modello `AWS::Lambda::Function`:

- `PackageType`: imposta il valore su `Zip`
- `Code`: inserisci il codice della funzione nel campo `ZipFile`
- `Runtime`: imposta il runtime prescelto

Il file.zip che CloudFormation genera non può superare i 4 MB. Per ulteriori informazioni sulla distribuzione delle funzioni utilizzando il file.zip in CloudFormation, AWS::Lambda::Functionconsultate la Guida per l'utente.CloudFormation

Distribuisci funzioni Lambda per Python con immagini di container

Esistono tre modi per creare un'immagine di container per una funzione Lambda in Python:

- Usare un'immagine AWS base per Python

  Le immagini di base AWS sono precaricate con un runtime in linguaggio, un client di interfaccia di runtime per gestire l'interazione tra Lambda e il codice della funzione e un emulatore di interfaccia di runtime per i test locali.
- Utilizzo di un'immagine di AWS base solo per il sistema operativo

  AWS Le immagini di base solo per il sistema operativo contengono una distribuzione Amazon Linux e l'emulatore di interfaccia di runtime. Queste immagini vengono comunemente utilizzate per creare immagini di container per linguaggi compilati, come Go e Rust, e per un linguaggio o una versione di linguaggio per cui Lambda non fornisce un'immagine di base, come Node.js 19. Puoi anche utilizzare immagini di base solo per il sistema operativo per implementare un runtime personalizzato. Per rendere l'immagine compatibile con Lambda, devi includere il client di interfaccia di runtime per Python nell'immagine.
- Utilizzo di un'immagine non di base AWS

  È possibile utilizzare un'immagine di base alternativa da un altro registro del container, come ad esempio Alpine Linux o Debian. Puoi anche utilizzare un'immagine personalizzata creata dalla tua organizzazione. Per rendere l'immagine compatibile con Lambda, devi includere il client di interfaccia di runtime per Python nell'immagine.

> **Tip**
>
> Per ridurre il tempo necessario all'attivazione delle funzioni del container Lambda, consulta Utilizzo di compilazioni a più fasi nella documentazione Docker. Per creare immagini di container efficienti, segui le best practice per scrivere file Docker.

Questa pagina spiega come creare, testare e implementare le immagini di container per Lambda.

Argomenti

- AWS immagini base per Python
- Usare un'immagine AWS base per Python

- Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

AWS immagini base per Python

AWS fornisce le seguenti immagini di base per Python:

Tag	Runtime	Sistema operativo	Dockerfile	Raggiunta obsolescenza
3.14	Python 3.14	Amazon Linux 2023	Dockerfile per Python 3.14 e versioni successive GitHub	30 giugno 2029
3.13	Python 3.13	Amazon Linux 2023	Dockerfile per Python 3.13 e versioni successive GitHub	30 giugno 2029
3,12	Python 3.12	Amazon Linux 2023	Dockerfile per Python 3.12 e versioni successive GitHub	31 ottobre 2028
3.11	Python 3.11	Amazon Linux 2	Dockerfile per Python 3.11 e versioni successive GitHub	30 giugno 2027
3,10	Python 3.10	Amazon Linux 2	Dockerfile per Python 3.10 e versioni successive GitHub	31 ottobre 2026

Archivio Amazon ECR: gallery.ecr. aws/lambda/python

Le immagini di base Python 3.12 e versioni successive si basano sull'immagine di container minima di Amazon Linux 2023. Le immagini di base di Python 3.8-3.11 sono basate sull'immagine Amazon Linux 2. AL2Le immagini basate su 023 offrono diversi vantaggi rispetto ad Amazon Linux 2, tra cui un ingombro di distribuzione ridotto e versioni aggiornate di librerie come. `glibc`

AL2Le immagini basate su 023 utilizzano `microdnf` (symlinked as`dnf`) come gestore di pacchetti anzichéyum, che è il gestore di pacchetti predefinito in Amazon Linux 2. `microdnf`è

un'implementazione autonoma di. `dnf` Per un elenco dei pacchetti inclusi nelle immagini AL2 basate su 023, consulta le colonne Minimal Container in Confronto dei pacchetti installati su Amazon Linux 2023 Container Images. Per ulteriori informazioni sulle differenze tra AL2 023 e Amazon Linux 2, consulta la sezione Introduzione al runtime di Amazon Linux 2023 AWS Lambda sul AWS Compute Blog.

> **Note**
>
> Per eseguire immagini AL2 basate su 023 localmente, incluso with AWS Serverless Application Model (AWS SAM), devi usare Docker versione 20.10.10 o successiva.

Percorso di ricerca delle dipendenze nelle immagini di base

Quando utilizzi un'istruzione `import` nel codice, il runtime Python cerca nelle directory del suo percorso di ricerca finché non trova il modulo o il pacchetto. Per impostazione predefinita, il runtime cerca prima nella directory `{LAMBDA_TASK_ROOT}`. Se includi una versione di una libreria inclusa nel runtime nella tua immagine, questa versione avrà la precedenza sulla versione inclusa nel runtime.

Gli altri passaggi del percorso di ricerca dipendono dalla versione dell'immagine base Lambda per il Python che stai utilizzando:

- Python 3.11 e versioni successive: le librerie incluse nel runtime e le librerie installate con pip sono installate nella directory `/var/lang/lib/python3.11/site-packages`. Questa directory ha la precedenza su `/var/runtime` nel percorso di ricerca. Puoi sovrascrivere l'SDK usando pip per installare una versione più recente. Puoi usare pip per verificare che l'SDK incluso nel runtime e le sue dipendenze siano compatibili con tutti i pacchetti che installi.
- Python 3.8-3.10: le librerie incluse nel runtime sono installate nella directory `/var/runtime`. Le librerie installate da pip sono installate nella directory `/var/lang/lib/python3.x/site-packages`. La directory `/var/runtime` ha la precedenza su `/var/lang/lib/python3.x/site-packages` nel percorso di ricerca.

Puoi visualizzare il percorso di ricerca completo per la tua funzione Lambda aggiungendo il seguente frammento di codice.

```
import sys

search_path = sys.path
```

```
print(search_path)
```

Usare un'immagine AWS base per Python

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS CLI versione 2
- Docker (versione minima 25.0.0)
- Il plugin buildx di Docker.
- Python

Creazione di un'immagine da un'immagine di base

Per creare un'immagine contenitore da un'immagine di AWS base per Python

1. Crea una directory per il progetto, quindi passa a quella directory.

   ```
   mkdir example
   cd example
   ```

2. Crea un nuovo file denominato `lambda_function.py`. A fini di test, puoi utilizzare il codice della funzione di esempio seguente o sostituirlo con il tuo codice personalizzato.

   Example Funzione Python

   ```
   import sys
   def handler(event, context):
       return 'Hello from AWS Lambda using Python' + sys.version + '!'
   ```

3. Crea un nuovo file denominato `requirements.txt`. Se stai utilizzando il codice della funzione di esempio del passaggio precedente, puoi lasciare il file vuoto perché non ci sono dipendenze. Altrimenti, elenca ogni libreria richiesta. Ad esempio, ecco come dovrebbe apparire la tua versione di `requirements.txt` se la funzione utilizza AWS SDK per Python (Boto3):

   Example requirements.txt

   ```
   boto3
   ```

4. Crea un nuovo Dockerfile con la seguente configurazione:

   - Imposta la proprietà `FROM` sull'URI dell'immagine di base.
   - Utilizza il comando COPY per copiare il codice della funzione e le dipendenze di runtime in `{LAMBDA_TASK_ROOT}`, una variabile d'ambiente definita da Lambda.
   - Imposta l'argomento `CMD` specificando il gestore della funzione Lambda.

   Nota che l'esempio Dockerfile non include un'istruzione USER. Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

   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. Crea l'immagine Docker con il comando docker build. L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il tag `test`. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'`--provenance=false`opzione.

   ```
   docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
    .
   ```

> **Note**
>
> Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Se intendi creare una funzione Lambda utilizzando l'architettura del set di ARM64 istruzioni, assicurati di modificare il comando per utilizzare invece l'`--platform linux/arm64`opzione.

(Facoltativo) Test dell'immagine in locale

1. Avvia l'immagine Docker con il comando docker run. In questo esempio, `docker-image` è il nome dell'immagine e `test` è il tag.

   ```
   docker run --platform linux/amd64 -p 9000:8080 docker-image:test
   ```

   Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Se hai creato l'immagine Docker per l'architettura del set di ARM64 istruzioni, assicurati di utilizzare l'`--platform linux/arm64`opzione invece di. `--platform linux/amd64`

2. Da una nuova finestra di terminale, invia un evento all'endpoint locale.

   Linux/macOS

   Su MacOS o Linux, esegui il comando seguente `curl`:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

In PowerShell, esegui il seguente `Invoke-WebRequest` comando:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Ottieni l'ID del container.

```
docker ps
```

4. Utilizza il comando docker kill per arrestare il container. In questo comando, sostituisci `3766c4ab331c` con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

Implementazione dell'immagine

Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Esegui il get-login-passwordcomando per autenticare la CLI Docker nel tuo registro Amazon ECR.
   - Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
   - Sostituiscilo `111122223333` con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
    "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. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando docker tag per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:

- `docker-image:test` è il nome e tag dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
- Sostituisci l'`<ECRrepositoryUri>` con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando docker push per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crea un ruolo di esecuzione per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'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**
>
> È possibile creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta Autorizzazioni multiaccount Amazon ECR.

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nell'archivio Amazon ECR e quindi utilizzare il update-function-codecomando per distribuire l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per distribuire la nuova immagine nella stessa funzione Lambda, è necessario utilizzare update-function-codeil comando, anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

Se utilizzi un'immagine di base solo per il sistema operativo o un'immagine di base alternativa, devi includere il client dell'interfaccia di runtime nell'immagine. Il client dell'interfaccia di runtime estende l'API Runtime, che gestisce l'interazione tra Lambda e il codice della funzione.

Installa il client di interfaccia di runtime per Python utilizzando il gestore di pacchetti pip:

```
pip install awslambdaric
```

Puoi anche scaricare il client dell'interfaccia di runtime Python da. GitHub

L'esempio seguente dimostra come creare un'immagine contenitore per Python usando un'immagine non di base AWS . Il Dockerfile di esempio utilizza un'immagine di base Python ufficiale. Il Dockerfile include il client di interfaccia di runtime per Python.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS CLI versione 2
- Docker (versione minima 25.0.0)
- Il plugin buildx di Docker.
- Python

Creazione di un'immagine da un'immagine di base alternativa

Per creare un'immagine contenitore da un'immagine non di AWS base

1. Crea una directory per il progetto, quindi passa a quella directory.

   ```
   mkdir example
   cd example
   ```

2. Crea un nuovo file denominato `lambda_function.py`. A fini di test, puoi utilizzare il codice della funzione di esempio seguente o sostituirlo con il tuo codice personalizzato.

   Example Funzione Python

   ```
   import sys
   def handler(event, context):
       return 'Hello from AWS Lambda using Python' + sys.version + '!'
   ```

3. Crea un nuovo file denominato `requirements.txt`. Se stai utilizzando il codice della funzione di esempio del passaggio precedente, puoi lasciare il file vuoto perché non ci sono dipendenze. Altrimenti, elenca ogni libreria richiesta. Ad esempio, ecco come dovrebbe apparire la tua versione di `requirements.txt` se la funzione utilizza AWS SDK per Python (Boto3):

Example requirements.txt

```
boto3
```

4. Crea un nuovo Dockerfile. Il seguente Dockerfile utilizza un'immagine di base Python ufficiale anziché un'immagine di base AWS. Il Dockerfile include il client di interfaccia di runtime, che rende l'immagine compatibile con Lambda. Il seguente Dockerfile di esempio utilizza una build multi-fase.

   - Imposta la proprietà `FROM` sull'immagine di base.
   - Imposta l'`ENTRYPOINT` sul modulo su cui desideri che il container Docker venga eseguito all'avvio. In questo caso, il modulo è il client di interfaccia di runtime.
   - Imposta il `CMD` specificando il gestore della funzione Lambda.

   Nota che l'esempio Dockerfile non include un'istruzione USER. Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

   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. Crea l'immagine Docker con il comando docker build. L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il tag `test`. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'`--provenance=false`opzione.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Se intendi creare una funzione Lambda utilizzando l'architettura del set di ARM64 istruzioni, assicurati di modificare il comando per utilizzare invece l'`--platform linux/arm64`opzione.

(Facoltativo) Test dell'immagine in locale

Usa il simulatore dell'interfaccia di runtime per testare localmente l'immagine. Puoi creare l'emulatore nella tua immagine o seguire la procedura riportata e installarlo sul tuo computer locale.

Installazione ed esecuzione dell'emulatore di interfaccia di runtime sul computer locale

1. Dalla directory del progetto, esegui il comando seguente per scaricare l'emulatore di interfaccia di runtime (architettura x86-64) GitHub e installarlo sul computer locale.

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
```

Per installare l'emulatore arm64, sostituisci l'URL del GitHub repository nel comando precedente con il seguente:

```
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
```

Per installare l'emulatore arm64, sostituisci `$downloadLink` con quanto segue:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Avvia l'immagine Docker con il comando docker run. Tenere presente quanto segue:

   - `docker-image` è il nome dell'immagine e `test` è il tag.
   - `/usr/local/bin/python -m awslambdaric lambda_function.handler` è l'`ENTRYPOINT` seguito dal `CMD` del Dockerfile.

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
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Se hai creato l'immagine Docker per l'architettura del set di ARM64 istruzioni, assicurati di utilizzare l'`--platform linux/arm64`opzione invece di. `--platform linux/amd64`

3. Pubblica un evento nell'endpoint locale.

Linux/macOS

Su MacOS o Linux, esegui il comando seguente `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

In PowerShell, esegui il seguente `Invoke-WebRequest` comando:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Ottieni l'ID del container.

```
docker ps
```

5. Utilizza il comando docker kill per arrestare il container. In questo comando, sostituisci `3766c4ab331c` con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

Implementazione dell'immagine

Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Esegui il get-login-passwordcomando per autenticare la CLI Docker nel tuo registro Amazon ECR.
   - Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
   - Sostituiscilo `111122223333` con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
    "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. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando docker tag per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:

- `docker-image:test` è il nome e tag dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
- Sostituisci l'`<ECRrepositoryUri>` con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando docker push per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crea un ruolo di esecuzione per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'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**
>
> È possibile creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta Autorizzazioni multiaccount Amazon ECR.

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nell'archivio Amazon ECR e quindi utilizzare il update-function-codecomando per distribuire l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per distribuire la nuova immagine nella stessa funzione Lambda, è necessario utilizzare update-function-codeil comando, anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Per un esempio di come creare un'immagine Python da un'immagine di base Alpine, consulta Supporto delle immagini di container per Lambda sul Blog AWS .

Utilizzo dei livelli per le funzioni Lambda in Python

Usa i livelli Lambda per impacchettare codice e dipendenze che desideri riutilizzare in più funzioni. I livelli di solito contengono dipendenze dalla libreria, un runtime personalizzato o file di configurazione. La creazione di un livello prevede tre passaggi generali:

1. Crea un pacchetto per il contenuto del livello. Ciò significa creare un archivio di file con estensione .zip che contiene le dipendenze che desideri utilizzare nelle funzioni.
2. Crea il livello in Lambda.
3. Aggiungi il livello alle tue funzioni.

Argomenti

- Crea un pacchetto per il contenuto del livello
- Crea il layer in Lambda
- Aggiungi il layer alla tua funzione
- App di esempio

Crea un pacchetto per il contenuto del livello

Per creare un layer, raggruppa i pacchetti in un archivio di file.zip che soddisfi i seguenti requisiti:

- Costruite il layer utilizzando la stessa versione di Python che intendete utilizzare per la funzione Lambda. Ad esempio, se crei il tuo layer usando Python 3.14, usa il runtime Python 3.14 per la tua funzione.
- Il file.zip deve includere una `python` directory a livello root.
- I pacchetti del livello devono essere compatibili con Linux. Le funzioni Lambda vengono eseguite su Amazon Linux.

È possibile creare livelli che contengono librerie Python di terze parti installate con `pip` (come `requests` `opandas`) o moduli e pacchetti Python personalizzati.

Dipendenze da terze parti

Per creare un livello utilizzando pacchetti pip

1. Scegliete uno dei seguenti metodi per installare `pip` i pacchetti nella directory di primo livello richiesta ()`python/`:

   pip install

   Per pacchetti Python puri (come requests o boto3):

   ```
   pip install requests -t python/
   ```

   Alcuni pacchetti Python, come NumPy e Pandas, includono componenti C compilati. Se state creando un layer con questi pacchetti su macOS o Windows, potreste aver bisogno di usare questo comando per installare una ruota compatibile con Linux:

   ```
   pip install numpy --platform manylinux2014_x86_64 --only-binary=:all: -t python/
   ```

   Per ulteriori informazioni sull'utilizzo dei pacchetti Python che contengono componenti compilati, vedere. Creazione di un pacchetto di implementazione .zip con librerie native

   requirements.txt

   L'utilizzo di un `requirements.txt` file consente di gestire le versioni dei pacchetti e garantire installazioni coerenti.

   Example requirements.txt

   ```
   requests==2.31.0
   boto3==1.37.34
   numpy==1.26.4
   ```

   Se il tuo `requirements.txt` file include solo pacchetti Python puri (come requests o boto3):

   ```
   pip install -r requirements.txt -t python/
   ```

Alcuni pacchetti Python, come NumPy e Pandas, includono componenti C compilati. Se state creando un layer con questi pacchetti su macOS o Windows, potreste aver bisogno di usare questo comando per installare una ruota compatibile con Linux:

```
pip install -r requirements.txt --platform manylinux2014_x86_64 --only-
binary=:all: -t python/
```

Per ulteriori informazioni sull'utilizzo dei pacchetti Python che contengono componenti compilati, vedere. Creazione di un pacchetto di implementazione .zip con librerie native

2. Comprimi il contenuto della `python` directory.

Linux/macOS

```
zip -r layer.zip python/
```

PowerShell

```
Compress-Archive -Path .\python -DestinationPath .\layer.zip
```

La struttura delle cartelle del file.zip dovrebbe essere la seguente:

```
python/              # Required top-level directory
### requests/
### boto3/
### numpy/
### (dependencies of the other packages)
```

> **Note**
>
> Se usi un ambiente virtuale Python (venv) per installare i pacchetti, la struttura delle cartelle sarà diversa (ad esempio,). `python/lib/python3.x/site-packages` Finché il file.zip include la `python` directory a livello principale, Lambda può localizzare e importare i pacchetti.

Moduli Python personalizzati

Per creare un livello utilizzando il proprio codice

1. Crea la directory di primo livello richiesta per il tuo layer:

```
mkdir python
```

2. Crea i tuoi moduli Python nella `python` directory. Il seguente modulo di esempio convalida gli ordini confermando che contengono le informazioni richieste.

   Example modulo personalizzato: 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. Comprimi il contenuto della `python` directory.

   Linux/macOS

   ```
   zip -r layer.zip python/
   ```

   PowerShell

   ```
   Compress-Archive -Path .\python -DestinationPath .\layer.zip
   ```

   La struttura delle cartelle del file.zip dovrebbe essere la seguente:

   ```
   python/     # Required top-level directory
   ### validator.py
   ```

4. Nella tua funzione, importa e usa i moduli come faresti con qualsiasi pacchetto Python. Esempio:

   ```
   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'
   ```

```
        })
```

È possibile utilizzare il seguente evento di test per richiamare la funzione:

```
{
    "body": "{\"product_id\": \"ABC123\", \"quantity\": 2, \"priority\": \"express
\"}"
}
```

Risposta prevista:

```
{
  "statusCode": 200,
  "body": "{\"message\": \"Order validated successfully\", \"order\": {\"product_id
\": \"ABC123\", \"quantity\": 2, \"shipping_priority\": \"express\"}}"
}
```

Crea il layer in Lambda

È possibile pubblicare il layer utilizzando la console AWS CLI o la console Lambda.

AWS CLI

Esegui il publish-layer-version AWS CLI comando per creare il livello Lambda:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip
 --compatible-runtimes python3.14
```

Il parametro di runtime compatibile è facoltativo. Quando specificato, Lambda utilizza questo parametro per filtrare i livelli nella console Lambda.

Console

Creazione di un livello (console)

1. Apri la pagina Layers (Livelli) nella console Lambda.
2. Scegli Create layer (Crea livello).
3. Scegli Carica un file.zip, quindi carica l'archivio.zip che hai creato in precedenza.

4. (Facoltativo) Per i runtime compatibili, scegliete il runtime Python che corrisponde alla versione di Python che avete usato per creare il layer.
5. Scegli Create (Crea).

Aggiungi il layer alla tua funzione

AWS CLI

Per collegare il layer alla tua funzione, esegui il update-function-configuration AWS CLI comando. Per il `--layers` parametro, utilizzate il livello ARN. L'ARN deve specificare la versione (ad esempio,`arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`). Per ulteriori informazioni, consulta Livelli e versioni di livelli.

```
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"
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Console

Aggiunta di un livello a una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la funzione.
3. Scorri verso il basso fino alla sezione Livelli, quindi scegli Aggiungi un livello.
4. In Scegli un livello, seleziona Livelli personalizzati, quindi scegli il tuo livello.

> **Note**
>
> Se non hai aggiunto un runtime compatibile quando hai creato il layer, il tuo layer non verrà elencato qui. È possibile specificare invece il livello ARN.

5. Scegliere Aggiungi.

App di esempio

Per altri esempi di utilizzo dei layer Lambda, consultate l'applicazione di esempio layer-python nel repository Developer Guide. AWS Lambda GitHub Questa applicazione include due livelli che contengono librerie Python. Dopo aver creato i livelli, è possibile distribuire e richiamare le funzioni corrispondenti per confermare che i livelli funzionino come previsto.

Utilizzo dell'oggetto del contesto Lambda per recuperare le informazioni sulla funzione Python

Quando Lambda esegue la funzione, passa un oggetto Context al gestore. Questo oggetto fornisce i metodi e le proprietà che forniscono le informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. Per ulteriori informazioni su come l'oggetto contesto viene passato al gestore di funzioni, consulta Definire l'handler della funzione Lambda in Python.

Metodi del contesto

- `get_remaining_time_in_millis`: restituisce il numero di millisecondi rimasti prima del timeout dell'esecuzione.

Proprietà del contesto

- `function_name`: il nome della funzione Lambda.
- `function_version`: la versione della funzione.
- `invoked_function_arn`: l'Amazon Resource Name (ARN) utilizzato per richiamare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `memory_limit_in_mb`: la quantità di memoria allocata per la funzione.
- `aws_request_id`: l'identificatore della richiesta di invocazione.
- `log_group_name`: il gruppo di log per la funzione.
- `log_stream_name`: il flusso di log per l'istanza della funzione.
- `identity`: (app per dispositivi mobili) Informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
  - `cognito_identity_id`: l'identità autenticata di Amazon Cognito.
  - `cognito_identity_pool_id`: il pool di identità Amazon Cognito che ha autorizzato l'invocazione.
- `client_context`: (app per dispositivi mobili) Contesto client fornito a Lambda dall'applicazione client.
  - `client.installation_id`
  - `client.app_title`
  - `client.app_version_name`
  - `client.app_version_code`

- `client.app_package_name`
- `custom`: un `dict` di valori personalizzati impostati dall'applicazione client per dispositivi mobili.
- `env`: un `dict` di informazioni di ambiente fornite dall'SDK AWS.

Powertools for Lambda (Python) fornisce una definizione di interfaccia per l'oggetto del contesto Lambda. È possibile utilizzare la definizione dell'interfaccia per i suggerimenti sul tipo o per esaminare ulteriormente la struttura dell'oggetto del contesto Lambda. Per la definizione dell'interfaccia, consulta lambda_context.py nel repository powertools-lambda-python su GitHub.

L'esempio seguente mostra una funzione di gestione che registra le informazioni di contesto.

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())
```

Oltre alle opzioni elencate in precedenza, è possibile utilizzare l'SDK AWS X-Ray per Strumentazione del codice Python in AWS Lambda per identificare i percorsi del codice critici, tenere traccia delle relative prestazioni e acquisire i dati per l'analisi.

Registrare e monitorare le funzioni Lambda con Python

AWS Lambda monitora automaticamente le funzioni Lambda e invia le voci di registro ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della funzione. L'ambiente di runtime di Lambda invia al flusso di log i dettagli su ogni invocazione e altri output dal codice della funzione. Per ulteriori informazioni sui CloudWatch registri, consulta. Lambda invia automaticamente i log delle funzioni a CloudWatch Logs.

Per l'output dei log dal codice della funzione, puoi utilizzare il modulo di `logging` integrato. Per ottenere voci più dettagliate, puoi utilizzare qualunque libreria di log che scrive in `stdout` o `stderr`.

Stampa nel log

Per inviare l'output base ai log, puoi utilizzare un metodo `print` nella funzione. L'esempio seguente registra i valori del gruppo e del flusso di log CloudWatch Logs e dell'oggetto evento.

Nota che se la tua funzione genera log usando istruzioni Python`print`, Lambda può inviare output di log a Logs solo in formato testo semplice. CloudWatch Per acquisire i log in formato JSON strutturato, è necessario utilizzare una libreria di registrazione supportata. Per ulteriori informazioni, consulta the section called “Utilizzo dei controlli di registrazione avanzati di Lambda con 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
```

Il runtime di Python registra `START`, `END` e `REPORT` per ogni chiamata. La riga `REPORT` include i seguenti dati:

Campi dati della riga REPORT

- RequestId— L'ID univoco della richiesta per la chiamata.
- Durata – La quantità di tempo che il metodo del gestore della funzione impiega durante l'elaborazione dell'evento.
- Durata fatturata – La quantità di tempo fatturata per la chiamata.
- Dimensioni memoria – La quantità di memoria allocata per la funzione.
- Quantità max utilizzata – La quantità di memoria utilizzata dalla funzione. Quando le invocazioni condividono un ambiente di esecuzione, Lambda riporta la memoria massima utilizzata in tutte le invocazioni. Questo comportamento potrebbe comportare un valore riportato superiore al previsto.
- Durata Init – Per la prima richiesta servita, la quantità di tempo impiegato dal runtime per caricare la funzione ed eseguire il codice al di fuori del metodo del gestore.
- XRAY TraceId — Per le richieste tracciate, l'ID di traccia.AWS X-Ray
- SegmentId— Per le richieste tracciate, l'ID del segmento X-Ray.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

Utilizzo di una libreria di log

Per registri più dettagliati, utilizza il modulo di log nella libreria standard o qualunque libreria di log di terzi che scrive in `stdout` o `stderr`.

Per i runtime Python supportati, puoi scegliere se i log creati utilizzando il modulo `logging` standard vengono acquisiti in testo normale o JSON. Per ulteriori informazioni, consulta the section called “Utilizzo dei controlli di registrazione avanzati di Lambda con Python”.

Attualmente, il formato di log predefinito per tutti i runtime di Python è il testo normale. L'esempio seguente mostra come gli output di registro creati utilizzando il `logging` modulo standard vengono acquisiti in testo semplice in Logs. CloudWatch

```
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)
```

L'output di `logger` include il livello del log, il timestamp e l'ID della richiesta.

```
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**
>
> Quando il formato di log della funzione è impostato su testo semplice, l'impostazione predefinita a livello di log per i runtime Python è WARN. Ciò significa che Lambda invia solo output di log di livello WARN e inferiore a Logs. CloudWatch Per modificare il livello di log predefinito, utilizza il metodo `logging` `setLevel()` di Python come mostrato in questo codice di esempio. Se imposti il formato di log della funzione su JSON, consigliamo di configurare il livello di log della funzione utilizzando i controlli di registrazione avanzati di

Lambda e non impostando il livello di log nel codice. Per ulteriori informazioni, consulta the section called “Utilizzo del filtraggio a livello di log con Python”

Utilizzo dei controlli di registrazione avanzati di Lambda con Python

Per avere un maggiore controllo sul modo in cui i log delle tue funzioni vengono acquisiti, elaborati e utilizzati, puoi configurare le seguenti opzioni di registrazione per i runtime Python di Lambda supportati:

- Formato di log: scegli tra i formati di testo normale e JSON strutturato per i log della funzione
- Livello di log: per i log in formato JSON, scegli il livello di dettaglio dei log che Lambda invia ad CloudWatch Amazon, come ERROR, DEBUG o INFO
- Gruppo di log: scegli il gruppo di log a cui la CloudWatch funzione invia i log

Per ulteriori informazioni su queste opzioni di registrazione e istruzioni su come configurare la funzione per utilizzarle, consulta la pagina the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”.

Per ulteriori informazioni sull'utilizzo delle opzioni di formato e livello di log con le funzioni Lambda in Python, consulta le istruzioni nelle sezioni seguenti.

Utilizzo di log JSON strutturati con Python

Se selezioni JSON per il formato di log della tua funzione, Lambda invierà i log in uscita dalla libreria di registrazione standard Python in un formato JSON strutturato. CloudWatch Ogni oggetto di log JSON contiene almeno quattro coppie chiave-valore con le seguenti chiavi:

- `"timestamp"`: l'ora in cui è stato generato il messaggio di log
- `"level"`: il livello di log assegnato al messaggio
- `"message"`: il contenuto del messaggio di log
- `"requestId"`: l'ID di richiesta univoco dell'invocazione alla funzione

La libreria `logging` di Python può anche aggiungere a tale oggetto JSON ulteriori coppie chiave-valore, come `"logger"`.

Gli esempi nelle sezioni seguenti mostrano come gli output di log generati utilizzando la libreria `logging` Python vengono acquisiti CloudWatch in Logs quando si configura il formato di log della funzione come JSON.

Tieni presente che se usi il metodo `print` per generare output log di base come descritto in the section called “Stampa nel log”, Lambda acquisirà questi output come testo normale anche se configuri il formato di registrazione della funzione come JSON.

Output log JSON standard che utilizzano la libreria di registrazione di Python

Il seguente frammento di codice e l'output di registro mostrano come gli output di log standard generati utilizzando la `logging` libreria Python vengono acquisiti in CloudWatch Logs quando il formato di log della funzione è impostato su JSON.

Example Codice di registrazione di Python

```
import logging
logger = logging.getLogger()

def lambda_handler(event, context):
    logger.info("Inside the handler function")
```

Example Record di log JSON

```
{
    "timestamp":"2025-10-27T19:17:45.586Z",
    "level":"INFO",
    "message":"Inside the handler function",
    "logger": "root",
    "requestId":"79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

Registrazione di parametri aggiuntivi in JSON

Quando il formato dei log della funzione è impostato su JSON, è possibile registrare ulteriori parametri con la libreria standard `logging` di Python utilizzando la parola chiave `extra` per passare un dizionario Python all'output dei log.

Example Codice di registrazione di Python

```
import logging
```

```
def lambda_handler(event, context):
    logging.info(
        "extra parameters example",
        extra={"a":"b", "b": [3]},
    )
```

Example Record di 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
  ]
}
```

Registrazione delle eccezioni in JSON

Il seguente frammento di codice mostra come le eccezioni Python vengono acquisite nell'output log della funzione quando si configura JSON come formato di log. Nota che agli output log generati utilizzando `logging.exception` viene assegnato il livello di log ERROR.

Example Codice di registrazione di Python

```
import logging

def lambda_handler(event, context):
    try:
        raise Exception("exception")
    except:
        logging.exception("msg")
```

Example Record di 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 JSON strutturati con altri strumenti di registrazione

Se il codice utilizza già un'altra libreria di registrazione, come Powertools for, per produrre log strutturati JSON AWS Lambda, non è necessario apportare alcuna modifica. AWS Lambda non codifica due volte i log che sono già codificati in JSON. Anche se configuri la tua funzione per utilizzare il formato di registro JSON, i tuoi output di registrazione vengono visualizzati nella struttura JSON che definisci. CloudWatch

L'esempio seguente mostra come gli output di log generati utilizzando il pacchetto Powertools for AWS Lambda vengono acquisiti in Logs. CloudWatch Il formato di questo output log è lo stesso indipendentemente dal fatto che la configurazione di registrazione della funzione sia impostata su JSON o TEXT. Per ulteriori informazioni sull'utilizzo di Powertools per, vedere e AWS Lambdathe section called “Utilizzo di Powertools per AWS Lambda (Python) AWS SAM e per la registrazione strutturata” the section called “Utilizzo di Powertools per AWS Lambda (Python) AWS CDK e per la registrazione strutturata”

Example Frammento di codice di registrazione Python (usando Powertools for) AWS Lambda

```
from aws_lambda_powertools import Logger

logger = Logger()

def lambda_handler(event, context):
    logger.info("Inside the handler function")
```

Example Record di registro JSON (utilizzando Powertools per) 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"
}
```

Utilizzo del filtraggio a livello di log con Python

Configurando il filtraggio a livello di registro, puoi scegliere di inviare solo i log di un determinato livello di registrazione o inferiore a Logs. CloudWatch Per informazioni su come configurare il filtraggio a livello di log della funzione, consulta la pagina the section called “Filtraggio a livello di log”.

Per AWS Lambda filtrare i log delle applicazioni in base al relativo livello di registro, la funzione deve utilizzare log in formato JSON. Puoi farlo in due modi:

- Crea output log utilizzando la libreria standard `logging` di Python e configura la funzione affinché utilizzi JSON come formato di log. Successivamente, AWS Lambda filtra gli output log utilizzando la coppia chiave-valore "livello" nell'oggetto JSON descritto in the section called “Utilizzo di log JSON strutturati con Python”. Per informazioni su come configurare il formato di log della funzione, consulta la pagina the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”.
- Utilizza un'altra libreria o metodo di registrazione per creare nel codice dei log JSON strutturati che includono una coppia chiave-valore "livello" che definisce il livello dell'output log. Ad esempio, puoi utilizzare Powertools per AWS Lambda generare output di log strutturati JSON dal tuo codice.

  Inoltre, è possibile utilizzare un'istruzione di stampa per generare un oggetto JSON contenente un identificatore a livello di log. La seguente istruzione print produce un output in formato JSON in cui il livello di registro è impostato su INFO. AWS Lambda invierà l'oggetto JSON a CloudWatch Logs se il livello di registrazione della funzione è impostato su INFO, DEBUG o TRACE.

  ```
  print('{"msg":"My log message", "level":"info"}')
  ```

Per consentire a Lambda di filtrare i log della funzione, è necessario includere anche una coppia chiave-valore `"timestamp"` nell'output log JSON. L'ora deve essere specificata in un formato di timestamp RFC 3339 valido. Se non fornisci un timestamp valido, Lambda assegnerà al log il livello INFO e aggiungerà un timestamp per tuo conto.

Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor del codice incorporato, troverai i log nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l'output del log nella sezione Dettagli.

Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (***your-function-name***/aws/lambda/).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Viene visualizzato un flusso di log quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire le chiamate simultanee. Per trovare i log per una chiamata specifica, consigliamo di strumentare la funzione con. AWS X-Ray X-Ray registra i dettagli sulla richiesta e il flusso di log nella traccia.

Visualizzazione dei log con AWS CLI

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della AWS CLI versione 2.

È possibile utilizzare AWS CLI per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'`base64` utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `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
```

L'cli-binary-formatopzione è obbligatoria se utilizzi la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
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
```

L'utilità `base64` è disponibile su Linux, macOS e Ubuntu su Windows. Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

Example Script get-logs.sh

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza `sed` per rimuovere le virgolette dal file di output e rimane in sospensione per 15

secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.

L'cli-binary-formatopzione è obbligatoria se utilizzi la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 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 (solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il seguente comando per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
    "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"
}
```


Eliminazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, eliminare il gruppo di log o configurare un periodo di conservazione trascorso il quale i log vengono eliminati automaticamente.

Utilizzo di altri strumenti di registrazione e librerie

Powertools for AWS Lambda (Python) è un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori. L'utilità di registrazione fornisce un logger ottimizzato per Lambda che include informazioni aggiuntive sul contesto delle funzioni in tutte le funzioni con output strutturato come JSON. Utilizza l'utility per eseguire le seguenti operazioni:

- Acquisizione di campi essenziali dal contesto Lambda, avvii a freddo e output di registrazione della struttura come JSON
- Registrazione degli eventi di chiamata Lambda quando richiesto (disabilitata per impostazione predefinita)
- Stampa di tutti i log solo per una percentuale di chiamate tramite campionamento dei log (disabilitata per impostazione predefinita)
- Aggiunta di chiavi supplementari al log strutturato in qualsiasi momento
- Utilizzo di un formattatore di log personalizzato (Bring Your Own Formatter) per generare i log in una struttura compatibile con Logging RFC dell'organizzazione

Utilizzo di Powertools per AWS Lambda (Python) AWS SAM e per la registrazione strutturata

Segui i passaggi riportati sotto per scaricare, creare e implementare un'applicazione Python Hello World di esempio con moduli Powertools per Python integrati utilizzando AWS SAM. Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio `hello world`.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Python 3.9
- AWS CLI versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Distribuisci un'applicazione di esempio AWS SAM

1. Inizializza l'applicazione utilizzando il modello Python Hello World.

```
sam init --app-template hello-world-powertools-python --name sam-app --package-type
 Zip --runtime python3.9 --no-tracing
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi `Enter`.

> **Note**
>
> Perché HelloWorldFunction potrebbe non avere un'autorizzazione definita, va bene? , assicurati di entrarey.

5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Richiama l'endpoint dell'API:

```
curl GET <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere i log per la funzione, esegui sam logs. Per ulteriori informazioni, consulta l'argomento relativo all'utilizzo dei log nella Guida per sviluppatori AWS Serverless Application Model .

```
sam logs --stack-name sam-app
```

L'output del log ha la struttura seguente:

```
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. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

Gestione della conservazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, elimina il gruppo di log o configura un periodo di conservazione dopo il quale i log CloudWatch vengono eliminati automaticamente. Per configurare la conservazione dei log, aggiungi quanto segue al tuo modello: AWS SAM

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

Utilizzo di Powertools per AWS Lambda (Python) AWS CDK e per la registrazione strutturata

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Hello World Python di esempio con i moduli Powertools for AWS Lambda (Python) integrati utilizzando il. AWS CDK Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio hello world.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Python 3.9
- AWS CLI versione 2
- AWS CDK versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Distribuisci un'applicazione di esempio AWS CDK

1. Crea una directory di progetto per la nuova applicazione.

```
mkdir hello-world
cd hello-world
```

2. Inizializza l'app.

```
cdk init app --language python
```

3. Installa le dipendenze di Python.

```
pip install -r requirements.txt
```

4. Crea una directory lambda_function nella cartella root.

```
mkdir lambda_function
cd lambda_function
```

5. Crea un file app.py e aggiungi il seguente codice al file. Questo è il codice per la funzione Lambda.

```
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. Apri la directory hello_world. Dovrebbe essere visualizzato un file denominato hello_world_stack.py.

```
cd ..
cd hello_world
```

7. Apri hello_world_stack.py e aggiungi il seguente codice al file. Contiene il Lambda Constructor, che crea la funzione Lambda, configura le variabili di ambiente per Powertools e imposta la conservazione dei log su una settimana, e il costruttore ApiGatewayv 1, che crea l'API REST.

```
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 via 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. Distribuisci l'applicazione.

```
cd ..
cdk deploy
```

9. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?OutputKey==`apiUrl`].OutputValue' --output text
```

10. Richiama l'endpoint dell'API:

```
curl GET <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

11. Per ottenere i log per la funzione, esegui sam logs. Per ulteriori informazioni, consulta l'argomento relativo all'utilizzo dei log nella Guida per sviluppatori AWS Serverless Application Model .

```
sam logs --stack-name HelloWorldStack
```

L'output del log ha la struttura seguente:

```
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. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
cdk destroy
```

Test delle funzioni AWS Lambda in Python

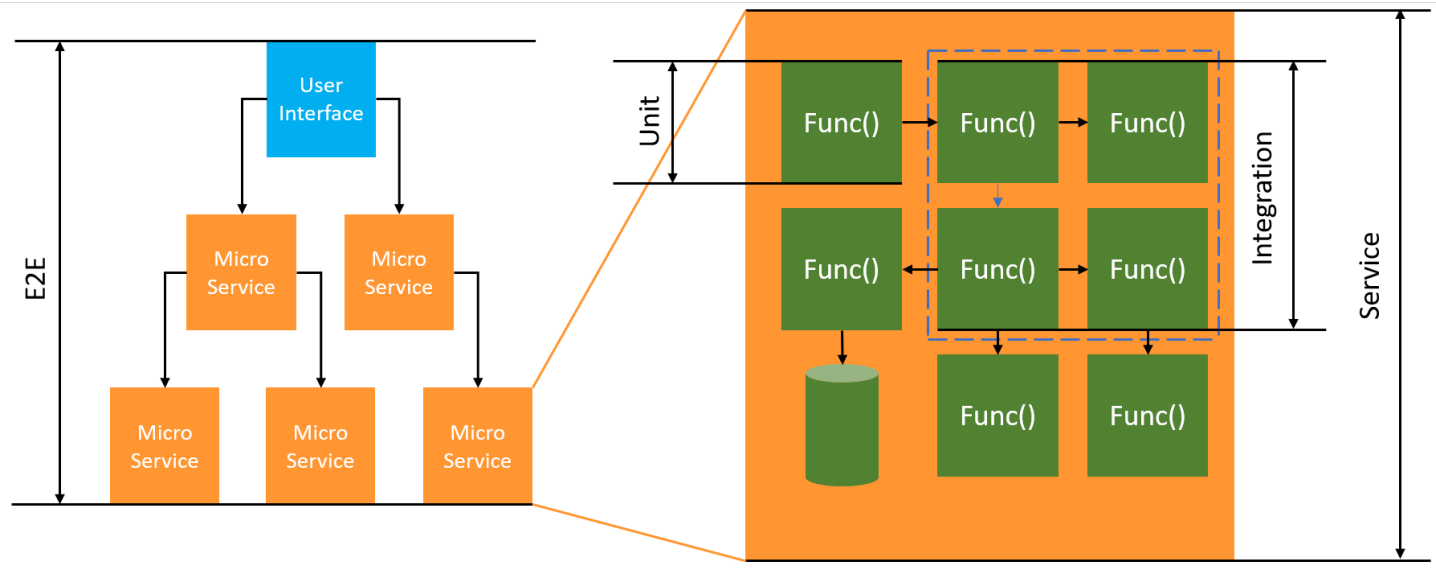
> **Note**
>
> Consulta il capitolo sul Test delle funzioni per un'introduzione completa alle tecniche e alle best practice per testare soluzioni serverless.

Sebbene il test delle funzioni serverless si basi su tipologie e tecniche di test consolidate, è importante tenere in considerazione la possibilità di effettuare test sulle applicazioni serverless nella loro interezza. I test basati sul cloud forniranno la misura più accurata della qualità sia delle funzioni sia delle applicazioni serverless.

Un'architettura applicativa serverless include servizi gestiti che forniscono funzionalità applicative critiche tramite chiamate API. Pertanto, è fondamentale che il ciclo di sviluppo preveda test automatici in grado di verificare il corretto funzionamento dell'interazione tra le funzioni e i servizi.

Se non crei test basati sul cloud, potresti riscontrare problemi dovuti alle differenze tra l'ambiente locale e quello implementato. Il processo di integrazione continua dovrebbe eseguire test su un insieme di risorse con provisioning nel cloud prima di promuovere il codice nell'ambiente di implementazione successivo, come quello di controllo qualità (QA), staging o produzione.

Continua a leggere questa breve guida per scoprire le strategie di test per le applicazioni serverless oppure visita il repository Serverless Test Samples per approfondire esempi pratici, specifici per il linguaggio e il runtime selezionati.



Per i test in ambito serverless, è necessario scrivere test unitari, test di integrazione e test end-to-end.

- Test unitari: vengono eseguiti su un blocco di codice isolato. Ad esempio, possono essere utilizzati per verificare la logica aziendale per calcolare le spese di spedizione in base a un articolo e a una destinazione specifici.
- Test di integrazione: coinvolgono due o più componenti o servizi che interagiscono, in genere in un ambiente cloud. Ad esempio, possono essere utilizzati per verificare che una funzione elabori gli eventi di una coda.
- Test end-to-end: verificano il comportamento di un'intera applicazione. Ad esempio, possono essere utilizzati per verificare che l'infrastruttura sia configurata correttamente e che gli eventi fluiscano tra i servizi come previsto per registrare l'ordine di un cliente.

Test delle applicazioni serverless

Generalmente, utilizzerai una combinazione di approcci per testare il codice dell'applicazione serverless, inclusi test nel cloud, test con mock e occasionalmente test con emulatori.

Test nel cloud

I test nel cloud sono utili per tutte le fasi di test, inclusi i test unitari, i test di integrazione e i test end-to-end. Esegui test su codice implementato nel cloud e interagisci con servizi basati su cloud. Questo approccio fornisce la misura più accurata della qualità del codice.

Un modo conveniente per eseguire il debug di una funzione Lambda nel cloud è con un evento di test nella console. Un evento di test è un input JSON per la funzione. Se la funzione non richiede input, l'evento può essere un documento JSON (`{}`) vuoto. La console fornisce eventi di esempio per una varietà di integrazioni di servizi. Dopo aver creato un evento nella console, puoi condividerlo con il tuo team per rendere i test più semplici e coerenti.

> **Note**
>
> Testare una funzione nella console è un modo rapido per iniziare, ma l'automazione dei cicli di test garantisce la qualità delle applicazioni e la velocità di sviluppo.

Strumenti di test

Esistono strumenti e tecniche per accelerare i cicli di feedback sullo sviluppo. Ad esempio, AWS SAM Accelerate e AWS CDK watch mode riducono entrambi il tempo necessario per aggiornare gli ambienti cloud.

Moto è una libreria Python per simulare servizi e risorse AWS, in modo da poter testare le funzioni con poche o nessuna modifica utilizzando decoratori per intercettare e simulare le risposte.

La funzione di convalida di Powertools per AWS Lambda (Python) fornisce decoratori che consentono di convalidare gli eventi di input e le risposte di output dalle funzioni Python.

Per ulteriori informazioni, leggi il post del blog Test unitari di Lambda con Python e servizi AWS fittizi.

Per ridurre la latenza associata alle iterazioni di implementazione del cloud, consulta le sezioni AWS Serverless Application Model (AWS SAM) Accelerate e AWS Cloud Development Kit (AWS CDK) watch mode. Questi strumenti monitorano l'infrastruttura e il codice per rilevare eventuali modifiche. Reagiscono a questi cambiamenti creando e implementando automaticamente aggiornamenti incrementali nell'ambiente cloud.

Alcuni esempi che utilizzano questi strumenti sono disponibili nel repository di codice Python Test Samples.

Strumentazione del codice Python in AWS Lambda

Lambda si integra con AWS X-Ray per aiutarti a tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzare X-Ray per tracciare una richiesta mentre attraversa le risorse nell'applicazione, che possono includere funzioni Lambda e altri servizi AWS.

Per inviare dati di tracciamento a X-Ray, è possibile utilizzare una delle tre librerie SDK:

- AWS Distro for OpenTelemetry (ADOT): una distribuzione supportata da AWS sicura, pronta per la produzione, dell'SDK OpenTelemetry (OTel).
- SDK AWS X-Ray per Python – un SDK per generare e inviare i dati di traccia su X-Ray.
- Powertools per AWS Lambda (Python): un toolkit rivolto agli sviluppatori per implementare le best practice serverless e aumentare la velocità di sviluppo.

Ciascun SDK offre dei modi per inviare i dati di telemetria al servizio X-Ray. Puoi quindi utilizzare X-Ray per visualizzare, filtrare e analizzare le metriche delle prestazioni dell'applicazione per identificare i problemi e le opportunità di ottimizzazione.

> **Important**
>
> Gli SDK X-Ray e Powertools per AWS Lambda fanno parte di una soluzione di strumentazione strettamente integrata offerta da AWS. I livelli Lambda ADOT fanno parte di uno standard di settore per la strumentazione di tracciamento che in generale raccoglie più dati, ma potrebbero non essere adatti a tutti i casi d'uso. È possibile implementare il tracciamento end-to-end di X-Ray utilizzando entrambe le soluzioni. Per ulteriori informazioni sulla scelta più adatta, consulta Scegliere tra le AWS Distro per OpenTelemetry e SDK X-Ray.

Sezioni

- Utilizzo di Powertools per AWS Lambda (Python) e di AWS SAM per il tracciamento
- Utilizzo di Powertools per AWS Lambda (Python) e del AWS CDK per il tracciamento
- Utilizzo di ADOT per strumentare le funzioni Python
- Utilizzo dell'SDK X-Ray per strumentare le funzioni Python
- Attivazione del tracciamento con la console Lambda

- Attivazione del tracciamento con l'API Lambda
- Attivazione del tracciamento con CloudFormation
- Interpretazione di una traccia X-Ray
- Memorizzazione delle dipendenze di runtime in un livello (SDK X-Ray)

Utilizzo di Powertools per AWS Lambda (Python) e di AWS SAM per il tracciamento

Segui i passaggi riportati sotto per scaricare, creare e implementare un'applicazione Hello World in Python di esempio con i moduli Powertools per AWS Lambda (Python) integrati utilizzando AWS SAM. Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando si invia una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e parametri a CloudWatch utilizzando Embedded Metric Format e invia le tracce a AWS X-Ray. La funzione restituisce un messaggio hello world.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Python 3.11
- AWS CLI versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della CLI AWS SAM, consulta Aggiornamento della CLI AWS SAM.

Implementare un'applicazione AWS SAM di esempio

1. Inizializza l'applicazione utilizzando il modello Python Hello World.

```
sam init --app-template hello-world-powertools-python --name sam-app --package-type
 Zip --runtime python3.11 --no-tracing
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi `Enter`.

> **Note**
>
> Per HelloWorldFunction potrebbe non avere l'autorizzazione definita, va bene?, assicurati di inserire `y`.

5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Richiama l'endpoint dell'API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere le tracce per la funzione, esegui sam traces.

```
sam traces
```

L'output della traccia ha il seguente aspetto:

```
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. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non è possibile configurare la frequenza di campionamento di X-Ray per le funzioni.

Utilizzo di Powertools per AWS Lambda (Python) e del AWS CDK per il tracciamento

Segui i passaggi riportati sotto per scaricare, creare e implementare un'applicazione Hello World in Python di esempio con i moduli Powertools per AWS Lambda (Python) integrati utilizzando AWS CDK. Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando si invia una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e parametri a CloudWatch utilizzando Embedded Metric Format e invia le tracce a AWS X-Ray. La funzione restituisce un messaggio hello world.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Python 3.11
- AWS CLI versione 2
- AWS CDK versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della CLI AWS SAM, consulta Aggiornamento della CLI AWS SAM.

Implementare un'applicazione AWS CDK di esempio

1. Crea una directory di progetto per la nuova applicazione.

```
mkdir hello-world
cd hello-world
```

2. Inizializza l'app.

```
cdk init app --language python
```

3. Installa le dipendenze di Python.

```
pip install -r requirements.txt
```

4. Crea una directory lambda_function nella cartella root.

```
mkdir lambda_function
```

```
cd lambda_function
```

5. Crea un file app.py e aggiungi il seguente codice al file. Questo è il codice per la funzione Lambda.

```
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. Apri la directory hello_world. Dovrebbe essere visualizzato un file denominato hello_world_stack.py.

```
cd ..
cd hello_world
```

7. Apri hello_world_stack.py e aggiungi il seguente codice al file. Contiene Lambda Constructor, che crea la funzione Lambda, configura le variabili di ambiente per Powertools e imposta la conservazione dei log su una settimana, e ApiGatewayv1 Constructor, che crea la 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. Distribuisci l'applicazione.

```
cd ..
cdk deploy
```

9. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?OutputKey==`apiUrl`].OutputValue' --output text
```

10. Richiama l'endpoint dell'API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

11. Per ottenere le tracce per la funzione, esegui sam traces.

```
sam traces
```

L'output delle tracce ha la struttura seguente:

```
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. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
cdk destroy
```

Utilizzo di ADOT per strumentare le funzioni Python

ADOT fornisce livelli Lambda completamente gestiti che mettono insieme tutto il necessario per raccogliere i dati di telemetria utilizzando l'SDK OTel. Usando questo livello, è possibile strumentare le funzioni Lambda senza dover modificare alcun codice funzione. È inoltre possibile configurare il livello per eseguire l'inizializzazione personalizzata di OTel. Per ulteriori informazioni, consulta la sezione relativa alla configurazione personalizzata per ADOT Collector su Lambda nella documentazione di ADOT.

Per i runtime Python, è possibile aggiungere il livello Lambda gestito da AWS per ADOT Python per la strumentazione automatica delle funzioni. Questo livello funziona sia per architetture arm64 che x86_64. Per istruzioni dettagliate su come aggiungere questo livello, consulta la sezione relativa a AWS Distro per il supporto Lambda OpenTelemetry per Python nella documentazione di ADOT.

Utilizzo dell'SDK X-Ray per strumentare le funzioni Python

Per registrare i dettagli sulle chiamate effettuate dalla funzione Lambda ad altre risorse nell'applicazione, è anche possibile utilizzare il SDK AWS X-Ray per Python. Per ottenere l'SDK, aggiungere il pacchetto `aws-xray-sdk` alle dipendenze dell'applicazione.

Example requirements.txt

```
jsonpickle==1.3
aws-xray-sdk==2.4.3
```

Nel codice della funzione, puoi strumentare i client SDK AWS tramite patch alla libreria `boto3` con il modulo `aws_xray_sdk.core`.

Example funzione: tracciamento di un client SDK AWS

```
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)))
  ...
```

Dopo aver aggiunto le dipendenze corrette e aver apportato le modifiche necessarie al codice, attivare il tracciamento nella configurazione della funzione tramite la console Lambda o l'API.

Attivazione del tracciamento con la console Lambda

Per attivare il tracciamento attivo sulla funzione Lambda con la console, attenersi alla seguente procedura:

Per attivare il tracciamento attivo

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione) e quindi Monitoring and operations tools (Strumenti di monitoraggio e operazioni).
4. Nel riquadro Strumenti di monitoraggio aggiuntivi scegli Modifica.
5. In CloudWatch Application Signals AWS X-Ray e, scegli Enable for Lambda service trace.
6. Selezionare Salva.

Attivazione del tracciamento con l'API Lambda

Configurare il tracciamento sulla funzione Lambda con la AWS CLI o SDK AWS e utilizzare le seguenti operazioni API:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Il seguente comando AWS CLI di esempio consente il tracciamento attivo su una funzione denominata my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

La modalità di tracciamento fa parte della configurazione specifica della versione quando si pubblica una versione della funzione. Non è possibile modificare la modalità di tracciamento in una versione pubblicata.

Attivazione del tracciamento con CloudFormation

Per attivare il tracciamento su una risorsa `AWS::Lambda::Function` in un modello CloudFormation, utilizzare la proprietà `TracingConfig`.

Example function-inline.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Per una risorsa AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, utilizzare la proprietà `Tracing`.

Example template.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretazione di una traccia X-Ray

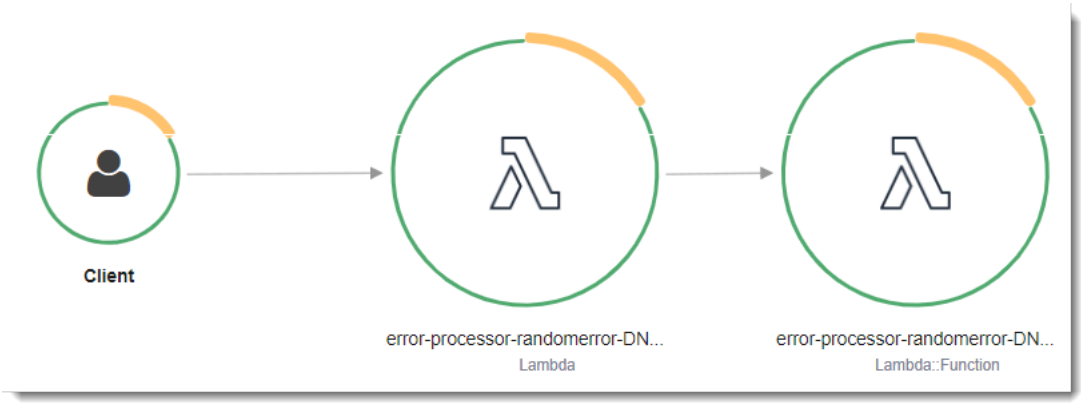
La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al ruolo di esecuzione della funzione. In caso contrario, aggiungere la policy AWSXRayDaemonWriteAccess al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico dei servizi X-Ray mostra informazioni sull'applicazione e tutti i suoi componenti. Il seguente esempio mostra un'applicazione con due funzioni. La funzione principale elabora gli eventi e talvolta restituisce errori. La seconda funzione elabora gli errori che appaiono nel gruppo di log della prima funzione e utilizza l'SDK AWS per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon CloudWatch Logs.

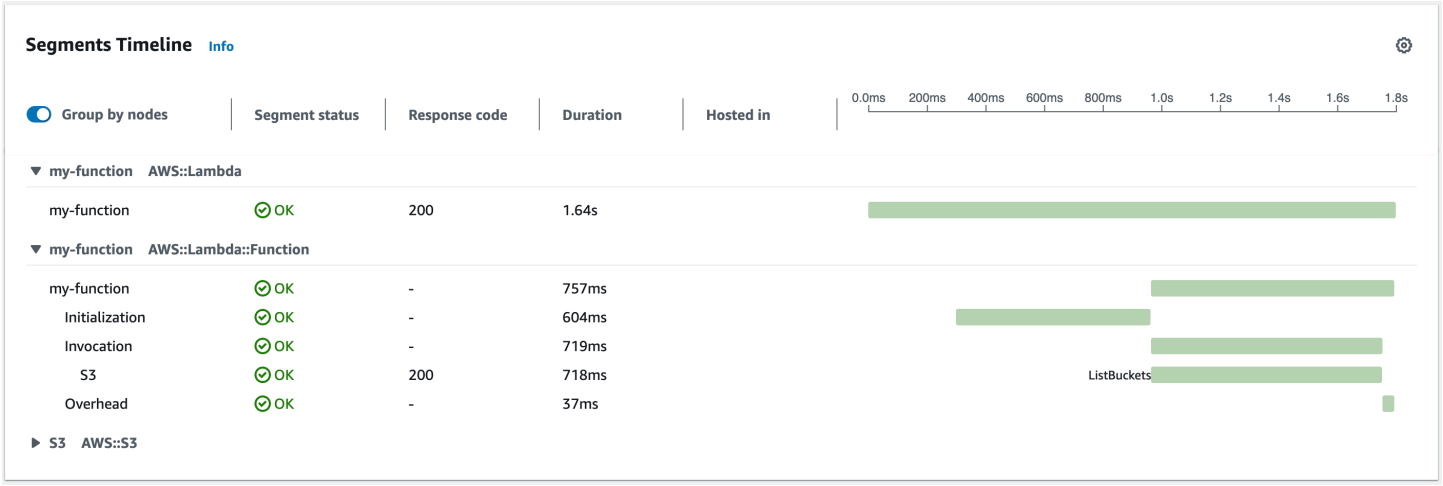


X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non è possibile configurare la frequenza di campionamento di X-Ray per le funzioni.

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda. L'esempio seguente mostra una traccia con questi 2 segmenti. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.



Questo esempio espande il segmento `AWS::Lambda::Function` per visualizzare i relativi tre sottosegmenti.

> **Note**
>
> AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. La traccia di esempio mostrata qui illustra il segmento di funzione vecchio stile. Le differenze tra i segmenti vecchio e nuovo stile sono descritte nei paragrafi seguenti.
> Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni in tutte le Regioni AWS, ad eccezione delle regioni Cina e GovCloud, passeranno all'utilizzo dei messaggi di log e dei segmenti di traccia di nuovo formato.

Il segmento di funzioni vecchio stile contiene i seguenti sottosegmenti:

- Inizializzazione – Rappresenta il tempo trascorso a caricare la funzione e ad eseguire il codice di inizializzazione. Questo sottosegmento viene visualizzato solo per il primo evento che viene elaborato da ogni istanza della funzione.

- Chiamata: rappresenta il tempo impiegato per eseguire il codice del gestore.
- Overhead: rappresenta il tempo impiegato dal runtime Lambda per prepararsi a gestire l'evento successivo.

Il segmento di funzione di nuovo stile non contiene un sottosegmento `Invocation`. I sottosegmenti dei clienti sono invece collegati direttamente al segmento di funzioni. Per ulteriori informazioni sulla struttura dei segmenti di funzioni vecchio e nuovo stile, consulta the section called “Informazioni sui monitoraggi di X-Ray”.

È inoltre possibile strumentare i client HTTP, registrare query SQL e creare segmenti secondari personalizzati con annotazioni e metadati. Per ulteriori informazioni, consulta SDK AWS X-Ray per Python nella Guida per gli sviluppatori di AWS X-Ray.

> **Prezzi**
>
> È possibile utilizzare il tracciamento X-Ray gratuitamente ogni mese fino a un certo limite come parte del piano gratuito di AWS. Oltre la soglia, X-Ray addebita lo storage di traccia e il recupero. Per ulteriori informazioni, consultare Prezzi di AWS X-Ray.

Memorizzazione delle dipendenze di runtime in un livello (SDK X-Ray)

Se si utilizza l'SDK X-Ray per strumentare i client SDK AWS del codice della funzione, il pacchetto di implementazione può diventare piuttosto grande. Per evitare di caricare dipendenze di runtime ogni volta che si aggiorna il codice della funzione, includere l'SDK X-Ray in un livello Lambda.

L'esempio seguente mostra una risorsa `AWS::Serverless::LayerVersion` che memorizza SDK AWS X-Ray per Python.

Example template.yml – Livello delle dipendenze

```
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
```

Con questa configurazione, si aggiorna il livello della libreria solo se si modificano le dipendenze di runtime. Poiché il pacchetto di implementazione della funzione contiene solo il codice, questo può contribuire a ridurre i tempi di caricamento.

La creazione di un layer per le dipendenze richiede modifiche alla compilazione per generare l'archivio dei layer prima della distribuzione. Per un esempio funzionante, vedere l'applicazione di esempio blank-python .

Compilazione di funzioni Lambda con Ruby

Puoi eseguire il codice Ruby in AWS Lambda. Lambda fornisce Runtime per Ruby che eseguono il tuo codice per elaborare gli eventi. Il codice viene eseguito in un ambiente che include AWS SDK per Ruby, con le credenziali di un ruolo AWS Identity and Access Management (IAM) che gestisci. Per saperne di più sulle versioni degli SDK incluse nei runtime di Ruby, consulta the section called "Versioni SDK incluse nel runtime".

Lambda supporta i seguenti runtime di Ruby.

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Ruby 3.4	`ruby3.4`	Amazon Linux 2023	31 marzo 2028	30 aprile 2028	31 maggio 2028
Ruby 3.3	`ruby3.3`	Amazon Linux 2023	31 marzo 2027	30 aprile 2027	31 maggio 2027
Ruby 3.2	`ruby3.2`	Amazon Linux 2	31 marzo 2026	31 agosto 2026	30 settembre 2026

Per creare una funzione Ruby

1. Aprire la console Lambda.
2. Scegli Crea funzione.
3. Configura le impostazioni seguenti:
   - Nome della funzione: inserisci il nome della funzione.
   - Runtime: scegli Ruby 3.4.
4. Scegli Crea funzione.

La console crea una funzione Lambda con un singolo file di origine denominato `lambda_function.rb`. È possibile modificare questo file e aggiungere altri file nell'editor di codice

predefinito. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione. Quindi, per eseguire il codice, scegli Crea evento di test nella sezione EVENTI DI TEST.

Il file `lambda_function.rb` esporta una funzione denominata `lambda_handler` che richiede un oggetto evento e un oggetto contesto. Questa è la funzione del gestore chiamata da Lambda quando la funzione viene richiamata. Il runtime della funzione Ruby riceve gli eventi di chiamata da Lambda e li passa al gestore. Nella configurazione della funzione il valore del gestore è `lambda_function.lambda_handler`.

Quando si salva il codice funzione, la console Lambda crea un pacchetto di implementazione dell'archivio di file .zip. Quando sviluppi il codice funzione al di fuori della console (utilizzando un IDE) devi creare un pacchetto di implementazione per caricare il codice nella funzione Lambda.

Il runtime della funzione passa un oggetto contesto al gestore, oltre all'evento di chiamata. L'oggetto contesto contiene ulteriori informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. Altre informazioni sono disponibili con le variabili di ambiente.

La funzione Lambda include un gruppo di CloudWatch log Logs. Il runtime della funzione invia i dettagli su ogni chiamata a Logs. CloudWatch Si trasmette qualsiasi log che la tua funzione emette durante la chiamata. Se la funzione restituisce un errore, Lambda formatta l'errore e lo restituisce al chiamante.

Argomenti

- Versioni SDK incluse nel runtime
- Abilitazione di Yet Another Ruby JIT (YJIT)
- Definire l'handler di funzioni Lambda in Ruby
- Distribuire le funzioni Ruby Lambda con gli archivi di file .zip
- Distribuisci funzioni Lambda per Ruby con immagini di container
- Utilizzo dei livelli per le funzioni Lamba di Ruby
- Utilizzo dell'oggetto del contesto Lambda per recuperare le informazioni sulla funzione Ruby
- Registrare e monitorare le funzioni Lambda con Ruby
- Strumentazione del codice Ruby in AWS Lambda

Versioni SDK incluse nel runtime

La versione dell' AWS SDK inclusa nel runtime di Ruby dipende dalla versione di runtime e dalla tua. Regione AWS L' AWS SDK for Ruby è progettato per essere modulare ed è separato da. Servizio AWS Per trovare la versione di una determinata gemma di servizio inclusa nel runtime che stai utilizzando, crea una funzione Lambda con il codice nel seguente formato. Sostituisci `aws-sdk-s3` e `Aws::S3` con il nome delle gemme di servizio utilizzate dal tuo codice.

```
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
```

Abilitazione di Yet Another Ruby JIT (YJIT)

Il runtime di Ruby 3.2 supporta YJIT, un compilatore Ruby JIT leggero e minimalista. YJIT offre prestazioni significativamente più elevate, ma utilizza anche più memoria rispetto all'interprete Ruby. YJIT è consigliato per i carichi di lavoro Ruby on Rails.

YJIT non è abilitato per impostazione predefinita. Per abilitare YJIT per una funzione Ruby 3.2, imposta la variabile di ambiente `RUBY_YJIT_ENABLE` su `1`. Per verificare che YJIT sia abilitato, stampa il risultato del metodo `RubyVM::YJIT.enabled?`.

Example - Conferma che YJIT è abilitato

```
puts(RubyVM::YJIT.enabled?())
# => true
```

Definire l'handler di funzioni Lambda in Ruby

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Argomenti

- Nozioni di base sull'handler Ruby
- Best practice di codice per le funzioni Lambda con Ruby

Nozioni di base sull'handler Ruby

In questo esempio il file `function.rb` definisce un metodo del gestore denominato `handler`. La funzione del gestore richiede due oggetti come input e restituisce un documento JSON.

Example function.rb

```
require 'json'

def handler(event:, context:)
    { event: JSON.generate(event), context: JSON.generate(context.inspect) }
end
```

Nella configurazione della funzione, l'impostazione `handler` indica a Lambda dove trovare il gestore. Nell'esempio precedente, il valore corretto per questa impostazione è **`function.handler`**. Include due nomi separati da un punto: il nome del file e il nome del metodo del gestore.

È inoltre possibile definire il metodo del gestore in una classe. L'esempio seguente definisce il metodo del gestore `process` nella classe `Handler` nel modulo `LambdaFunctions`.

Example source.rb

```
module LambdaFunctions
  class Handler
    def self.process(event:,context:)
      "Hello!"
    end
  end
end
```

In questo caso, l'impostazione del gestore è **`source.LambdaFunctions::Handler.process`**.

I due oggetti che il gestore accetta sono il contesto e l'evento di chiamata. L'evento è un oggetto Ruby che contiene il payload fornito dal chiamante. Se il payload è un documento JSON, l'oggetto evento è un hash Ruby. In caso contrario, è una stringa. L'oggetto contesto include i metodi e le proprietà che forniscono le informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

Il gestore della funzione viene eseguito ogni volta che la funzione Lambda viene richiamata. Il codice statico all'esterno del gestore viene eseguito una volta per istanza della funzione. Se il gestore utilizza risorse come client SDK e connessioni database, puoi crearle al di fuori del metodo del gestore per riutilizzarle per più chiamate.

Ogni istanza della funzione può elaborare più eventi di chiamata, ma elabora un solo evento alla volta. Il numero di istanze che elaborano un evento in un dato momento costituisce la simultaneità della funzione. Per ulteriori informazioni sull'ambiente di esecuzione Lambda, consulta Comprendere il ciclo di vita dell'ambiente di esecuzione Lambda.

Best practice di codice per le funzioni Lambda con Ruby

Segui le linee guida riportate nell'elenco seguente per utilizzare le best practice di codifica durante la creazione delle funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. In questo modo è possibile creare una funzione di cui è più semplice eseguire l'unit test. Ad esempio, in Ruby, l'aspetto è analogo al seguente:

```
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
```

- Controllare le dipendenze nel pacchetto di distribuzione della funzione. L'ambiente di esecuzione AWS Lambda contiene diverse librerie. Per il runtime Ruby, queste includono l'SDK AWS. Per abilitare il set di caratteristiche e aggiornamenti della sicurezza più recenti, Lambda aggiorna

periodicamente tali librerie. Tali aggiornamenti possono introdurre lievi modifiche al comportamento della funzione Lambda. Per mantenere il controllo completo delle dipendenze utilizzate dalla funzione, inserire tutte le dipendenze nel pacchetto di implementazione.

- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del contesto di esecuzione.
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. In questo modo viene ridotta la quantità di tempo necessaria per il download del pacchetto e per la relativa decompressione prima dell'invocazione. Per le funzioni create in Ruby, evitare di caricare l'intera libreria dell'SDK AWS come parte del pacchetto di implementazione. Dipendono invece in modo selettivo dalle gemme che prelevano i componenti dell'SDK necessari (ad esempio le gemme SDK DynamoDB e Amazon S3).

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedere Riutilizzo delle connessioni con Keep-Alive in Node.js.

Utilizzare le variabili di ambiente per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su `0` per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare API non documentate e non pubbliche nel codice della funzione Lambda. Per i tempi di esecuzione gestiti AWS Lambda, Lambda applica periodicamente aggiornamenti di sicurezza e funzionalità alle API interne di Lambda. Questi aggiornamenti API interni potrebbero essere incompatibili con le versioni precedenti, causando conseguenze indesiderate come errori di chiamata se la funzione ha una dipendenza su queste API non pubbliche. Consulta il riferimento all'API per un elenco di API disponibili pubblicamente.

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda?.

Distribuire le funzioni Ruby Lambda con gli archivi di file .zip

Il codice della funzione AWS Lambda comprende un file .rb contenente il codice del gestore della funzione, insieme a tutti i moduli aggiuntivi (gemme) da cui dipende il codice. Per implementare questo codice della funzione in Lambda, utilizza un pacchetto di implementazione. Questo pacchetto può essere un archivio di file .zip o un'immagine di container. Per ulteriori informazioni sull'uso delle immagini di container con Ruby, consulta la pagina Implementazione di funzioni Lambda in Ruby con immagini di container.

Per creare un pacchetto di implementazione come archivio di file .zip, puoi utilizzare l'utilità di archiviazione di file .zip incorporata del tuo strumento della linea di comando o qualsiasi altra utilità file .zip, come ad esempio 7zip. Gli esempi mostrati nelle sezioni seguenti presuppongono che tu stia utilizzando uno strumento della linea di comando `zip` in un ambiente Linux o MacOS. Per utilizzare gli stessi comandi in Windows, puoi installare il sottosistema Windows per Linux per ottenere una versione di Ubuntu e Bash integrata con Windows.

Nota che Lambda utilizza le autorizzazioni dei file POSIX, quindi potresti aver bisogno di impostare le autorizzazioni per la cartella del pacchetto di implementazione prima di creare l'archivio di file .zip.

I comandi di esempio nelle sezioni seguenti utilizzano l'utilità Creatore di bundle per aggiungere dipendenze al pacchetto di implementazione. Per installare il creatore di bundle, esegui il comando sotto riportato.

```
gem install bundler
```

Sezioni

- Dipendenze in Ruby
- Creazione di un pacchetto di implementazione .zip senza dipendenze
- Creazione di un pacchetto di implementazione .zip con dipendenze
- Creazione di un livello Ruby per dipendenze
- Creazione di un pacchetto di implementazione .zip con librerie native
- Creazione e aggiornamento delle funzioni Lambda di Ruby utilizzando file .zip

Dipendenze in Ruby

Per le funzioni Lambda che utilizzano il runtime di Ruby, una dipendenza può essere una qualsiasi gemma Ruby. Se implementi la funzione utilizzando un archivio .zip, puoi aggiungere queste

dipendenze al file .zip con il tuo codice della funzione o utilizzare un livello Lambda. Un livello è un file .zip separato che può contenere codice aggiuntivo o altri contenuti. Per maggiori informazioni sull'uso dei livelli Lambda, consulta Livelli Lambda.

Il runtime di Ruby include AWS SDK per Ruby. Se la tua funzione utilizza l'SDK, non è necessario raggrupparla con il codice. Tuttavia, per mantenere il pieno controllo delle tue dipendenze o per utilizzare una versione specifica dell'SDK, puoi aggiungerlo al pacchetto di implementazione della tua funzione. Puoi includere l'SDK nel tuo file .zip o aggiungerlo utilizzando un livello Lambda. Le dipendenze nel file .zip o nei livelli Lambda hanno la precedenza sulle versioni incluse nel runtime. Per vedere quale versione dell'SDK per Ruby è inclusa nella tua versione di runtime, consulta the section called “Versioni SDK incluse nel runtime”.

In base al modello di responsabilità condivisa di AWS, è tua responsabilità gestire eventuali dipendenze nei pacchetti di implementazione delle tue funzioni. Ciò include l'applicazione di aggiornamenti e patch di sicurezza. Per aggiornare le dipendenze nel pacchetto di implementazione della funzione, crea prima un nuovo file .zip e poi caricalo su Lambda. Per ulteriori informazioni, consulta Creazione di un pacchetto di implementazione .zip con dipendenze e Creazione e aggiornamento delle funzioni Lambda di Ruby utilizzando file .zip.

Creazione di un pacchetto di implementazione .zip senza dipendenze

Se il codice della funzione non ha dipendenze, il file .zip contiene solo il file .rb con il codice del gestore della funzione. Utilizza il tuo strumento di compressione preferito per creare un file .zip con il file .rb nella directory principale. Se il file .rb non si trova nella directory principale del file .zip, Lambda non sarà in grado di eseguire il codice.

Per informazioni su come implementare il file .zip per creare una nuova funzione Lambda o aggiornarne una esistente, consulta la sezione Creazione e aggiornamento delle funzioni Lambda di Ruby utilizzando file .zip.

Creazione di un pacchetto di implementazione .zip con dipendenze

Se il codice della funzione dipende da gem Ruby aggiuntivi, puoi aggiungere queste dipendenze al file .zip con il codice della funzione o utilizzare un livello Lambda. Le istruzioni in questa sezione mostrano come includere le dipendenze nel pacchetto di implementazione .zip. Per istruzioni sull’inclusione di dipendenze in un livello, consulta the section called “Creazione di un livello Ruby per dipendenze”.

Supponiamo che il codice della funzione sia salvato in un file denominato `lambda_function.rb` nella directory del progetto. I seguenti comandi della CLI di esempio creano un file .zip denominato `my_deployment_package.zip` contenente il codice della funzione e le relative dipendenze.

Per creare il pacchetto di implementazione

1. Nella directory del progetto, crea un `Gemfile` in cui specificare le tue dipendenze.

   ```
   bundle init
   ```

2. Utilizzando il tuo editor di testo preferito, modifica il `Gemfile` per specificare le dipendenze della tua funzione. Ad esempio, per utilizzare la gemma TZInfo, modifica il tuo `Gemfile` in modo che assomigli a quanto segue.

   ```
   source "https://rubygems.org"
   gem "tzinfo"
   ```

3. Esegui il comando sotto riportato per installare le gemme specificate nel `Gemfile` nella tua directory del progetto. Questo comando imposta `vendor/bundle` come percorso predefinito per le installazioni delle gemme.

   ```
   bundle config set --local path 'vendor/bundle' && bundle install
   ```

   Verrà visualizzato un output simile al seguente.

   ```
   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**
   >
   > Per installare nuovamente le gemme a livello globale in un secondo momento, esegui il comando sotto riportato.
   >
   > ```
   > bundle config set --local system 'true'
   > ```

4. Crea un archivio di file .zip contenente il file `lambda_function.rb` con il codice del gestore della funzione e le dipendenze che hai installato nel passaggio precedente.

```
zip -r my_deployment_package.zip lambda_function.rb vendor
```

Verrà visualizzato un output simile al seguente.

```
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%)
...
```

Creazione di un livello Ruby per dipendenze

Per informazioni su come impacchettare le dipendenze di Ruby in un livello Lambda, consulta the section called “Livelli”.

Creazione di un pacchetto di implementazione .zip con librerie native

Molte gemme Ruby comuni, come `nokogiri`, `nio4r` e `mysql`, contengono estensioni native scritte in C. Quando aggiungi librerie contenenti codice C al pacchetto di implementazione, devi creare il pacchetto correttamente per assicurarti che sia compatibile con l'ambiente di esecuzione Lambda.

Per le applicazioni di produzione, consigliamo di creare e implementare il codice utilizzando AWS Serverless Application Model (AWS SAM). In AWS SAM, usa l'opzione `sam build --use-container` per creare la tua funzione all'interno di un container Docker simile a Lambda. Per ulteriori informazioni sull'utilizzo di AWS SAM per implementare il codice della funzione, consulta la pagina Creazione di applicazioni nella Guida per gli sviluppatori di AWS SAM.

In alternativa, per creare un pacchetto di implementazione .zip contenente gemme con estensioni native senza utilizzare AWS SAM, puoi usare un container per raggruppare le tue dipendenze nello stesso ambiente del runtime Lambda di Ruby. Per completare questi passaggi, Docker deve essere installato sulla tua macchina di compilazione. Per ulteriori informazioni sull'installazione di Docker, consulta la pagina Install Docker Engine.

Creazione di un pacchetto di implementazione .zip in un container Docker

1. Sulla tua macchina di compilazione locale, crea una cartella in cui salvare il container. All'interno di quella cartella, crea un file denominato `dockerfile` e incolla il seguente codice.

```
FROM public.ecr.aws/sam/build-ruby3.2:latest-x86_64
RUN gem update bundler
CMD "/bin/bash"
```

2. All'interno della cartella in cui hai creato il tuo `dockerfile`, esegui il comando sotto riportato per creare il container Docker.

```
docker build -t awsruby32 .
```

3. Accedi alla directory del progetto contenente il file `.rb` con il codice del gestore della funzione e il `Gemfile` specificato nelle dipendenze della funzione. Dall'interno di quella directory, esegui il comando sotto riportato per avviare il container Lambda Ruby.

   Linux/MacOS

   ```
   docker run --rm -it -v $PWD:/var/task -w /var/task awsruby32
   ```

   > **Note**
   >
   > In MacOS, potresti visualizzare un avviso che ti informa che la piattaforma dell'immagine richiesta non corrisponde alla piattaforma host rilevata. Ignora questo avviso.

   Windows PowerShell

   ```
   docker run --rm -it -v ${pwd}:var/task -w /var/task awsruby32
   ```

   All'avvio del container, dovresti vedere un prompt bash.

   ```
   bash-4.2#
   ```

4. Configura l'utilità di creazione di bundle per installare le gemme specificate nel `Gemfile` in una directory `vendor/bundle` locale e installare le tue dipendenze.

```
bash-4.2# bundle config set --local path 'vendor/bundle' && bundle install
```

5. Crea il pacchetto di implementazione .zip con il codice della funzione e le relative dipendenze. In questo esempio, il file contenente il codice del gestore della funzione è denominato `lambda_function.rb`.

```
bash-4.2# zip -r my_deployment_package.zip lambda_function.rb vendor
```

6. Esci dal container e torna alla directory del progetto locale.

```
bash-4.2# exit
```

   Ora puoi usare il pacchetto di implementazione di file .zip per creare o aggiornare la tua funzione Lambda. Per informazioni, consultare Creazione e aggiornamento delle funzioni Lambda di Ruby utilizzando file .zip

Creazione e aggiornamento delle funzioni Lambda di Ruby utilizzando file .zip

Dopo aver creato il pacchetto di implementazione .zip, puoi utilizzarlo per creare una nuova funzione Lambda o aggiornarne una esistente. È possibile caricare il pacchetto .zip utilizzando la console Lambda, l'AWS Command Line Interface e l'API Lambda. Puoi anche creare e aggiornare le funzioni Lambda usando AWS Serverless Application Model (AWS SAM) e CloudFormation.

La dimensione massima per un pacchetto di implementazione .zip per Lambda è di 250 MB (dopo l'estrazione). Nota che questo limite si applica alla dimensione combinata di tutti i file caricati, inclusi eventuali livelli Lambda.

Il runtime Lambda necessita dell'autorizzazione per leggere i file nel pacchetto di distribuzione. Nella notazione ottale delle autorizzazioni di Linux, Lambda richiede 644 autorizzazioni per i file non eseguibili (rw-r--r--) e 755 autorizzazioni (rwxr-xr-x) per le directory e i file eseguibili.

In Linux e macOS, utilizza il comando `chmod` per modificare le autorizzazioni file su file e directory nel pacchetto di implementazione. Ad esempio, per assegnare a un file non eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 644 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta Set, View, Change, or Remove Permissions on an Object nella documentazione di Microsoft Windows.

> **Note**
>
> Se non concedi a Lambda le autorizzazioni necessarie per accedere alle directory del pacchetto di implementazione, Lambda imposta le autorizzazioni per tali directory su 755 (rwxr-xr-x).

Creazione e aggiornamento delle funzioni con file .zip utilizzando la console

Per creare una nuova funzione, devi prima creare la funzione nella console, quindi devi caricare il tuo archivio .zip. Per aggiornare una funzione esistente, apri la pagina relativa alla funzione, quindi segui la stessa procedura per aggiungere il file .zip aggiornato.

Se il file .zip ha dimensioni inferiori a 50 MB, è possibile creare o aggiornare una funzione caricando il file direttamente dal computer locale. Per i file .zip di dimensioni superiori a 50 MB, prima è necessario caricare il pacchetto in un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando la Console di gestione AWS, consulta la sezione Nozioni di base su Amazon S3. Per caricare file utilizzando la AWS CLI, consulta la sezione Spostamento di oggetti nella Guida per l'utente di AWS CLI.

> **Note**
>
> Non è possibile modificare il tipo di pacchetto di implementazione (.zip o immagine di container) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine di container esistente per utilizzare un archivio di file .zip. È necessario creare una nuova funzione.

Creazione di una nuova funzione (console)

1. Apri la pagina Funzioni della console Lambda e scegli Crea funzione.
2. Scegli Author from scratch (Crea da zero).

3. In Basic information (Informazioni di base) eseguire queste operazioni:
   a. In Nome funzione, inserisci il nome della funzione.
   b. Per Runtime, seleziona il runtime che desideri utilizzare.
   c. (Facoltativo) Per Architettura, scegli l'architettura del set di istruzioni per la funzione. L'architettura predefinita è x86_64. Assicurati che il pacchetto di implementazione per la tua funzione sia compatibile con l'architettura del set di istruzioni scelta.
4. (Opzionale) In Autorizzazioni espandere Modifica ruolo di esecuzione predefinito. Puoi creare un nuovo ruolo di esecuzione o utilizzare un ruolo esistente.
5. Scegli Crea funzione. Lambda crea una funzione di base "Hello world" utilizzando il runtime scelto.

Caricamento di un archivio .zip dal computer locale (console)

1. Nella pagina Funzioni della console Lambda, scegli la funzione per cui vuoi caricare il file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli File .zip.
5. Per caricare il file .zip, procedi come segue:
   a. Seleziona Carica, quindi seleziona il tuo file .zip nel selettore di file.
   b. Seleziona Apri.
   c. Selezionare Salva.

Caricamento di un archivio .zip da un bucket Amazon S3 (console)

1. Nella pagina Funzioni della console Lambda, scegli la funzione per cui vuoi caricare un nuovo file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli Posizione Amazon S3.
5. Incolla l'URL del link Amazon S3 del tuo file .zip e scegli Salva.

Aggiornamento delle funzioni dei file .zip tramite l'editor di codice della console

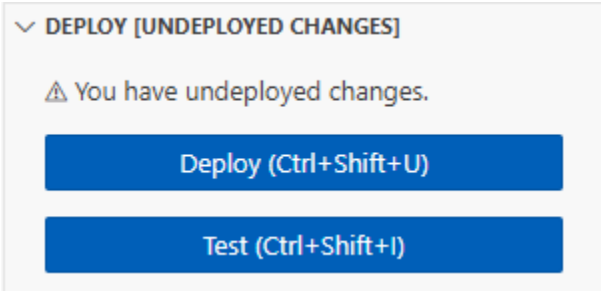
Per alcune funzioni con pacchetti di implementazione .zip, puoi utilizzare l'editor di codice integrato nella console Lambda per aggiornare direttamente il codice della funzione. Per utilizzare questa funzione, la funzione deve soddisfare i seguenti criteri:

- La funzione deve utilizzare uno dei runtime del linguaggio interpretato (Python, Node.js o Ruby)
- Il pacchetto di implementazione della funzione deve avere dimensioni inferiori a 50 MB (non compresso).

Il codice della funzione per le funzioni con pacchetti di implementazione di immagini di container non può essere modificato direttamente nella console.

Aggiornamento del codice della funzione utilizzando l'editor di codice della console

1. Apri la pagina Funzioni della console Lambda e scegli la tua funzione.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, seleziona il tuo file di codice sorgente e modificalo nell'editor di codice integrato.
4. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



Creazione e aggiornamento delle funzioni con file .zip utilizzando la AWS CLI

È possibile utilizzare la AWS CLI per creare una nuova funzione o aggiornare una funzione esistente mediante un file .zip. Utilizza i comandi create-function e update-function-code per implementare il tuo pacchetto .zip. Se il file .zip ha dimensioni inferiori a 50 MB, è possibile caricare il pacchetto .zip da una posizione di file nella macchina di compilazione locale. Per i file di dimensioni maggiori, è necessario caricare il pacchetto .zip da un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando la AWS CLI, consulta la sezione Spostamento di oggetti nella Guida per l'utente di AWS CLI.

> **Note**
>
> Se carichi il file .zip da un bucket Amazon S3 utilizzando la AWS CLI, il bucket deve trovarsi nella stessa Regione AWS della funzione.

Per creare una nuova funzione mediante un file .zip con la AWS CLI, è necessario specificare quanto segue:

- Il nome della funzione (`--function-name`)
- Il runtime della tua funzione (`--runtime`)
- Il nome della risorsa Amazon (ARN) del ruolo di esecuzione della funzione (`--role`)
- Il nome del metodo del gestore nel codice della funzione (`--handler`)

È inoltre necessario specificare la posizione del file .zip. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
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
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza l'opzione `--code` illustrata nel seguente comando di esempio. È necessario utilizzare il parametro `S3ObjectVersion` solo per gli oggetti con controllo delle versioni.

```
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
```

Per aggiornare una funzione esistente mediante la CLI, specifica il nome della funzione utilizzando il parametro `--function-name`. È inoltre necessario specificare la posizione del file .zip che desideri utilizzare per aggiornare il codice della funzione. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza le opzioni `--s3-bucket` e `--s3-key` come illustrato nel seguente comando di esempio. È necessario utilizzare il parametro `--s3-object-version` solo per gli oggetti con controllo delle versioni.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Creazione e aggiornamento delle funzioni con file .zip utilizzando l'API Lambda

Per creare e aggiornare le funzioni mediante un archivio di file .zip, utilizza le seguenti operazioni API:

- CreateFunction
- UpdateFunctionCode

Creazione e aggiornamento delle funzioni con file .zip utilizzando AWS SAM

AWS Serverless Application Model (AWS SAM) è un toolkit che aiuta a semplificare il processo di creazione ed esecuzione di applicazioni serverless su AWS. Definisci le risorse per la tua applicazione in un modello YAML o JSON e usa l'interfaccia della linea di comando AWS SAM (AWS SAM CLI) per compilare, creare pacchetti e implementare le tue applicazioni. Quando crei una funzione Lambda da un modello AWS SAM, AWS SAM crea automaticamente un pacchetto di implementazione .zip o un'immagine di container con il codice della funzione e le dipendenze specificate. Per ulteriori informazioni sull'utilizzo di AWS SAM per creare e implementare funzioni Lambda, consulta la pagina Nozioni di base su AWS SAM nella Guida introduttiva per gli sviluppatori di AWS Serverless Application Model.

Puoi anche utilizzare AWS SAM per creare una funzione Lambda mediante un archivio di file .zip esistente. Per creare una funzione Lambda utilizzando AWS SAM, puoi salvare il tuo file .zip in un bucket Amazon S3 o in una cartella locale sul tuo computer di compilazione. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando la AWS CLI, consulta la sezione Spostamento di oggetti nella Guida per l'utente di AWS CLI.

Nel modello AWS SAM, la risorsa `AWS::Serverless::Function` specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `CodeUri`: imposta il valore sull'URI Amazon S3, sul percorso della cartella locale o sull'oggetto FunctionCode del codice della funzione
- `Runtime`: imposta il runtime prescelto

Con AWS SAM, se il file .zip è superiore a 50 MB, non è necessario caricarlo prima in un bucket Amazon S3. AWS SAM può caricare pacchetti .zip fino alla dimensione massima consentita di 250 MB (dopo l'estrazione) da una posizione sulla macchina di compilazione locale.

Per ulteriori informazioni sull'implementazione di funzioni mediante il file .zip in AWS SAM, consulta la sezione AWS::Serverless::Function nella Guida per gli sviluppatori di AWS SAM.

Creazione e aggiornamento delle funzioni con file .zip utilizzando CloudFormation

È possibile utilizzare CloudFormation per creare una funzione Lambda mediante un archivio di file .zip. Per creare una funzione Lambda da un file .zip, devi prima caricare il file su un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando la AWS CLI, consulta la sezione Spostamento di oggetti nella Guida per l'utente di AWS CLI.

Nel modello CloudFormation, la risorsa `AWS::Lambda::Function` specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `Code`: inserisci il nome del bucket Amazon S3 e il nome del file .zip nei campi `S3Bucket` e `S3Key`
- `Runtime`: imposta il runtime prescelto

Il file .zip che CloudFormation genera non può superare i 4 MB. Per ulteriori informazioni sull'implementazione di funzioni mediante un file .zip in CloudFormation, consulta la sezione AWS::Lambda::Function nella Guida per gli sviluppatori di CloudFormation.

Distribuisci funzioni Lambda per Ruby con immagini di container

Esistono tre modi per creare un'immagine di container per una funzione Lambda in Ruby:

- Usare un'immagine AWS di base per Ruby

  Le immagini di base AWS sono precaricate con un runtime in linguaggio, un client di interfaccia di runtime per gestire l'interazione tra Lambda e il codice della funzione e un emulatore di interfaccia di runtime per i test locali.
- Utilizzo di un'immagine di AWS base solo per il sistema operativo

  AWS Le immagini di base solo per il sistema operativo contengono una distribuzione Amazon Linux e l'emulatore di interfaccia di runtime. Queste immagini vengono comunemente utilizzate per creare immagini di container per linguaggi compilati, come Go e Rust, e per un linguaggio o una versione di linguaggio per cui Lambda non fornisce un'immagine di base, come Node.js 19. Puoi anche utilizzare immagini di base solo per il sistema operativo per implementare un runtime personalizzato. Per rendere l'immagine compatibile con Lambda, devi includere il client di interfaccia di runtime per Ruby nell'immagine.
- Utilizzo di un'immagine non di base AWS

  È possibile utilizzare un'immagine di base alternativa da un altro registro del container, come ad esempio Alpine Linux o Debian. Puoi anche utilizzare un'immagine personalizzata creata dalla tua organizzazione. Per rendere l'immagine compatibile con Lambda, devi includere il client di interfaccia di runtime per Ruby nell'immagine.

> **Tip**
>
> Per ridurre il tempo necessario all'attivazione delle funzioni del container Lambda, consulta Utilizzo di compilazioni a più fasi nella documentazione Docker. Per creare immagini di container efficienti, segui le best practice per scrivere file Docker.

Questa pagina spiega come creare, testare e implementare le immagini di container per Lambda.

Argomenti

- AWS immagini di base per Ruby
- Usare un'immagine AWS di base per Ruby

- Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

AWS immagini di base per Ruby

AWS fornisce le seguenti immagini di base per Ruby:

Tag	Runtime	Sistema operativo	Dockerfile	Raggiunta obsolescenza
3.4	Ruby 3.4	Amazon Linux 2023	Dockerfile per Ruby 3.4 e versioni successive GitHub	31 marzo 2028
3.3	Ruby 3.3	Amazon Linux 2023	Dockerfile per Ruby 3.3 su GitHub	31 marzo 2027
3.2	Ruby 3.2	Amazon Linux 2	Dockerfile per Ruby 3.2 su GitHub	31 marzo 2026

Archivio Amazon ECR: gallery.ecr. aws/lambda/ruby

Usare un'immagine AWS di base per Ruby

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS CLI versione 2
- Docker (versione minima 25.0.0)
- Il plugin buildx di Docker.
- Ruby

Creazione di un'immagine da un'immagine di base

Creazione di un'immagine di container per Ruby

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir example
```

```
cd example
```

2. Crea un nuovo file denominato `Gemfile`. Qui è possibile elencare i RubyGems pacchetti richiesti dall'applicazione. AWS SDK per Ruby È disponibile da RubyGems. Dovresti scegliere AWS service gem specifici da installare. Ad esempio, per usare la gemma Ruby per Lambda, il tuo Gemfile dovrebbe avere questo aspetto:

```
source 'https://rubygems.org'

gem 'aws-sdk-lambda'
```

   In alternativa, la gemma aws-sdk contiene tutte le gem di servizio disponibili. AWS Questa gemma è molto grande. Ti consigliamo di utilizzarlo solo se dipendi da molti servizi. AWS

3. Installa le dipendenze specificate nel Gemfile utilizzando l'installazione in bundle.

```
bundle install
```

4. Crea un nuovo file denominato `lambda_function.rb`. A fini di test, puoi utilizzare il codice della funzione di esempio seguente o sostituirlo con il tuo codice personalizzato.

   Example Funzione Ruby

```
module LambdaFunction
  class Handler
    def self.process(event:,context:)
      "Hello from Lambda!"
    end
  end
end
```

5. Crea un nuovo Dockerfile. Il Dockerfile di esempio seguente utilizza l'immagine di base AWS. Il Dockerfile utilizza la seguente configurazione:

   - Imposta la proprietà `FROM` sull'URI dell'immagine di base.
   - Utilizza il comando COPY per copiare il codice della funzione e le dipendenze di runtime in `{LAMBDA_TASK_ROOT}`, una variabile d'ambiente definita da Lambda.
   - Imposta l'argomento `CMD` specificando il gestore della funzione Lambda.

Nota che l'esempio Dockerfile non include un'istruzione USER. Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione USER.

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. Crea l'immagine Docker con il comando docker build. L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il tag `test`. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'`--provenance=false`opzione.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Se intendi creare una funzione Lambda utilizzando l'architettura del set di ARM64 istruzioni, assicurati di modificare il comando per utilizzare invece l'`--platform linux/arm64`opzione.

(Facoltativo) Test dell'immagine in locale

1. Avvia l'immagine Docker con il comando docker run. In questo esempio, `docker-image` è il nome dell'immagine e `test` è il tag.

   ```
   docker run --platform linux/amd64 -p 9000:8080 docker-image:test
   ```

   Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Se hai creato l'immagine Docker per l'architettura del set di ARM64 istruzioni, assicurati di utilizzare l'`--platform linux/arm64`opzione invece di. `--platform linux/amd64`

2. Da una nuova finestra di terminale, invia un evento all'endpoint locale.

   Linux/macOS

   Su MacOS o Linux, esegui il comando seguente `curl`:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   In PowerShell, esegui il seguente `Invoke-WebRequest` comando:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Ottieni l'ID del container.

```
docker ps
```

4. Utilizza il comando docker kill per arrestare il container. In questo comando, sostituisci `3766c4ab331c` con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

Implementazione dell'immagine

Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Esegui il get-login-passwordcomando per autenticare la CLI Docker nel tuo registro Amazon ECR.
   - Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
   - Sostituiscilo `111122223333` con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il commando create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
    "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. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando docker tag per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:

   - `docker-image:test` è il nome e tag dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
   - Sostituisci l'`<ECRrepositoryUri>` con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando docker push per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crea un ruolo di esecuzione per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'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**
>
> È possibile creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta Autorizzazioni multiaccount Amazon ECR.

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nell'archivio Amazon ECR e quindi utilizzare il update-function-codecomando per distribuire l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per distribuire la nuova immagine nella stessa funzione Lambda, è necessario utilizzare update-function-codeil comando, anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

Se utilizzi un'immagine di base solo per il sistema operativo o un'immagine di base alternativa, devi includere il client dell'interfaccia di runtime nell'immagine. Il client dell'interfaccia di runtime estende l'API Runtime, che gestisce l'interazione tra Lambda e il codice della funzione.

Installa il client dell'interfaccia di runtime Lambda per Ruby utilizzando il gestore di RubyGems pacchetti.org:

```
gem install aws_lambda_ric
```

Puoi anche scaricare il client dell'interfaccia di runtime Ruby da. GitHub

L'esempio seguente mostra come creare un'immagine contenitore per Ruby utilizzando un'immagine non di base.AWS Il Dockerfile di esempio utilizza un'immagine di base Ruby ufficiale. Il Dockerfile include il client di interfaccia di runtime.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS CLI versione 2
- Docker (versione minima 25.0.0)
- Il plugin buildx di Docker.
- Ruby

Creazione di un'immagine da un'immagine di base alternativa

Creazione di un'immagine di container per Ruby utilizzando un'immagine di base alternativa

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir example
cd example
```

2. Crea un nuovo file denominato `Gemfile`. Qui è possibile elencare i RubyGems pacchetti richiesti dall'applicazione. AWS SDK per Ruby È disponibile da RubyGems. Dovresti scegliere AWS service gem specifici da installare. Ad esempio, per usare la gemma Ruby per Lambda, il tuo Gemfile dovrebbe avere questo aspetto:

```
source 'https://rubygems.org'

gem 'aws-sdk-lambda'
```

   In alternativa, la gemma aws-sdk contiene tutte le gem di servizio disponibili. AWS Questa gemma è molto grande. Ti consigliamo di utilizzarlo solo se dipendi da molti servizi. AWS

3. Installa le dipendenze specificate nel Gemfile utilizzando l'installazione in bundle.

```
bundle install
```

4. Crea un nuovo file denominato `lambda_function.rb`. A fini di test, puoi utilizzare il codice della funzione di esempio seguente o sostituirlo con il tuo codice personalizzato.

   Example Funzione Ruby

```
module LambdaFunction
  class Handler
    def self.process(event:,context:)
      "Hello from Lambda!"
    end
```

```
  end
end
```

5. Crea un nuovo Dockerfile. Il seguente Dockerfile utilizza un'immagine di base Ruby anziché un'immagine di base AWS. Il Dockerfile include il client di interfaccia di runtime per Ruby, che rende l'immagine compatibile con Lambda. In alternativa, puoi aggiungere il client di interfaccia di runtime al Gemfile dell'applicazione.

   - Imposta la proprietà `FROM` sull'immagine di base Ruby.
   - Crea una directory per il codice della funzione e una variabile di ambiente che punti a quella directory. In questo esempio, la directory è `/var/task`, che rispecchia l'ambiente di esecuzione Lambda. Tuttavia, puoi scegliere qualsiasi directory per il codice della funzione perché il Dockerfile non utilizza un'immagine di AWS base.
   - Imposta l'`ENTRYPOINT` sul modulo su cui desideri che il container Docker venga eseguito all'avvio. In questo caso, il modulo è il client di interfaccia di runtime.
   - Imposta l'argomento `CMD` specificando il gestore della funzione Lambda.

   Nota che l'esempio Dockerfile non include un'istruzione USER. Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

   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. Crea l'immagine Docker con il comando docker build. L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il tag `test`. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'`--provenance=false`opzione.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Se intendi creare una funzione Lambda utilizzando l'architettura del set di ARM64 istruzioni, assicurati di modificare il comando per utilizzare invece l'`--platform linux/arm64`opzione.

(Facoltativo) Test dell'immagine in locale

Usa il simulatore dell'interfaccia di runtime per testare l'immagine in locale. Puoi creare l'emulatore nella tua immagine o seguire la procedura riportata e installarlo sul tuo computer locale.

Installazione ed esecuzione dell'emulatore di interfaccia di runtime sul computer locale

1. Dalla directory del progetto, esegui il comando seguente per scaricare l'emulatore di interfaccia di runtime (architettura x86-64) GitHub e installarlo sul computer locale.

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
```

Per installare l'emulatore arm64, sostituisci l'URL del GitHub repository nel comando precedente con il seguente:

```
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
```

Per installare l'emulatore arm64, sostituisci `$downloadLink` con quanto segue:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Avvia l'immagine Docker con il comando docker run. Tenere presente quanto segue:

   - `docker-image` è il nome dell'immagine e `test` è il tag.
   - `aws_lambda_ric lambda_function.LambdaFunction::Handler.process` è l'`ENTRYPOINT` seguito dal `CMD` del Dockerfile.

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
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Se hai creato l'immagine Docker per l'architettura del set di ARM64 istruzioni, assicurati di utilizzare l'`--platform linux/`*`arm64`*opzione invece di. `--platform linux/`*`amd64`*

3. Pubblica un evento nell'endpoint locale.

Linux/macOS

Su MacOS o Linux, esegui il comando seguente `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

In PowerShell, esegui il seguente `Invoke-WebRequest` comando:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Ottieni l'ID del container.

```
docker ps
```

5. Utilizza il comando docker kill per arrestare il container. In questo comando, sostituisci `3766c4ab331c` con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

Implementazione dell'immagine

Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Esegui il get-login-passwordcomando per autenticare la CLI Docker nel tuo registro Amazon ECR.
   - Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
   - Sostituiscilo `111122223333` con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
    "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. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando docker tag per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:

- `docker-image:test` è il nome e tag dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
- Sostituisci l'`<ECRrepositoryUri>` con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando docker push per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crea un ruolo di esecuzione per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'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**
>
> È possibile creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta Autorizzazioni multiaccount Amazon ECR.

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nell'archivio Amazon ECR e quindi utilizzare il update-function-codecomando per distribuire l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per distribuire la nuova immagine nella stessa funzione Lambda, è necessario utilizzare update-function-codeil comando, anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Utilizzo dei livelli per le funzioni Lamba di Ruby

Usa i livelli Lambda per impacchettare codice e dipendenze che desideri riutilizzare in più funzioni. I livelli di solito contengono dipendenze dalla libreria, un runtime personalizzato o file di configurazione. La creazione di un livello prevede tre passaggi generali:

1. Crea un pacchetto per il contenuto del livello. Ciò significa creare un archivio di file con estensione .zip che contiene le dipendenze che desideri utilizzare nelle funzioni.
2. Crea il livello in Lambda.
3. Aggiungi il livello alle tue funzioni.

Argomenti

- Crea un pacchetto per il contenuto del livello
- Crea il livello in Lambda.
- Usare le gemme dei livelli in una funzione
- Aggiungi il livello alla tua funzione.
- app di esempio

Crea un pacchetto per il contenuto del livello

Per creare un livello, raggruppa i pacchetti in un archivio di file con estensione zip che soddisfi i seguenti requisiti:

- Create il livello utilizzando la stessa versione di Ruby che intendete utilizzare per la funzione Lambda. Ad esempio, se create il vostro layer per Ruby 3.4, usate il runtime Ruby 3.4 per la vostra funzione.
- Il file.zip del livello deve utilizzare una di queste strutture di directory:
  - `ruby/gems/`*`x.x.x`*(dove *`x.x.x`* è la vostra versione di Ruby, ad esempio) `3.4.0`
  - `ruby/lib`

  Per ulteriori informazioni, consulta Percorsi dei livelli per ciascun runtime Lambda.
- I pacchetti del livello devono essere compatibili con Linux. Le funzioni Lambda vengono eseguite su Amazon Linux.

Puoi creare livelli che contengono gemme Ruby di terze parti o moduli e classi Ruby personalizzati. Molte gemme Ruby popolari contengono estensioni native (codice C) che devono essere compilate per l'ambiente Lambda Linux.

Gemme di ruby puro

Le gemme Pure Ruby contengono solo codice Ruby e non richiedono la compilazione. Queste gemme sono più semplici da impacchettare e funzionano su piattaforme diverse.

Creazione di un livello utilizzando gemme di Ruby puri

1. Crea uno `Gemfile` per specificare le gemme Ruby pure che desideri includere nel tuo livello:

   Example Gemfile

   ```
   source 'https://rubygems.org'

   gem 'tzinfo'
   ```

2. Installa le gemme nella `vendor/bundle` directory usando Bundler:

   ```
   bundle config set --local path vendor/bundle
   bundle install
   ```

3. Copia le gemme installate nella struttura di directory richiesta da Lambda`ruby/gems/3.4.0`):

   ```
   mkdir -p ruby/gems/3.4.0
   cp -r vendor/bundle/ruby/3.4.0*/* ruby/gems/3.4.0/
   ```

4. Comprimi il contenuto dei livelli:

   Linux/macOS

   ```
   zip -r layer.zip ruby/
   ```

   PowerShell

   ```
   Compress-Archive -Path .\ruby -DestinationPath .\layer.zip
   ```

   L'aspetto della struttura di directory dovrebbe essere simile al seguente:

```
ruby/
### gems/
    ### 3.4.0/
        ### gems/
        #   ### concurrent-ruby-1.3.5/
        #   ### tzinfo-2.0.6/
        ### specifications/
        ### cache/
        ### build_info/
        ### (other bundler directories)
```

> **Note**
>
> È necessario richiedere ogni gemma singolarmente nel codice della funzione. Non è possibile utilizzare `bundler/setup` con `Bundler.require`. Per ulteriori informazioni, consulta Usare le gemme dei livelli in una funzione.

Gems con estensioni native

Molte gemme Ruby popolari contengono estensioni native (codice C) che devono essere compilate per la piattaforma di destinazione. Le gemme popolari con estensioni native includono nokogiri, pg, mysql2, sqlite3 e ffi. Queste gemme devono essere create in un ambiente Linux compatibile con il runtime Lambda.

Creazione di un livello utilizzando gemme con estensioni native

1. Creazione di un del `Gemfile`.

   Example Gemfile

   ```
   source 'https://rubygems.org'

   gem 'nokogiri'
   gem 'httparty'
   ```

2. Usa Docker per creare le gemme in un ambiente Linux compatibile con Lambda. Specificate un'immagine di AWS base nel vostro Dockerfile:

Example Dockerfile per Ruby 3.4

```
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. Costruisci l'immagine ed estrai il livello:

```
docker build -t ruby-layer-builder .
docker run --rm -v $(pwd):/output --entrypoint cp ruby-layer-builder layer.zip /
output/
```

Questo crea le gemme nell'ambiente Linux corretto e copia il `layer.zip` file nella directory locale. L'aspetto della struttura di directory dovrebbe essere simile al seguente:

```
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**
>
> È necessario richiedere ogni gem singolarmente nel codice della funzione. Non è possibile utilizzare `bundler/setup` con `Bundler.require`. Per ulteriori informazioni, consulta Usare le gemme dei livelli in una funzione.

Moduli di Ruby personalizzati

Creazione di un livello utilizzando il tuo codice

1. Crea la struttura di directory richiesta per il tuo livello:

```
mkdir -p ruby/lib
```

2. Crea i tuoi moduli Ruby nella `ruby/lib` directory. Il seguente modulo di esempio convalida gli ordini confermando che contengono le informazioni richieste.

   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. Comprimi il contenuto dei livelli:

   Linux/macOS

   ```
   zip -r layer.zip ruby/
   ```

   PowerShell

   ```
   Compress-Archive -Path .\ruby -DestinationPath .\layer.zip
   ```

   L'aspetto della struttura di directory dovrebbe essere simile al seguente:

   ```
   ruby/
   ### lib/
       ### order_validator.rb
   ```

4. Nella tua funzione, richiedi e usa i moduli. È necessario richiedere ogni gemma singolarmente nel codice della funzione. Non è possibile utilizzare `bundler/setup` con `Bundler.require`. Per ulteriori informazioni, consulta Usare le gemme dei livelli in una funzione. Esempio:

```
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
```

È possibile utilizzare i seguenti dati evento per invocare la funzione:

```
{
    "body": "{\"product_id\": \"ABC123\", \"quantity\": 2, \"priority\": \"express
\"}"
}
```

Risposta prevista:

```
{
  "statusCode": 200,
```

```
  "body": "{\"message\":\"Order validated successfully\",\"order\":{\"product_id\":
\"ABC123\",\"quantity\":2,\"shipping_priority\":\"express\"}}"
}
```

Crea il livello in Lambda.

È possibile creare la funzione Lambda mediante la AWS CLI o la console Lambda.

AWS CLI

Esegui il AWS CLI comando publish-layer-version per creare il livello Lambda:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip
 --compatible-runtimes ruby3.4
```

Il parametro compatible runtimes è facoltativo. Quando specificato, Lambda utilizza questo parametro per filtrare i livelli nella console Lambda.

Console

Creazione di un livello (console)

1. Apri la pagina Layers (Livelli) nella console Lambda.
2. Scegli Create layer (Crea livello).
3. Scegli Carica un file.zip, quindi carica l'archivio.zip che hai creato in precedenza.
4. (Facoltativo) Per i runtime compatibili, scegli il runtime di Ruby che corrisponde alla versione di Ruby che hai usato per creare il tuo layer.
5. Seleziona Create (Crea).

Usare le gemme dei livelli in una funzione

Nel codice della funzione, è necessario richiedere esplicitamente ogni gemma che si desidera utilizzare. I comandi Bundler come `bundler/setup` e non `Bundler.require` sono supportati. Ecco come usare correttamente le gemme di un livello in una funzione 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
```

Aggiungi il livello alla tua funzione.

AWS CLI

Per collegare il layer alla tua funzione, esegui il comando AWS CLIupdate-function-configuration. Usa l'attività ARN per il parametro `--layers`. L'ARN deve specificare la versione (ad esempio,`arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`). Per ulteriori informazioni, consulta Livelli e versioni di livelli.

```
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"
```

L'opzione cli-binary-format è necessaria se si utilizza la versione 2 della AWS CLI. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Console

Aggiunta di un livello a una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la funzione ().
3. Scorri verso il basso fino alla sezione Livelli, quindi scegli Aggiungi un livello.
4. In Scegli un livello, seleziona Livelli personalizzati, quindi scegli il tuo livello.

> **Note**
>
> Se non hai aggiunto un runtime compatibile quando hai creato il layer, il tuo layer non verrà elencato qui. È possibile specificare invece il livello ARN.

5. Scegli Aggiungi.

app di esempio

Per altri esempi di utilizzo dei layer Lambda, consulta l'applicazione di esempio layer-ruby nel repository GitHub della Developer AWS Lambda Guide. Questa applicazione include un livello che contiene la libreria tzinfo. Dopo aver creato un livello, puoi implementare e richiamare la funzione corrispondente per verificare che tutto funzioni.

Utilizzo dell'oggetto del contesto Lambda per recuperare le informazioni sulla funzione Ruby

Quando Lambda esegue la funzione, passa un oggetto Context al gestore. Questo oggetto fornisce i metodi e le proprietà che forniscono le informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

Metodi del contesto

- `get_remaining_time_in_millis`: restituisce il numero di millisecondi rimasti prima del timeout dell'esecuzione.

Proprietà del contesto

- `function_name`: il nome della funzione Lambda.
- `function_version`: la versione della funzione.
- `invoked_function_arn`: l'Amazon Resource Name (ARN) utilizzato per richiamare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `memory_limit_in_mb`: la quantità di memoria allocata per la funzione.
- `aws_request_id`: l'identificatore della richiesta di invocazione.
- `log_group_name`: il gruppo di log per la funzione.
- `log_stream_name`: il flusso di log per l'istanza della funzione.
- `deadline_ms`: la data del timeout dell'esecuzione in millisecondi Unix.
- `identity`: (app per dispositivi mobili) Informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
- `client_context`: (app per dispositivi mobili) Contesto client fornito a Lambda dall'applicazione client.

Registrare e monitorare le funzioni Lambda con Ruby

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della funzione. L'ambiente del runtime Lambda invia i dettagli su ogni richiamo al flusso di log e inoltra i log e l'output del codice della funzione. Per ulteriori informazioni, consulta Lambda invia automaticamente i log delle funzioni a CloudWatch Logs..

Questa pagina descrive come produrre un output di registro dal codice della funzione Lambda e accedere ai log utilizzando AWS Command Line Interface la console Lambda o la console. CloudWatch

Sections

- Creazione di una funzione che restituisce i registri
- Visualizzazione dei log nella console Lambda
- Visualizzazione dei log nella console CloudWatch
- Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI
- Eliminazione dei log
- Utilizzo della libreria dei logger Ruby

Creazione di una funzione che restituisce i registri

Per i log di output del codice della funzione, puoi usare le istruzioni `puts` o qualsiasi libreria di registrazione che scriva in `stdout` o `stderr`. L'esempio seguente registra i valori delle variabili di ambiente e l'oggetto evento.

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 Formato dei 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
```

Il runtime di Ruby registra `START`, `END` e `REPORT` per ogni chiamata. La riga del report fornisce i seguenti dettagli.

Campi dati della riga REPORT

- RequestId— L'ID univoco della richiesta per la chiamata.
- Durata – La quantità di tempo che il metodo del gestore della funzione impiega durante l'elaborazione dell'evento.
- Durata fatturata – La quantità di tempo fatturata per la chiamata.
- Dimensioni memoria – La quantità di memoria allocata per la funzione.
- Quantità max utilizzata – La quantità di memoria utilizzata dalla funzione. Quando le invocazioni condividono un ambiente di esecuzione, Lambda riporta la memoria massima utilizzata in tutte le invocazioni. Questo comportamento potrebbe comportare un valore riportato superiore al previsto.
- Durata Init – Per la prima richiesta servita, la quantità di tempo impiegato dal runtime per caricare la funzione ed eseguire il codice al di fuori del metodo del gestore.
- XRAY TraceId — Per le richieste tracciate, l'ID di traccia.AWS X-Ray
- SegmentId— Per le richieste tracciate, l'ID del segmento X-Ray.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

Per ottenere log più dettagliati, utilizza la the section called “Utilizzo della libreria dei logger Ruby”.

Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor del codice incorporato, troverai i log nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l'output del log nella sezione Dettagli.

Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (***your-function-name***/aws/lambda/).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Viene visualizzato un flusso di log quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire le chiamate simultanee. Per trovare i log per una chiamata specifica, consigliamo di strumentare la funzione con. AWS X-Ray X-Ray registra i dettagli sulla richiesta e il flusso di log nella traccia.

Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della AWS CLI versione 2.

È possibile utilizzare AWS CLI per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'`base64` utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `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
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
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
```

L'utilità `base64` è disponibile su Linux, macOS e Ubuntu su Windows. Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

Example Script get-logs.sh

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza `sed` per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.

L'cli-binary-formatopzione è obbligatoria se utilizzi la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 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 (solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il seguente comando per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
    "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"
}
```

Eliminazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, eliminare il gruppo di log o configurare un periodo di conservazione trascorso il quale i log vengono eliminati automaticamente.

Utilizzo della libreria dei logger Ruby

La libreria di registrazione Ruby restituisce log semplificati e facilmente leggibili. Utilizza l'utilità di registrazione per generare informazioni dettagliate, messaggi e codici di errore relativi alla tua funzione.

```
# 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
```

L'output di `logger` include il livello del log, il timestamp e l'ID della richiesta.

```
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
```

Strumentazione del codice Ruby in AWS Lambda

Lambda si integra con AWS X-Ray per consentire di tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzare X-Ray per tracciare una richiesta mentre attraversa le risorse nell'applicazione, dall'API di frontend allo storage e al database di backend. Semplicemente aggiungendo la libreria SDK X-Ray alla configurazione della compilazione, è possibile registrare errori e latenza per qualsiasi chiamata effettuata dalla funzione a un servizio AWS.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico dei servizi X-Ray mostra informazioni sull'applicazione e tutti i suoi componenti. Il seguente esempio mostra un'applicazione con due funzioni. La funzione principale elabora gli eventi e talvolta restituisce errori. La seconda funzione elabora gli errori che appaiono nel gruppo di log della prima funzione e utilizza l'SDK AWS per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon CloudWatch Logs.



Per attivare il tracciamento attivo sulla funzione Lambda con la console, attenersi alla seguente procedura:

Per attivare il tracciamento attivo

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione) e quindi Monitoring and operations tools (Strumenti di monitoraggio e operazioni).
4. Nel riquadro Strumenti di monitoraggio aggiuntivi scegli Modifica.

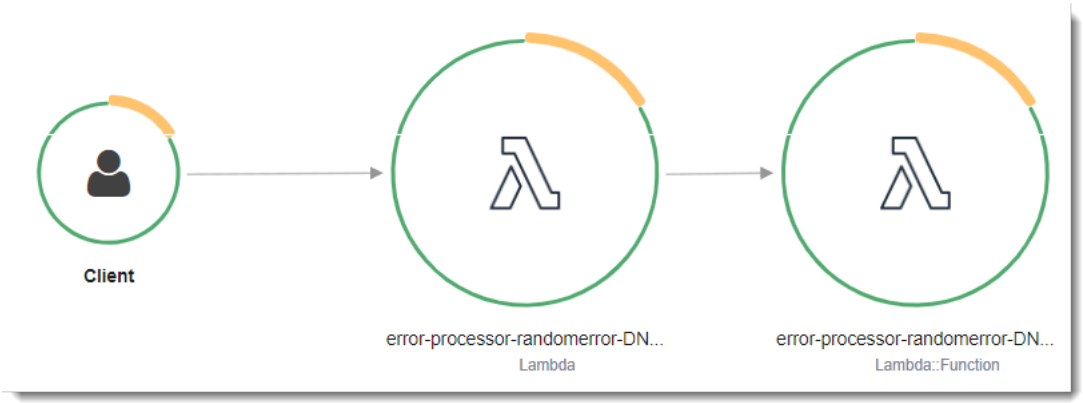
5. In CloudWatch Application Signals AWS X-Ray e, scegli Enable for Lambda service trace.
6. Selezionare Salva.

> **Prezzi**
>
> È possibile utilizzare il tracciamento X-Ray gratuitamente ogni mese fino a un certo limite come parte del piano gratuito di AWS. Oltre la soglia, X-Ray addebita lo storage di traccia e il recupero. Per ulteriori informazioni, consultare Prezzi di AWS X-Ray.

La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al ruolo di esecuzione della funzione. In caso contrario, aggiungere la policy AWSXRayDaemonWriteAccess al ruolo di esecuzione.

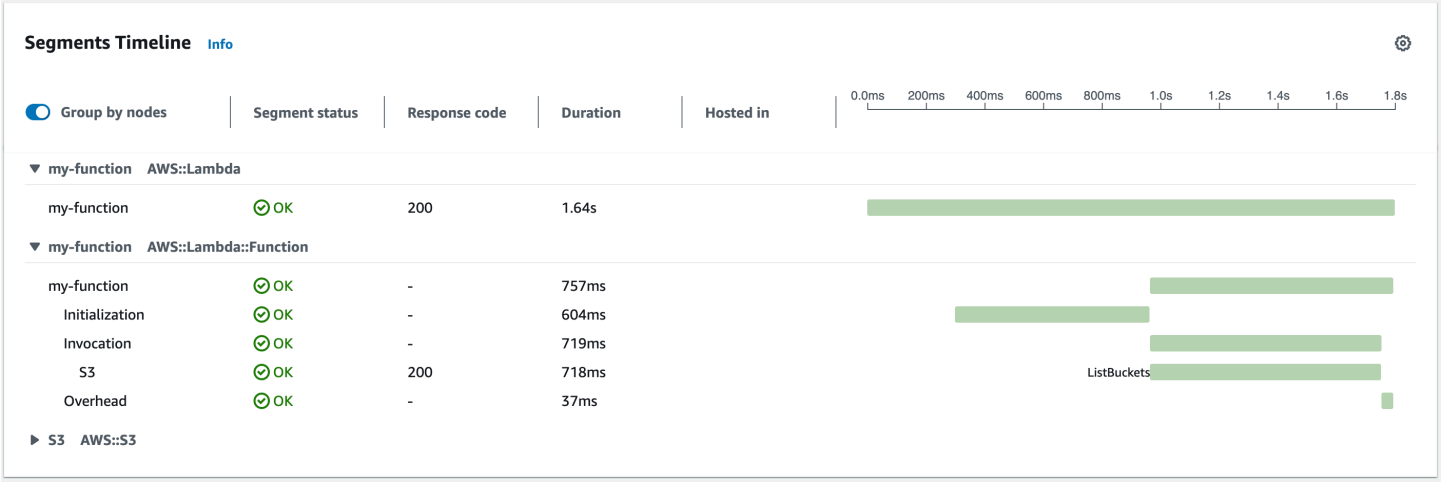
X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non è possibile configurare la frequenza di campionamento di X-Ray per le funzioni.

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda. L'esempio seguente mostra una traccia con questi 2 segmenti. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine

`AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.



Questo esempio espande il segmento `AWS::Lambda::Function` per visualizzare i relativi tre sottosegmenti.

> **Note**
>
> AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. La traccia di esempio mostrata qui illustra il segmento di funzione vecchio stile. Le differenze tra i segmenti vecchio e nuovo stile sono descritte nei paragrafi seguenti.
> Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni in tutte le Regioni AWS, ad eccezione delle regioni Cina e GovCloud, passeranno all'utilizzo dei messaggi di log e dei segmenti di traccia di nuovo formato.

Il segmento di funzioni vecchio stile contiene i seguenti sottosegmenti:

- Inizializzazione – Rappresenta il tempo trascorso a caricare la funzione e ad eseguire il codice di inizializzazione. Questo sottosegmento viene visualizzato solo per il primo evento che viene elaborato da ogni istanza della funzione.
- Chiamata: rappresenta il tempo impiegato per eseguire il codice del gestore.
- Overhead: rappresenta il tempo impiegato dal runtime Lambda per prepararsi a gestire l'evento successivo.

Il segmento di funzione di nuovo stile non contiene un sottosegmento `Invocation`. I sottosegmenti dei clienti sono invece collegati direttamente al segmento di funzioni. Per ulteriori informazioni sulla struttura dei segmenti di funzioni vecchio e nuovo stile, consulta the section called “Informazioni sui monitoraggi di X-Ray”.

È possibile strumentare il codice del gestore per registrare i metadati e tracciare le chiamate a valle. Per registrare dettagli sulle chiamate effettuate dal gestore ad altre risorse e servizi, utilizzare l'SDK for Ruby X-Ray. Per ottenere l'SDK, aggiungere il pacchetto `aws-xray-sdk` alle dipendenze dell'applicazione.

Example blank-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'
```

Per strumentare i client AWS SDK, richiedere il modulo `aws-xray-sdk/lambda` dopo aver creato un client nel codice di inizializzazione.

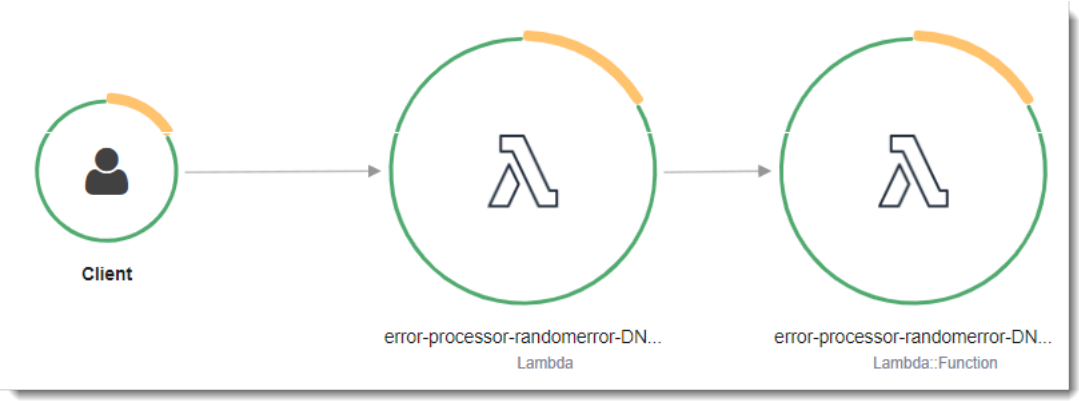
Example blank-ruby/function/lambda_function.rb – Tracciamento di un client SDK AWS

```
# lambda_function.rb
require 'logger'
require 'json'
require 'aws-sdk-lambda'
$client = Aws::Lambda::Client.new()
$client.get_account_settings()

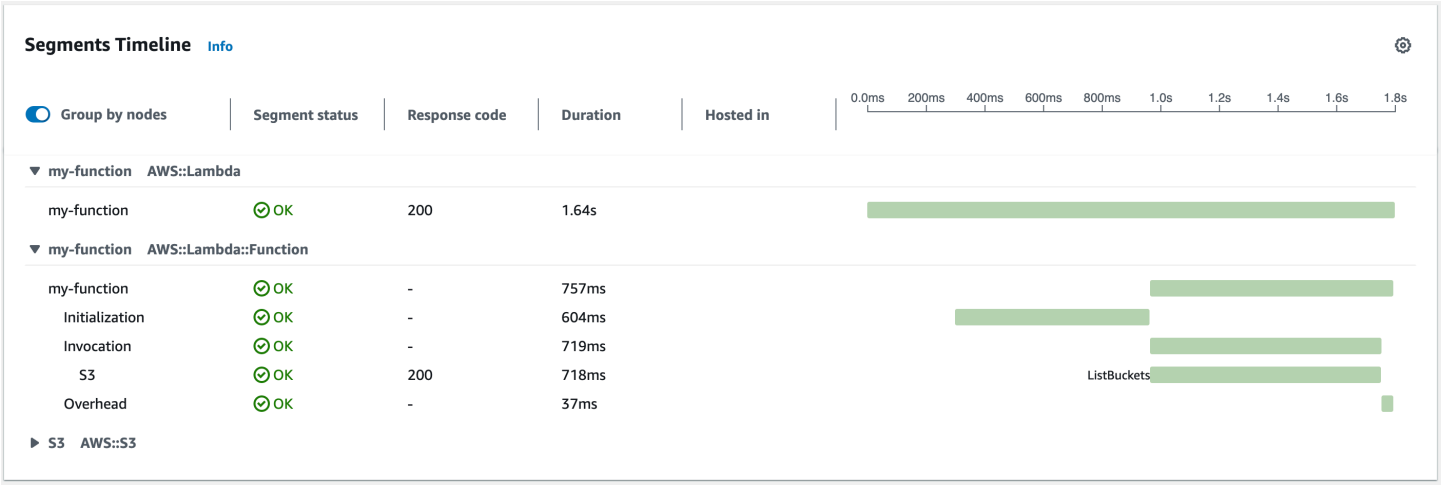
require 'aws-xray-sdk/lambda'

def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
  ...
```

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda. L'esempio seguente mostra una traccia con questi 2 segmenti. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.



Questo esempio espande il segmento `AWS::Lambda::Function` per visualizzare i relativi tre sottosegmenti.

> **Note**
>
> AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. La traccia di esempio mostrata qui illustra il segmento di funzione vecchio stile. Le differenze tra i segmenti vecchio e nuovo stile sono descritte nei paragrafi seguenti.

> Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni in tutte le Regioni AWS, ad eccezione delle regioni Cina e GovCloud, passeranno all'utilizzo dei messaggi di log e dei segmenti di traccia di nuovo formato.

Il segmento di funzioni vecchio stile contiene i seguenti sottosegmenti:

- Inizializzazione – Rappresenta il tempo trascorso a caricare la funzione e ad eseguire il codice di inizializzazione. Questo sottosegmento viene visualizzato solo per il primo evento che viene elaborato da ogni istanza della funzione.
- Chiamata: rappresenta il tempo impiegato per eseguire il codice del gestore.
- Overhead: rappresenta il tempo impiegato dal runtime Lambda per prepararsi a gestire l'evento successivo.

Il segmento di funzione di nuovo stile non contiene un sottosegmento `Invocation`. I sottosegmenti dei clienti sono invece collegati direttamente al segmento di funzioni. Per ulteriori informazioni sulla struttura dei segmenti di funzioni vecchio e nuovo stile, consulta the section called “Informazioni sui monitoraggi di X-Ray”.

È inoltre possibile strumentare i client HTTP, registrare query SQL e creare segmenti secondari personalizzati con annotazioni e metadati. Per ulteriori informazioni, consulta l'SDK for Ruby X-Ray nella Guida per gli sviluppatori di AWS X-Ray.

Sezioni

- Abilitazione del tracciamento attivo con l'API Lambda
- Abilitazione del tracciamento attivo con CloudFormation
- Memorizzazione delle dipendenze di runtime in un layer

Abilitazione del tracciamento attivo con l'API Lambda

Per gestire la configurazione del tracciamento con AWS CLI CLI o SDK AWS, utilizza le seguenti operazioni API.

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Il seguente comando AWS CLI di esempio consente il tracciamento attivo su una funzione denominata my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

La modalità di tracciamento fa parte della configurazione specifica della versione quando si pubblica una versione della funzione. Non è possibile modificare la modalità di tracciamento in una versione pubblicata.

Abilitazione del tracciamento attivo con CloudFormation

Per attivare il tracciamento su una risorsa `AWS::Lambda::Function` in un modello CloudFormation, utilizzare la proprietà `TracingConfig`.

Example function-inline.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Per una risorsa AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, utilizzare la proprietà `Tracing`.

Example template.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Memorizzazione delle dipendenze di runtime in un layer

Se si utilizza l'SDK X-Ray per strumentare i client SDK AWS del codice della funzione, il pacchetto di implementazione può diventare piuttosto grande. Per evitare di caricare dipendenze di runtime ogni volta che si aggiorna il codice della funzione, includere l'SDK X-Ray in un livello Lambda.

L'esempio seguente mostra una risorsa `AWS::Serverless::LayerVersion` che memorizza l'SDK for Ruby X-Ray.

Example template.yml – Livello delle dipendenze

```
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
```

Con questa configurazione, si aggiorna il livello della libreria solo se si modificano le dipendenze di runtime. Poiché il pacchetto di implementazione della funzione contiene solo il codice, questo può contribuire a ridurre i tempi di caricamento.

La creazione di un layer per le dipendenze richiede modifiche alla compilazione per generare l'archivio dei layer prima della distribuzione. Per un esempio funzionante, vedere l'applicazione di esempio blank-ruby .

Compilazione di funzioni Lambda con Java

Puoi eseguire il codice Java in AWS Lambda. Lambda fornisce Runtime per Java che eseguono il tuo codice per elaborare gli eventi. Il codice viene eseguito in un ambiente Amazon Linux che include AWS le credenziali di un ruolo AWS Identity and Access Management (IAM) che gestisci.

Lambda supporta i seguenti runtime di Java.

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Java 2.5	`java25`	Amazon Linux 2023	30 giugno 2029	31 luglio 2029	31 agosto 2029
Java 21	`java21`	Amazon Linux 2023	30 giugno 2029	31 luglio 2029	31 agosto 2029
Java 17	`java17`	Amazon Linux 2	30 giugno 2027	31 luglio 2027	31 agosto 2027
Java 11	`java11`	Amazon Linux 2	30 giugno 2027	31 luglio 2027	31 agosto 2027
Java 8	`java8.al2`	Amazon Linux 2	30 giugno 2027	31 luglio 2027	31 agosto 2027

AWS fornisce le seguenti librerie per le funzioni Java. Queste librerie sono disponibili tramite Maven Central Repository.

- com.amazonaws: aws-lambda-java-core (obbligatorio) — Definisce le interfacce del metodo del gestore e l'oggetto di contesto che il runtime passa al gestore. Se definisci i propri tipi di input, questa è l'unica libreria necessaria.
- com.amazonaws: aws-lambda-java-events — Tipi di input per eventi provenienti da servizi che richiamano funzioni Lambda.
- com.amazonaws: aws-lambda-java-log 4j2 — Una libreria di appender per Apache Log4j 2 che puoi utilizzare per aggiungere l'ID della richiesta per la chiamata corrente ai log delle funzioni.

- AWS SDK for Java 2.0 — L'SDK AWS ufficiale per il linguaggio di programmazione Java.

Aggiungi queste librerie alla definizione di build come segue.

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'
}
```

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>
  </dependencies>
```

Important

Non utilizzare componenti privati dell'API JDK, come campi, metodi o classi di tipo privato. I componenti dell'API non pubblici possono cambiare o essere rimossi in qualsiasi aggiornamento e causare l'interruzione dell'applicazione.

Per creare una funzione Java

1. Aprire la console Lambda.
2. Scegli Crea funzione.
3. Configura le impostazioni seguenti:
   - Nome della funzione: inserisci il nome della funzione.
   - Runtime: scegli Java 25.
4. Scegli Crea funzione.

La console crea una funzione Lambda con una classe del gestore denominata `Hello`. Poiché Java è un linguaggio compilato, non è possibile visualizzare o modificare il codice sorgente nella console Lambda, ma è possibile modificarne la configurazione, invocarla e configurare i trigger.

> **Note**
>
> Per iniziare a sviluppare applicazioni nel tuo ambiente locale, distribuisci una delle applicazioni di esempio disponibili nell' GitHub archivio di questa guida.

La classe `Hello` ha una funzione denominata `handleRequest` che richiede un oggetto evento e un oggetto contesto. Questa è la funzione del gestore chiamata da Lambda quando la funzione viene richiamata. Il runtime della funzione Lambda riceve gli eventi di chiamata da e li passa al gestore. Nella configurazione della funzione il valore del gestore è `example.Hello::handleRequest`.

Per aggiornare il codice di funzione, crea un pacchetto di distribuzione costituito da un archivio .zip contenente il codice di funzione. Man mano che lo sviluppo della tua funzione procede vorrai memorizzare il tuo codice di funzione nel controllo del codice sorgente, aggiungere le librerie e automatizzare le distribuzioni. Inizia creando un pacchetto di distribuzione e aggiornando il codice dalla riga di comando.

Il runtime della funzione passa un oggetto contesto al gestore, oltre all'evento di chiamata. L'oggetto contesto contiene ulteriori informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. Altre informazioni sono disponibili con le variabili di ambiente.

La funzione Lambda include un gruppo di CloudWatch log Logs. Il runtime della funzione invia i dettagli su ogni chiamata a Logs. CloudWatch Si trasmette qualsiasi log che la tua funzione emette

durante la chiamata. Se la funzione restituisce un errore, Lambda formatta l'errore e lo restituisce al chiamante.

Argomenti

- Definire l'handler della funzione Lambda in Java
- Distribuisci funzioni Lambda per Java con archivi di file .zip o JAR
- Distribuisci funzioni Java Lambda con immagini di container
- Utilizzo dei livelli per le funzioni Lambda in Java
- Personalizzare la serializzazione per le funzioni Lambda Java
- Personalizzare il comportamento di avvio del runtime Java per le funzioni Lambda
- Utilizzo dell'oggetto di contesto Lambda per recuperare le informazioni sulla funzione Java
- Registrare e monitorare funzioni Lambda in Java
- Strumentazione del codice Java in AWS Lambda
- Esempi di applicazioni Java per AWS Lambda

Definire l'handler della funzione Lambda in Java

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Questa pagina descrive come lavorare con i gestori di funzioni Lambda in Go, inclusa la configurazione del progetto, le convenzioni di denominazione e le migliori pratiche. Questa pagina include anche un esempio di funzione Go Lambda che raccoglie informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon Simple Storage Service (S3). Per informazioni su come distribuire una funzione dopo averla scritta, consulta o. the section called “Implementazione di archivi di file .zip” the section called “Implementazione di immagini di container”

Sezioni

- Configurazione di Go Handler
- Esempio di codice della funzione Lambda
- Definizioni di classe valide per i gestori Java
- Convenzioni di denominazione dei gestori
- Definizione e accesso all'oggetto evento di input
- Accesso e utilizzo dell'oggetto contestuale Lambda
- Utilizzo della AWS versione v2 nell'handler
- Accesso alle variabili d'ambiente
- Utilizzo dello stato globale
- Best practice di codice per funzioni Lambda in Java

Configurazione di Go Handler

Quando si lavora con le funzioni Lambda in Java, il processo prevede la scrittura del codice, la compilazione e la distribuzione degli artefatti compilati in Lambda. È possibile inizializzare un progetto Java Lambda in vari modi. Ad esempio, puoi utilizzare strumenti come le funzioni Maven Archetype for Lambda, AWS SAM il comando CLI sam init o persino una configurazione standard di progetto Java nel tuo IDE preferito, come IntelliJ IDEA o Visual Studio Code. In alternativa, puoi creare manualmente la struttura di file richiesta.

Un tipico progetto di funzione Java Lambda segue questa struttura generale:

```
/project-root
    # src
        # main
            # java
                # example
                    # OrderHandler.java (contains main handler)
                    # <other_supporting_classes>
     # build.gradle OR pom.xml
```

Puoi usare Maven o Gradle per creare il tuo progetto e gestire le dipendenze.

La logica principale del gestore per la funzione risiede in un file Java all'interno della directory. `src/main/java/example` In questo esempio viene assegnato un nome a questo file`OrderHandler.java`. Oltre a questo file, è possibile includere classi Java aggiuntive in base alle esigenze. Quando distribuisci la tua funzione in Lambda, assicurati di specificare la classe Java che contiene il metodo del gestore principale che Lambda deve invocare durante una chiamata.

Esempio di codice della funzione Lambda

Il seguente esempio di codice della funzione Go Lambda raccoglie le informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon S3.

Example **`OrderHandler.java`**Funzione 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);
        }
    }
}
```

Questo file `OrderHandler.java` contiene le sezioni seguenti:

- `package example`: In Java, può essere qualsiasi cosa, ma deve corrispondere alla struttura di directory del progetto. Qui, lo usiamo `package example` perché la struttura delle cartelle è`src/main/java/example`.
- `import`istruzioni: consente di importare le classi Java richieste dalla funzione Lambda.
- `public class OrderHandler ...`: definisce la classe Java e deve essere una definizione di classe valida.
- `private static final S3Client S3_CLIENT ...`: Questo inizializza un client S3 al di fuori di qualsiasi metodo della classe. Questo fa sì che Lambda esegua questo codice durante la fase di inizializzazione.
- block: definisce la forma dell'evento di input previsto in questa struttura Go.
- `public String handleRequest(Order event, Context context)`: questo è il metodo dell'handler principale, che contiene la logica principale dell'applicazione.
- `private void uploadReceiptToS3(...) {}`: questo è un metodo helper a cui fa riferimento il metodo dell'handler principale `handleRequest`.

File build.gradle e pom.xml di esempio

Il seguente `pom.xml` file `build.gradle` o accompagna questa funzione.

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>
```

Affinché questa funzione funzioni correttamente, il suo ruolo di esecuzione deve consentire l'`s3:PutObject`azione. Inoltre, assicuratevi di definire la variabile di `RECEIPT_BUCKET` ambiente. Dopo una chiamata riuscita, il bucket Amazon S3 dovrebbe contenere un file di ricevuta.

> **Note**
>
> Questa funzione può richiedere impostazioni di configurazione aggiuntive per funzionare correttamente senza timeout. Si consiglia di configurare 256 MB di memoria e un timeout di 10 secondi. La prima chiamata potrebbe richiedere più tempo a causa di un avvio a freddo. Le chiamate successive dovrebbero essere eseguite molto più velocemente grazie al riutilizzo dell'ambiente di esecuzione.

Definizioni di classe valide per i gestori Java

La libreria aws-lambda-java-core definisce due interfacce per i metodi del gestore. Utilizzare le interfacce fornite per semplificare la configurazione del gestore e convalidare la firma del metodo del gestore in fase di compilazione.

- com.amazonaws.services.lambda.runtime.RequestHandler
- com.amazonaws.services.lambda.runtime.RequestStreamHandler

L'interfaccia `RequestHandler` è un tipo generico che accetta due parametri: il tipo di input e il tipo di output. Entrambi i tipi devono essere oggetti. In questo esempio, la nostra `OrderHandler` classe

implementa. `RequestHandler<OrderHandler.Order, String>` Il tipo di input è il `Order` record che definiamo all'interno della classe e il tipo di output è`String`.

```
public class OrderHandler implements RequestHandler<OrderHandler.Order, String> {
    ...
}
```

Quando si utilizza questa interfaccia, il runtime Java deserializza l'evento in un oggetto con il tipo di input e serializza l'output in testo. Utilizzare questa interfaccia quando la serializzazione integrata funziona con i tipi di input e output.

Per utilizzare la propria serializzazione, implementare l'interfaccia `RequestStreamHandler`. Con questa interfaccia, Lambda passa al gestore un flusso di input e un flusso di output. Il gestore legge i byte dal flusso di input, scrive nel flusso di output e restituisce il valore void. Per un esempio di questo utilizzo del runtime Java 21, vedete HandlerStream.java.

Se lavori solo con tipi di base e generici (ad esempio `String` `IntegerList`, o`Map`) nella tua funzione Java, non è necessario implementare un'interfaccia. Ad esempio, se la funzione riceve un `Map<String, String>` input e restituisce un`String`, la definizione della classe e la firma del gestore potrebbero essere simili alle seguenti:

```
public class ExampleHandler {
    public String handleRequest(Map<String, String> input, Context context) {
        ...
    }
}
```

Inoltre, quando non implementate un'interfaccia, l'oggetto context è facoltativo. Ad esempio, la definizione della classe e la firma dell'handler potrebbero essere simili alle seguenti:

```
public class NoContextHandler {
   public String handleRequest(Map<String, String> input) {
        ...
   }
}
```

Convenzioni di denominazione dei gestori

Per le funzioni Lambda in Java, se si implementa l'`RequestStreamHandler`interfaccia `RequestHandler` or, è necessario denominare il metodo del gestore principale. `handleRequest`

Inoltre, includi il `@Override` tag sopra il `handleRequest` metodo. Quando distribuisci la tua funzione in Lambda, specifica il gestore principale nella configurazione della funzione nel seguente formato:

- <package>. <Class>— Ad esempio,`example.OrderHandler`.

Per le funzioni Lambda in Java che non implementano l'`RequestStreamHandler`interfaccia `RequestHandler` or, puoi usare qualsiasi nome per l'handler. Quando distribuisci la tua funzione in Lambda, specifica il gestore principale nella configurazione della funzione nel seguente formato:

- <package>. <Class>:: <handler_method_name>— Ad esempio,`example.Handler::mainHandler`.

Definizione e accesso all'oggetto evento di input

JSON è il formato di input più comune e standard per le funzioni Lambda. In questo esempio, la funzione prevede un input simile a quanto segue:

```
{
    "orderId": "12345",
    "amount": 199.99,
    "item": "Wireless Headphones"
}
```

Quando si utilizzano le funzioni Lambda in Go, è possibile definire la forma dell'evento di input previsto come struttura Go. In questo esempio, definiamo un record all'interno della `OrderHandler` classe per rappresentare un `Order` oggetto:

```
public record Order(String orderId, double amount, String item) {}
```

Questa struttura corrisponde alla forma di input prevista. Dopo aver definito il record, puoi scrivere una firma dell'handler che includa un input JSON conforme alla definizione del record. Il runtime Java deserializza automaticamente questo JSON in un oggetto Java. È quindi possibile accedere ai campi dell'oggetto. Ad esempio, `event.orderId` recupera il valore di `orderId` dall'input originale.

> **Note**
>
> I record Java sono una funzionalità dei runtime Java 17 e versioni successive. In tutti i runtime di Java, è possibile utilizzare una classe per rappresentare i dati degli eventi. In questi casi, puoi usare una libreria come jackson per deserializzare gli input JSON.

Altri tipi di eventi di input

Esistono molti eventi di input possibili per le funzioni Lambda in Java:

- `Integer`, `Long`, `Double`, e così via. – L' evento è un numero senza formattazione aggiuntiva, ad esempio ., `3.5`. Il runtime converte il valore in un oggetto del tipo specificato.
- `String` – l'evento è una stringa JSON, incluse le virgolette, ad esempio, `“My string”`. Il runtime converte il valore (senza virgolette) in un oggetto `String`.
- `List<Integer>`, `List<String>`, `List<Object>`, e così via. – L'evento è un array JSON. Il runtime lo deserializza in un oggetto del tipo o dell'interfaccia specificati.
- `InputStream`: l'evento è un tipo qualsiasi di JSON. Il runtime passa un flusso di byte del documento al gestore senza modifiche. Si deserializza l'output di input e scrittura in un flusso di output.
- Tipo di libreria: per gli eventi inviati dai servizi AWS, utilizzare i tipi nella libreria aws-lambda-java-events. Ad esempio, se la tua funzione Lambda viene richiamata da Amazon Simple Queue Service (SQS), usa l'oggetto come input. `SQSEvent`

Accesso e utilizzo dell'oggetto contestuale Lambda

L'oggetto contesto: contiene informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. In questo esempio, l'oggetto di contesto è di tipo `com.amazonaws.services.lambda.runtime.Context` ed è il secondo argomento della funzione di gestione principale.

```
public String handleRequest(Order event, Context context) {
    ...
}
```

Se la classe implementa l'interfaccia RequestHandler o RequestStreamHandler, l'oggetto context è un argomento obbligatorio. In caso contrario, l'oggetto oggetto oggetto è facoltativo. Per ulteriori

informazioni sulle firme dell'handler accettate, consulta the section called “Definizioni di classe valide per i gestori Java”.

Se si effettuano chiamate ad altri servizi utilizzando l'AWSSDK, l'oggetto context è necessario in alcune aree chiave. Ad esempio, per produrre log di funzioni per Amazon CloudWatch, puoi `context.getLogger()` utilizzare il metodo per `LambdaLogger` ottenere un oggetto per la registrazione. In questo esempio, possiamo usare il logger per registrare un messaggio di errore se l'elaborazione fallisce per qualsiasi motivo:

```
context.getLogger().log("Failed to process order: " + e.getMessage());
```

Oltre alle richieste SDK, puoi anche utilizzare l'oggetto context per il monitoraggio delle funzioni. Per ulteriori informazioni sulla copia di oggetti, consulta la sezione the section called “Contesto”.

Utilizzo della AWS versione v2 nell'handler

Spesso, utilizzerai le funzioni Lambda per interagire o aggiornare altre AWS risorse. Il modo più semplice per interfacciarsi con queste risorse è usare la AWS v2.

> **Note**
>
> La AWS (v1) è in modalità manutenzione e terminerà il supporto il 31 luglio 2025. In futuro, si consiglia di utilizzare solo la AWS v2.

Per aggiungere dipendenze SDK alla tua funzione, aggiungile nel tuo `build.gradle` for Gradle o `pom.xml` nel file per Maven. Ti consigliamo di aggiungere solo le librerie necessarie per la tua funzione. Nel codice di esempio precedente, abbiamo usato la libreria e la libreria. In Gradle, puoi aggiungere questa dipendenza aggiungendo la seguente riga nella sezione delle dipendenze del tuo: `build.gradle`

```
implementation 'software.amazon.awssdk:s3:2.28.29'
```

In Maven, aggiungi le seguenti righe nella sezione del tuo: `<dependencies>` `pom.xml`

```
    <dependency>
        <groupId>software.amazon.awssdk</groupId>
        <artifactId>s3</artifactId>
```

```
        <version>2.28.29</version>
    </dependency>
```

> **Note**
>
> Questa potrebbe non essere la versione più recente dell'SDK. Scegli la versione dell'SDK appropriata per la tua applicazione.

Quindi, importa le dipendenze direttamente nella tua classe Java:

```
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.S3Exception;
```

Il codice di esempio inizializza quindi un client Amazon S3 come segue:

```
private static final S3Client S3_CLIENT = S3Client.builder().build();
```

In questo esempio, abbiamo inizializzato il nostro client Amazon S3 nella funzione per evitare di doverlo inizializzare ogni volta che richiamiamo la nostra funzione. Dopo aver configurato e inizializzato il client SDK, puoi utilizzarlo per interagire con altri servizi. AWS Il codice di esempio richiama l'`PutObject`API Amazon S3 nel modo seguente:

```
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)));
```

Accesso alle variabili d'ambiente

Nel codice dell'handler, puoi fare riferimento a qualsiasi variabile di ambiente utilizzando il metodo `System.getenv()`. In questo esempio, facciamo riferimento alla variabile di `RECEIPT_BUCKET` ambiente definita utilizzando la seguente riga di codice:

```
String bucketName = System.getenv("RECEIPT_BUCKET");
if (bucketName == null || bucketName.isEmpty()) {
    throw new IllegalArgumentException("RECEIPT_BUCKET environment variable is not
 set");
}
```

Utilizzo dello stato globale

Lambda esegue il codice statico e il costruttore della classe durante la fase di inizializzazione prima di richiamare la funzione per la prima volta. Le risorse create durante l'inizializzazione restano in memoria tra le chiamate, in modo da evitare di doverle creare ogni volta che si richiama la funzione.

Nell'esempio seguente, il codice di inizializzazione del client non rientra nel metodo del gestore principale. Il runtime inizializza il client prima che la funzione gestisca il suo primo evento e il client rimane disponibile per il riutilizzo in tutte le chiamate.

Best practice di codice per funzioni Lambda in Java

Segui le linee guida riportate nell'elenco seguente per utilizzare le best practice di codifica durante la creazione delle funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. In questo modo è possibile creare una funzione di cui è più semplice eseguire l'unit test.
- Controllare le dipendenze nel pacchetto di distribuzione della funzione. L'ambiente di esecuzione AWS Lambda contiene diverse librerie. Per abilitare il set di caratteristiche e aggiornamenti della sicurezza più recenti, Lambda aggiorna periodicamente tali librerie. Tali aggiornamenti possono introdurre lievi modifiche al comportamento della funzione Lambda. Per mantenere il controllo completo delle dipendenze utilizzate dalla funzione, inserire tutte le dipendenze nel pacchetto di implementazione.
- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del contesto di esecuzione. Preferire ad esempio l'utilizzo di framework di inserimento di dipendenze Java, come Dagger o Guice, rispetto a framework più complessi come Spring Framework.
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. In questo modo viene ridotta la quantità di tempo necessaria per il download del pacchetto e per la relativa decompressione prima dell'invocazione. Per le funzioni create in Java, evitare di caricare l'intera libreria dell'SDK AWS come parte del pacchetto di implementazione.

Utilizzare invece in modo selettivo i moduli che prelevano i componenti dell'SDK necessari (ad esempio i moduli SDK Dynamo DB e Amazon S3 e le librerie di base Lambda).

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedere Riutilizzo delle connessioni con Keep-Alive in Node.js.

Utilizzare le variabili di ambiente per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su `0` per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare API non documentate e non pubbliche nel codice della funzione Lambda. Per i tempi di esecuzione gestiti AWS Lambda, Lambda applica periodicamente aggiornamenti di sicurezza e funzionalità alle API interne di Lambda. Questi aggiornamenti API interni potrebbero essere incompatibili con le versioni precedenti, causando conseguenze indesiderate come errori di chiamata se la funzione ha una dipendenza su queste API non pubbliche. Consulta il riferimento all'API per un elenco di API disponibili pubblicamente.

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente

gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda?.

- Evita di usare la cache DNS Java. Le funzioni Lambda memorizzano già nella cache le risposte DNS. Se viene utilizzata un'altra cache DNS, è possibile che si verifichino dei timeout di connessione.

  La classe `java.util.logging.Logger` può abilitare indirettamente la cache DNS JVM. Per sovrascrivere le impostazioni predefinite, imposta networkaddress.cache.ttl su 0 prima dell'inizializzazione di `logger`. Esempio:

  ```
  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);
  }
  ```

- Ridurre il tempo necessario a Lambda per decomprimere i pacchetti di distribuzione creati in Java inserendo i file `.jar` della dipendenza in una directory /lib separata. Questo metodo è più rapido rispetto all'inserimento di tutto il codice della funzione in un unico file .jar con un elevato numero di file `.class`. Per istruzioni, consulta Distribuisci funzioni Lambda per Java con archivi di file .zip o JAR.

Distribuisci funzioni Lambda per Java con archivi di file .zip o JAR

Il codice della AWS Lambda funzione è costituito da script o programmi compilati e dalle relative dipendenze. Utilizza un pacchetto di implementazione per distribuire il codice della funzione a Lambda. Lambda supporta due tipi di pacchetti di implementazione: immagini di container e archivi di file .zip.

Questa pagina descrive come creare il pacchetto di distribuzione come file.zip o file Jar, quindi utilizzare il pacchetto di distribuzione per distribuire il codice della funzione su (). AWS Lambda AWS Command Line Interface AWS CLI

> **Important**
>
> Java 25 ha introdotto il supporto per le cache Ahead-of-Time (AOT). Ti consigliamo vivamente di non utilizzare le cache AOT quando distribuisci le tue funzioni come archivi di file.zip o JAR, poiché le cache possono causare un comportamento imprevisto quando Lambda aggiorna il runtime gestito. Per ulteriori informazioni, vedere. Ahead-of-Time (AOT) e cache CDS

Sections

- Prerequisiti
- Strumenti e librerie
- Creazione di un pacchetto di distribuzione con Gradle
- Utilizzo dei livelli per le dipendenze
- Creazione di un pacchetto di distribuzione con Maven
- Caricamento di un pacchetto di implementazione con la console Lambda
- Caricamento di un pacchetto di distribuzione con AWS CLI
- Caricamento di un pacchetto di distribuzione con AWS SAM

Prerequisiti

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della AWS CLI versione 2.

Strumenti e librerie

AWS fornisce le seguenti librerie per le funzioni Java. Queste librerie sono disponibili tramite Maven Central Repository.

- com.amazonaws: aws-lambda-java-core (obbligatorio) — Definisce le interfacce del metodo del gestore e l'oggetto di contesto che il runtime passa al gestore. Se definisci i propri tipi di input, questa è l'unica libreria necessaria.
- com.amazonaws: aws-lambda-java-events — Tipi di input per eventi provenienti da servizi che richiamano funzioni Lambda.
- com.amazonaws: aws-lambda-java-log 4j2 — Una libreria di appender per Apache Log4j 2 che puoi utilizzare per aggiungere l'ID della richiesta per la chiamata corrente ai log delle funzioni.
- AWS SDK for Java 2.0 — L'SDK AWS ufficiale per il linguaggio di programmazione Java.

Aggiungi queste librerie alla definizione di build come segue.

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'
}
```

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>
  </dependencies>
```

Per creare un pacchetto di distribuzione, compila il codice della funzione e le dipendenze in un singolo file .zip o un file di archivio Java (JAR). Per Gradle, utilizza il tipo di build `Zip`. Per Apache Maven, utilizza il plugin Maven Shade. Per caricare il pacchetto di distribuzione, usa la console Lambda, l'API Lambda o (). AWS Serverless Application Model AWS SAM

> **Note**
>
> Per mantenere contenute le dimensioni del pacchetto di distribuzione, raggruppa le dipendenze della funzione in livelli. I livelli consentono di gestire le dipendenze in modo indipendente, possono essere utilizzate da più funzioni e possono essere condivisi con altri account. Per ulteriori informazioni, consulta Livelli Lambda.

Creazione di un pacchetto di distribuzione con Gradle

Per creare un pacchetto di implementazione con il codice e le dipendenze della funzione in Gradle, utilizza il tipo di compilazione `Zip`. Ecco un esempio tratto da un file build.gradle di esempio completo:

Example build.gradle: attività di compilazione

```
task buildZip(type: Zip) {
    into('lib') {
        from(jar)
        from(configurations.runtimeClasspath)
    }
}
```

Questa configurazione di build crea un pacchetto di distribuzione nella directory `build/distributions`. All'interno dell'istruzione `into('lib')`, l'attività `jar` assembla un archivio jar contenente le classi principali in una cartella denominata `lib`. Inoltre, l'istruzione `configurations.runtimeClassPath` copia le librerie delle dipendenze dal classpath della build in una cartella denominata `lib`.

Example build.gradle: dipendenze

```
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 carica i file JAR in ordine alfabetico Unicode. Se più file JAR nella directory `lib` contengono la stessa classe, viene utilizzato il primo. È possibile utilizzare il seguente script di shell per identificare le classi duplicate:

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
```

Utilizzo dei livelli per le dipendenze

È possibile impacchettare le dipendenze della funzione in livelli per mantenere piccolo il pacchetto di distribuzione e gestire le dipendenze in modo indipendente. Per ulteriori informazioni, consulta the section called “Livelli”.

Creazione di un pacchetto di distribuzione con Maven

Per creare un pacchetto di distribuzione con Maven, utilizzare il plugin Maven Shade. Il plugin crea un file JAR che contiene il codice della funzione compilato e tutte le relative dipendenze.

Example pom.xml: configurazione del 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>
```

Per compilare il pacchetto di distribuzione, utilizzare il comando `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] ------------------------------------------------------------------------
```

Questo comando genera un file JAR nella directory `target`.

> **Note**
>
> Se stai lavorando con un JAR multi-rilascio (MRJAR), devi includere l'MRJAR (ossia il JAR shaded prodotto dal plug-in Maven Shade) nella directory `lib` e comprimerlo prima di

caricare il pacchetto di implementazione in Lambda. In caso contrario, Lambda potrebbe non decomprimere correttamente il file JAR, facendo sì che il file `MANIFEST.MF` venga ignorato.

Se si utilizza la libreria appender (`aws-lambda-java-log4j2`), è necessario configurare anche un trasformatore per il plugin Maven Shade. La libreria del trasformatore combina le versioni di un file di cache presenti sia nella libreria appender che in Log4j.

Example pom.xml: configurazione del plugin con l'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>
```

Caricamento di un pacchetto di implementazione con la console Lambda

Per creare una nuova funzione, devi prima creare la funzione nella console, quindi devi caricare il tuo file .zip o JAR. Per aggiornare una funzione esistente, apri la pagina relativa alla tua funzione, quindi segui la stessa procedura per aggiungere il file .zip o JAR aggiornato.

Se il file del pacchetto di implementazione ha dimensioni inferiori a 50 MB, è possibile creare o aggiornare una funzione caricando il file direttamente dal computer locale. Per i file .zip o JAR di dimensioni superiori a 50 MB, prima è necessario caricare il pacchetto in un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando il Console di gestione AWS, consulta la Guida introduttiva ad Amazon S3. Per caricare file utilizzando la AWS CLI, consulta Move objects nella Guida per l'AWS CLI utente.

> **Note**
>
> Non è possibile modificare il tipo di pacchetto di implementazione (.zip o immagine di container) per una funzione esistente. Ad esempio, non è possibile convertire una funzione di immagine di container esistente per utilizzare un archivio di file .zip. È necessario creare una nuova funzione.

Creazione di una nuova funzione (console)

1. Apri la pagina Funzioni della console Lambda e scegli Crea funzione.
2. Scegli Author from scratch (Crea da zero).
3. In Basic information (Informazioni di base) eseguire queste operazioni:
   a. In Nome funzione, inserisci il nome della funzione.
   b. Per Runtime, seleziona il runtime che desideri utilizzare.
   c. (Facoltativo) Per Architettura, scegli l'architettura del set di istruzioni per la funzione. L'architettura predefinita è x86_64. Assicurati che il pacchetto di implementazione per la tua funzione sia compatibile con l'architettura del set di istruzioni scelta.
4. (Opzionale) In Autorizzazioni espandere Modifica ruolo di esecuzione predefinito. Puoi creare un nuovo ruolo di esecuzione o utilizzare un ruolo esistente.
5. Scegli Crea funzione. Lambda crea una funzione di base "Hello world" utilizzando il runtime scelto.

Caricamento di un archivio .zip o JAR dal computer locale (console)

1. Nella pagina Funzioni della console Lambda, scegli la funzione per cui vuoi caricare il file .zip o JAR.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli File .zip o .jar.
5. Per caricare il file .zip o JAR, procedi come segue:
    a. Seleziona Carica, quindi seleziona il tuo file .zip o JAR nel selettore di file.
    b. Seleziona Apri.
    c. Scegli Save (Salva).

Caricamento di un archivio .zip o JAR da un bucket Amazon S3 (console)

1. Nella pagina Funzioni della console Lambda, scegli la funzione per cui vuoi caricare un nuovo file .zip o JAR.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli Posizione Amazon S3.
5. Incolla l'URL del link Amazon S3 del tuo file .zip e scegli Salva.

Caricamento di un pacchetto di distribuzione con AWS CLI

È possibile utilizzare la AWS CLI per creare una nuova funzione o aggiornare una funzione esistente mediante un file .zip o JAR. Usa la funzione create-function e update-function-codei comandi per distribuire il tuo pacchetto.zip o JAR. Se il file ha dimensioni inferiori a 50 MB, è possibile caricare il pacchetto da una posizione nella macchina di compilazione locale. Per i file di dimensioni superiori, è necessario caricare il pacchetto .zip o JAR da un bucket Amazon S3. Per istruzioni su come caricare un file su un bucket Amazon S3 utilizzando AWS CLI, consulta Move objects nella User Guide.AWS CLI

> **Note**
>
> Se carichi il tuo file.zip o JAR da un bucket Amazon S3 utilizzando AWS CLI il, il bucket deve trovarsi nella stessa posizione della Regione AWS tua funzione.

Per creare una nuova funzione utilizzando un file.zip o JAR con AWS CLI, devi specificare quanto segue:

- Il nome della funzione (`--function-name`)
- Il runtime della tua funzione (`--runtime`)
- Il nome della risorsa Amazon (ARN) del ruolo di esecuzione della funzione (`--role`)
- Il nome del metodo del gestore nel codice della funzione (`--handler`)

È inoltre necessario specificare la posizione del file .zip o JAR. Se il file .zip o JAR si trova in una cartella nella macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
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
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza l'opzione `--code` illustrata nel seguente comando di esempio. È necessario utilizzare il parametro `S3ObjectVersion` solo per gli oggetti con controllo delle versioni.

```
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
```

Per aggiornare una funzione esistente mediante la CLI, specifica il nome della funzione utilizzando il parametro `--function-name`. È inoltre necessario specificare la posizione del file .zip che desideri utilizzare per aggiornare il codice della funzione. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza le opzioni `--s3-bucket` e `--s3-key` come illustrato nel seguente comando di esempio. È necessario utilizzare il parametro `--s3-object-version` solo per gli oggetti con controllo delle versioni.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Caricamento di un pacchetto di distribuzione con AWS SAM

È possibile utilizzarlo AWS SAM per automatizzare le distribuzioni del codice funzionale, della configurazione e delle dipendenze. AWS SAM è un'estensione CloudFormation che fornisce una sintassi semplificata per la definizione di applicazioni serverless. Il modello di esempio seguente definisce una funzione con un pacchetto di distribuzione nella directory `build/distributions` utilizzata da 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
```

Per creare la funzione, utilizzare i comandi `package` e `deploy`. Questi comandi sono personalizzazioni per l' AWS CLI. Essi avvolgono altri comandi per caricare il pacchetto di distribuzione Amazon S3, riscrivere il modello con l'URI dell'oggetto e aggiornare il codice della funzione.

Lo script di esempio seguente esegue una compilazione Gradle e carica il pacchetto di distribuzione creato. Crea uno CloudFormation stack la prima volta che lo esegui. Se lo stack esiste già, lo script lo aggiorna.

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
```

Per un esempio pratico completo, consulta le seguenti applicazioni di esempio:

Applicazioni Lambda di esempio in Java

- example-java: una funzione Java che dimostra come utilizzare Lambda per elaborare gli ordini. Questa funzione illustra come definire e deserializzare un oggetto evento di input personalizzato, utilizzare l' AWS SDK e registrare l'output.
- java-basic: una raccolta di funzioni Java minimali con unit test e configurazione della registrazione dei log delle variabili.
- java-events: una raccolta di funzioni Java che contengono codice skeleton per la gestione degli eventi di vari servizi, ad esempio Gateway Amazon API, Amazon SQS e Amazon Kinesis. Queste funzioni utilizzano la versione più recente della aws-lambda-java-eventslibreria (3.0.0 e successive). Questi esempi non richiedono l' AWS SDK come dipendenza.
- s3-java – Una funzione Java che elabora gli eventi di notifica da Amazon S3 e utilizza la Java Class Library (JCL) per creare anteprime dai file di immagine caricati.
- layer-java: una funzione Java che illustra come utilizzare un livello Lambda per impacchettare dipendenze separate dal codice della funzione principale.

Distribuisci funzioni Java Lambda con immagini di container

Esistono tre modi per creare un'immagine di container per una funzione Lambda in Java:

- Utilizzo di un'immagine di base per Java AWS

  Le immagini di base AWS sono precaricate con un runtime in linguaggio, un client di interfaccia di runtime per gestire l'interazione tra Lambda e il codice della funzione e un emulatore di interfaccia di runtime per i test locali.
- Utilizzo di un'immagine di AWS base solo per il sistema operativo

  AWS Le immagini di base solo per il sistema operativo contengono una distribuzione Amazon Linux e l'emulatore di interfaccia di runtime. Queste immagini vengono comunemente utilizzate per creare immagini di container per linguaggi compilati, come Go e Rust, e per un linguaggio o una versione di linguaggio per cui Lambda non fornisce un'immagine di base, come Node.js 19. Puoi anche utilizzare immagini di base solo per il sistema operativo per implementare un runtime personalizzato. Per rendere l'immagine compatibile con Lambda, devi includere il client di interfaccia di runtime per Java nell'immagine.
- Utilizzo di un'immagine non di base AWS

  È possibile utilizzare un'immagine di base alternativa da un altro registro del container, come ad esempio Alpine Linux o Debian. Puoi anche utilizzare un'immagine personalizzata creata dalla tua organizzazione. Per rendere l'immagine compatibile con Lambda, devi includere il client di interfaccia di runtime per Java nell'immagine.

> **Tip**
>
> Per ridurre il tempo necessario all'attivazione delle funzioni del container Lambda, consulta Utilizzo di compilazioni a più fasi nella documentazione Docker. Per creare immagini di container efficienti, segui le best practice per scrivere file Docker.

Questa pagina spiega come creare, testare e implementare le immagini di container per Lambda.

Argomenti

- AWS immagini di base per Java
- Utilizzo di un'immagine di base per Java AWS

- Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

AWS immagini di base per Java

AWS fornisce le seguenti immagini di base per Java:

Tag	Runtime	Sistema operativo	Dockerfile	Raggiunta obsolescenza
25	Java 2.5	Amazon Linux 2023	Dockerfile per Java 2.5 attivo GitHub	30 giugno 2029
21	Java 21	Amazon Linux 2023	Dockerfile per Java 2.1 su GitHub	30 giugno 2029
17	Java 17	Amazon Linux 2	Dockerfile per Java 17 su GitHub	30 giugno 2027
11	Java 11	Amazon Linux 2	Dockerfile per Java 11 attivo GitHub	30 giugno 2027
8.al2	Java 8	Amazon Linux 2	Dockerfile per Java 8 attivo GitHub	30 giugno 2027

Archivio Amazon ECR: gallery.ecr. aws/lambda/java

Le immagini di base Java 21 e versioni successive si basano sull'immagine di container minima di Amazon Linux 2023. Le immagini di base precedenti utilizzavano Amazon Linux 2. AL2023 offre diversi vantaggi rispetto ad Amazon Linux 2, tra cui un ingombro di distribuzione ridotto e versioni aggiornate di librerie come. `glibc`

AL2Le immagini basate su 023 utilizzano `microdnf` (symlinked as`dnf`) come gestore di pacchetti anzichéyum, che è il gestore di pacchetti predefinito in Amazon Linux 2. `microdnf`è un'implementazione autonoma di. `dnf` Per un elenco dei pacchetti inclusi nelle immagini AL2 basate su 023, consulta le colonne Minimal Container in Confronto dei pacchetti installati su Amazon Linux 2023 Container Images. Per ulteriori informazioni sulle differenze tra AL2 023 e Amazon Linux 2, consulta la sezione Introduzione al runtime di Amazon Linux 2023 AWS Lambda sul AWS Compute Blog.

> **Note**
>
> Per eseguire immagini AL2 basate su 023 localmente, incluso with AWS Serverless Application Model (AWS SAM), devi usare Docker versione 20.10.10 o successiva.

Utilizzo di un'immagine di base per Java AWS

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Java (ad esempio, Amazon Corretto)
- Apache Maven o Gradle
- AWS CLI versione 2
- Docker (versione minima 25.0.0)
- Il plugin buildx di Docker.

Creazione di un'immagine da un'immagine di base

Maven

1. Esegui il comando seguente per creare un progetto Maven utilizzando l'archetipo per Lambda. I parametri seguenti sono obbligatori:

   - service — Il Servizio AWS client da utilizzare nella funzione Lambda. Per un elenco delle fonti disponibili, consulta aws-sdk-java-v2/services on. GitHub
   - region — Il Regione AWS luogo in cui si desidera creare la funzione Lambda.
   - groupId: il namespace completo del pacchetto dell'applicazione.
   - artifactId: il nome del progetto. Questo sarà il nome della directory per il progetto.

   In Linux e macOS, esegui questo comando:

```
mvn -B archetype:generate \
   -DarchetypeGroupId=software.amazon.awssdk \
   -DarchetypeArtifactId=archetype-lambda -Dservice=s3 -Dregion=US_WEST_2 \
   -DgroupId=com.example.myapp \
```

```
    -DartifactId=myapp
```

In PowerShell, esegui questo comando:

```
mvn -B archetype:generate `
    "-DarchetypeGroupId=software.amazon.awssdk" `
    "-DarchetypeArtifactId=archetype-lambda" "-Dservice=s3" "-Dregion=US_WEST_2"
 `
    "-DgroupId=com.example.myapp" `
    "-DartifactId=myapp"
```

L'archetipo Maven per Lambda è preconfigurato per la compilazione con Java SE 8 e include una dipendenza da AWS SDK per Java. Se crei il tuo progetto con un archetipo diverso o utilizzando un altro metodo, devi configurare il compilatore Java per Maven e dichiarare l'SDK come dipendenza.

2. Apri la directory *myapp*`/src/main/java/com/example/`*myapp* e cerca il file `App.java`. Questo è il codice per la funzione Lambda. A fini di test, puoi utilizzare il codice di esempio fornito o sostituirlo con il tuo codice personalizzato.
3. Torna alla directory principale del progetto e crea un nuovo Dockerfile con la seguente configurazione:

   - Imposta la proprietà `FROM` sull'URI dell'immagine di base.
   - Imposta l'argomento `CMD` specificando il gestore della funzione Lambda.

   Nota che l'esempio Dockerfile non include un'istruzione USER. Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

   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. Compila il progetto e raccogli le dipendenze di runtime.

```
mvn compile dependency:copy-dependencies -DincludeScope=runtime
```

5. Crea l'immagine Docker con il comando docker build. L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il tag `test`. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'`--provenance=false`opzione.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-
image:test .
```

> **Note**
>
> Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Se intendi creare una funzione Lambda utilizzando l'architettura del set di ARM64 istruzioni, assicurati di modificare il comando per utilizzare invece l'`--platform linux/arm64`opzione.

Gradle

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir example
cd example
```

2. Esegui il comando seguente per fare in modo che Gradle generi un nuovo progetto di applicazione Java nella directory `example` dell'ambiente. Per Seleziona script di creazione DSL, scegli 2: Groovy.

```
gradle init --type java-application
```

3. Apri la directory */example*/app/src/main/java/*example* e cerca il file App.java. Questo è il codice per la funzione Lambda. A fini di test, puoi utilizzare il codice di esempio seguente o sostituirlo con il tuo codice personalizzato.

   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. Apri il file build.gradle. Se stai utilizzando il codice della funzione di esempio del passaggio precedente, sostituisci il contenuto di build.gradle con il seguente. Se utilizzi un codice di funzione personalizzato, modifica il file build.gradle secondo necessità.

   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. Il comando gradle init del passaggio 2 ha anche generato un caso di test fittizio nella directory app/test. Ai fini di questo tutorial, salta l'esecuzione dei test eliminando la directory /test.

6. Compilare il progetto.

```
gradle build
```

7. Nella directory root del progetto (`/example`), crea un Dockerfile con la seguente configurazione:

   - Imposta la proprietà `FROM` sull'URI dell'immagine di base.
   - Utilizza il comando COPY per copiare il codice della funzione e le dipendenze di runtime in `{LAMBDA_TASK_ROOT}`, una variabile d'ambiente definita da Lambda.
   - Imposta l'argomento `CMD` specificando il gestore della funzione Lambda.

   Nota che l'esempio Dockerfile non include un'istruzione USER. Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione USER.

   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. Crea l'immagine Docker con il comando docker build. L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il tag `test`. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'`--provenance=false`opzione.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-
image:test .
```

> **Note**
>
> Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Se intendi creare una funzione Lambda utilizzando l'architettura del set di ARM64 istruzioni, assicurati di modificare il comando per utilizzare invece l'`--platform linux/arm64`opzione.

(Facoltativo) Test dell'immagine in locale

1. Avvia l'immagine Docker con il comando docker run. In questo esempio, `docker-image` è il nome dell'immagine e `test` è il tag.

   ```
   docker run --platform linux/amd64 -p 9000:8080 docker-image:test
   ```

   Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

   > **Note**
   >
   > Se hai creato l'immagine Docker per l'architettura del set di ARM64 istruzioni, assicurati di utilizzare l'`--platform linux/arm64`opzione invece di. `--platform linux/amd64`

2. Da una nuova finestra di terminale, invia un evento all'endpoint locale.

   Linux/macOS

   Su MacOS o Linux, esegui il comando seguente `curl`:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

In PowerShell, esegui il seguente `Invoke-WebRequest` comando:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

3. Ottieni l'ID del container.

```
docker ps
```

4. Utilizza il comando docker kill per arrestare il container. In questo comando, sostituisci `3766c4ab331c` con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

Implementazione dell'immagine

Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Esegui il get-login-passwordcomando per autenticare la CLI Docker nel tuo registro Amazon ECR.
   - Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
   - Sostituiscilo `111122223333` con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
    "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. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando docker tag per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:

- `docker-image:test` è il nome e tag dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
- Sostituisci l'`<ECRrepositoryUri>` con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando docker push per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crea un ruolo di esecuzione per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'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**
>
> È possibile creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta Autorizzazioni multiaccount Amazon ECR.

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nell'archivio Amazon ECR e quindi utilizzare il update-function-codecomando per distribuire l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per distribuire la nuova immagine nella stessa funzione Lambda, è necessario utilizzare update-function-codeil comando, anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

Se utilizzi un'immagine di base solo per il sistema operativo o un'immagine di base alternativa, devi includere il client dell'interfaccia di runtime nell'immagine. Il client dell'interfaccia di runtime estende l'API Runtime, che gestisce l'interazione tra Lambda e il codice della funzione.

Installa il client di interfaccia di runtime per Java nel tuo Dockerfile o come dipendenza nel tuo progetto. Ad esempio, per installare il client di interfaccia di runtime utilizzando il gestore di pacchetti Maven, aggiungi quanto segue al file `pom.xml`:

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-runtime-interface-client</artifactId>
    <version>2.3.2</version>
</dependency>
```

Per i dettagli del pacchetto, consulta la pagina Client di interfaccia di runtime AWS Lambda Java nel Maven Central Repository. È inoltre possibile esaminare il codice sorgente del client dell'interfaccia di runtime nel GitHub repository AWS Lambda Java Support Libraries.

L'esempio seguente dimostra come creare un'immagine di container per Java utilizzando un'immagine Amazon Corretto. Amazon Corretto è una distribuzione pronta per la produzione gratuita, con un ambiente multipiattaforma di Open Java Development Kit (OpenJDK). Il progetto Maven include il client di interfaccia di runtime come dipendenza.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Java (ad esempio, Amazon Corretto)
- Apache Maven
- AWS CLI versione 2
- Docker (versione minima 25.0.0)
- Il plugin buildx di Docker.

Creazione di un'immagine da un'immagine di base alternativa

1. Crea un progetto Maven. I parametri seguenti sono obbligatori:
    - groupId: il namespace completo del pacchetto dell'applicazione.
    - artifactId: il nome del progetto. Questo sarà il nome della directory per il progetto.

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. Apri la directory del progetto.

```
cd myapp
```

3. Apri il file `pom.xml` e sostituisci il contenuto con quanto riportato di seguito. Questo file include aws-lambda-java-runtime-interface-client come dipendenza. In alternativa, è possibile installare il client di interfaccia di runtime nel Dockerfile. Tuttavia, l'approccio più semplice consiste nell'includere la libreria come dipendenza.

```
<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. Apri la directory *myapp*`/src/main/java/com/example/`*myapp* e cerca il file `App.java`. Questo è il codice per la funzione Lambda. Sostituisci il codice con il seguente.

   Example gestore della funzione

   ```
   package example;

   public class App {
       public static String sayHello() {
           return "Hello world!";
       }
   }
   ```

5. Il comando `mvn -B archetype:generate` del passaggio 1 ha anche generato un test case fittizio nella directory `src/test`. Ai fini di questo tutorial, salta l'esecuzione dei test eliminando la directory `/test` generata interamente.

6. Torna alla directory principale del progetto e crea un nuovo Dockerfile. Il Dockerfile di esempio seguente utilizza un'immagine Amazon Corretto. Amazon Corretto è una distribuzione multi-piattaforma gratuita e pronta per la produzione di OpenJDK.

   - Imposta la proprietà `FROM` sull'URI dell'immagine di base.
   - Imposta l'`ENTRYPOINT` sul modulo su cui desideri che il container Docker venga eseguito all'avvio. In questo caso, il modulo è il client di interfaccia di runtime.
   - Imposta l'argomento `CMD` specificando il gestore della funzione Lambda.

   Nota che l'esempio Dockerfile non include un'istruzione USER. Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

   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. Crea l'immagine Docker con il comando docker build. L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il tag `test`. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'`--provenance=false`opzione.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Se intendi creare una funzione Lambda utilizzando l'architettura del set di ARM64 istruzioni, assicurati di modificare il comando per utilizzare invece l'`--platform linux/arm64`opzione.

(Facoltativo) Test dell'immagine in locale

Usa il simulatore dell'interfaccia di runtime per testare localmente l'immagine. Puoi creare l'emulatore nella tua immagine o seguire la procedura riportata e installarlo sul tuo computer locale.

Installazione ed esecuzione dell'emulatore di interfaccia di runtime sul computer locale

1. Dalla directory del progetto, esegui il comando seguente per scaricare l'emulatore di interfaccia di runtime (architettura x86-64) GitHub e installarlo sul computer locale.

   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
```

   Per installare l'emulatore arm64, sostituisci l'URL del GitHub repository nel comando precedente con il seguente:

```
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
```

Per installare l'emulatore arm64, sostituisci `$downloadLink` con quanto segue:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Avvia l'immagine Docker con il comando docker run. Tenere presente quanto segue:

   - `docker-image` è il nome dell'immagine e `test` è il tag.
   - `/usr/bin/java -cp './*' com.amazonaws.services.lambda.runtime.api.client.AWSLambda example.App::sayHello` è l'`ENTRYPOINT` seguito dal `CMD` del Dockerfile.

   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
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Se hai creato l'immagine Docker per l'architettura del set di ARM64 istruzioni, assicurati di utilizzare l'`--platform linux/arm64`opzione invece di. `--platform linux/amd64`

3. Pubblica un evento nell'endpoint locale.

Linux/macOS

Su MacOS o Linux, esegui il comando seguente `curl`:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
 '{"payload":"hello world!"}'
```

PowerShell

In PowerShell, esegui il seguente `Invoke-WebRequest` comando:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{}' -ContentType "application/json"
```

Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Ottieni l'ID del container.

```
docker ps
```

5. Utilizza il comando docker kill per arrestare il container. In questo comando, sostituisci `3766c4ab331c` con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

Implementazione dell'immagine

Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Esegui il get-login-passwordcomando per autenticare la CLI Docker nel tuo registro Amazon ECR.
   - Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
   - Sostituiscilo `111122223333` con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il commando create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
    "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. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando docker tag per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:

   - `docker-image:test` è il nome e tag dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
   - Sostituisci l'`<ECRrepositoryUri>` con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando docker push per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crea un ruolo di esecuzione per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'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**
>
> È possibile creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta Autorizzazioni multiaccount Amazon ECR.

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{
  "ExecutedVersion": "$LATEST",
```

```
  "StatusCode": 200
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nell'archivio Amazon ECR e quindi utilizzare il update-function-codecomando per distribuire l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per distribuire la nuova immagine nella stessa funzione Lambda, è necessario utilizzare update-function-codeil comando, anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Utilizzo dei livelli per le funzioni Lambda in Java

Usa i livelli Lambda per impacchettare codice e dipendenze che desideri riutilizzare in più funzioni. I livelli di solito contengono dipendenze dalla libreria, un runtime personalizzato o file di configurazione. La creazione di un livello prevede tre passaggi generali:

1. Crea un pacchetto per il contenuto del livello. Ciò significa creare un archivio di file con estensione .zip che contiene le dipendenze che desideri utilizzare nelle funzioni.
2. Crea il livello in Lambda.
3. Aggiungi il livello alle tue funzioni.

Argomenti

- Crea un pacchetto per il contenuto del livello
- Crea il layer in Lambda
- Aggiungete il layer alla vostra funzione

Crea un pacchetto per il contenuto del livello

Per creare un layer, raggruppa i pacchetti in un archivio di file.zip che soddisfi i seguenti requisiti:

- Assicurati che la versione Java a cui fanno riferimento Maven o Gradle sia la stessa della versione Java della funzione che intendi distribuire. Ad esempio, per una funzione Java 25, il `mvn -v` comando dovrebbe elencare Java 25 nell'output.
- Le dipendenze devono essere archiviate nella `java/lib` directory, alla radice del file.zip. Per ulteriori informazioni, consulta Percorsi dei livelli per ciascun runtime Lambda.
- I pacchetti del livello devono essere compatibili con Linux. Le funzioni Lambda vengono eseguite su Amazon Linux.

Puoi creare livelli che contengono librerie Java di terze parti o moduli e pacchetti Java personalizzati. La procedura seguente utilizza Maven. Puoi anche usare Gradle per impacchettare il contenuto del tuo layer.

Per creare un livello utilizzando le dipendenze di Maven

1. Crea un progetto Apache Maven con un `pom.xml` file che definisce le tue dipendenze.

L'esempio seguente include Jackson Databind per l'elaborazione JSON. La <build> sezione utilizza il maven-dependency-pluginper creare file JAR separati per ogni dipendenza invece di raggrupparli in un unico uber-jar. Se vuoi creare un uber-jar, usa il. 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. Compilare il progetto. Questo comando crea tutti i file JAR di dipendenza nella directory. `target/lib/`

```
mvn clean package
```

3. Crea la struttura di directory richiesta per il tuo livello:

```
mkdir -p java/lib
```

4. Copia i file JAR delle dipendenze nella `java/lib` directory:

```
cp target/lib/*.jar java/lib/
```

5. Comprimi il contenuto del livello:

   Linux/macOS

   ```
   zip -r layer.zip java/
   ```

   PowerShell

   ```
   Compress-Archive -Path .\java -DestinationPath .\layer.zip
   ```

   La struttura delle cartelle del file.zip dovrebbe essere simile alla seguente:

```
java/
### lib/
    ### jackson-databind-2.17.0.jar
    ### jackson-core-2.17.0.jar
    ### jackson-annotations-2.17.0.jar
```

> **Note**
>
> Assicurati che il tuo file.zip includa la `java` directory a livello principale al `lib` suo interno. Questa struttura assicura che Lambda possa localizzare e importare le tue librerie. Ogni dipendenza viene conservata come file JAR separato anziché raggruppata in un uber-jar.

Crea il layer in Lambda

È possibile pubblicare il layer utilizzando la console AWS CLI o la console Lambda.

AWS CLI

Esegui il publish-layer-version AWS CLI comando per creare il livello Lambda:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip
 --compatible-runtimes java25
```

Il parametro di runtime compatibile è facoltativo. Quando specificato, Lambda utilizza questo parametro per filtrare i livelli nella console Lambda.

Console

Creazione di un livello (console)

1. Apri la pagina Layers (Livelli) nella console Lambda.
2. Scegli Create layer (Crea livello).
3. Scegli Carica un file.zip, quindi carica l'archivio.zip che hai creato in precedenza.
4. (Facoltativo) Per i runtime compatibili, scegliete il runtime Java che corrisponde alla versione Java utilizzata per creare il layer.
5. Scegli Create (Crea).

Aggiungete il layer alla vostra funzione

AWS CLI

Per collegare il layer alla tua funzione, esegui il update-function-configuration AWS CLI comando. Per il `--layers` parametro, utilizzate il livello ARN. L'ARN deve specificare la versione (ad esempio,`arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`). Per ulteriori informazioni, consulta Livelli e versioni di livelli.

```
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"
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Console

Aggiunta di un livello a una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la funzione.
3. Scorri verso il basso fino alla sezione Livelli, quindi scegli Aggiungi un livello.
4. In Scegli un livello, seleziona Livelli personalizzati, quindi scegli il tuo livello.

> **Note**
>
> Se non hai aggiunto un runtime compatibile quando hai creato il layer, il tuo layer non verrà elencato qui. È possibile specificare invece il livello ARN.

5. Scegliere Aggiungi.

Personalizzare la serializzazione per le funzioni Lambda Java

I runtime gestiti da Java di Lambda supportano la serializzazione personalizzata per gli eventi JSON. La serializzazione personalizzata può semplificare il codice e potenzialmente migliorare le prestazioni.

Argomenti

- Quando usare la serializzazione personalizzata
- Implementazione della serializzazione personalizzata
- Test della serializzazione personalizzata

Quando usare la serializzazione personalizzata

Quando viene richiamata la funzione Lambda, i dati degli eventi di input devono essere deserializzati in un oggetto Java e l'output della funzione deve essere nuovamente serializzato in un formato che può essere restituito come risposta della funzione. I runtime gestiti da Java per Lambda forniscono funzionalità di serializzazione e deserializzazione predefinite che funzionano bene per la gestione dei payload di eventi da vari servizi AWS, come Gateway Amazon API e Amazon Simple Queue Service (Amazon SQS). Per utilizzare questi eventi di integrazione dei servizi nella tua funzione, aggiungi la dipendenza aws-java-lambda-events al tuo progetto. Questa libreria AWS contiene oggetti Java che rappresentano questi eventi di integrazione dei servizi.

Puoi anche usare i tuoi oggetti per rappresentare l'evento JSON che passi alla tua funzione Lambda. Il runtime gestito prova a serializzare il JSON su una nuova istanza dell'oggetto con il suo comportamento predefinito. Se il serializzatore predefinito non ha il comportamento desiderato per il tuo caso d'uso, usa la serializzazione personalizzata.

Supponiamo, ad esempio, che l'handler di funzioni si aspetti una classe `Vehicle` come input, con la seguente struttura:

```
public class Vehicle {
    private String vehicleType;
    private long vehicleId;
}
```

Tuttavia, il payload dell'evento JSON ha il seguente aspetto:

```
{
```

```
    "vehicle-type": "car",
    "vehicleID": 123
}
```

In questo scenario, la serializzazione predefinita nel runtime gestito prevede che i nomi delle proprietà JSON corrispondano ai nomi delle proprietà della classe Java con notazione a cammello (`vehicleType`, `vehicleId`). Poiché i nomi delle proprietà nell'evento JSON non sono in notazione a cammello (`vehicle-type`, `vehicleID`), è necessario utilizzare la serializzazione personalizzata.

Implementazione della serializzazione personalizzata

Utilizza un'interfaccia del provider di servizi per caricare un serializzatore di tua scelta anziché la logica di serializzazione predefinita del runtime gestito. È possibile serializzare i payload di eventi JSON direttamente in oggetti Java, utilizzando l'interfaccia `RequestHandler` standard.

Per utilizzare la serializzazione personalizzata nella funzione Java per Lambda

1. Aggiungi la libreria aws-lambda-java-core come dipendenza. Questa libreria include l'interfaccia CustomPojoSerializer, insieme ad altre definizioni di interfaccia per lavorare con Java in Lambda.
2. Crea un file denominato `com.amazonaws.services.lambda.runtime.CustomPojoSerializer` nella directory `src/main/resources/META-INF/services/` del progetto.
3. In questo file, specifica il nome completo dell'implementazione del serializzatore personalizzato, che deve implementare l'interfaccia `CustomPojoSerializer`. Esempio:

   ```
   com.mycompany.vehicles.CustomLambdaSerialzer
   ```
4. Implementa l'interfaccia `CustomPojoSerializer` per fornire la tua logica di serializzazione personalizzata.
5. Usa l'interfaccia `RequestHandler` standard nella tua funzione Lambda. Il runtime gestito utilizzerà il serializzatore personalizzato.

Per altri esempi di come implementare la serializzazione personalizzata tramite le librerie comuni come fastJson, Gson, Moshi e jackson-jr, consulta l'esempio custom-serialization nel repository GitHub AWS.

Test della serializzazione personalizzata

Verifica la tua funzione per assicurarti che la logica di serializzazione e deserializzazione funzioni come previsto. È possibile utilizzare l'interfaccia a riga di comando AWS Serverless Application Model (AWS SAM CLI) per emulare l'invocazione del payload Lambda. Questo può aiutarti a testare e iterare rapidamente la tua funzione man mano che introduci un serializzatore personalizzato.

1. Crea un file con il payload dell'evento JSON con cui desideri richiamare la tua funzione, quindi chiama la CLI AWS SAM.
2. Esegui il comando sam local invoke per richiamare la tua funzione in locale. Esempio:

```
sam local invoke -e src/test/resources/event.json
```

Per ulteriori informazioni, consulta Richiamare in locale le funzioni Lambda con AWS SAM.

Personalizzare il comportamento di avvio del runtime Java per le funzioni Lambda

Questa pagina descrive le impostazioni specifiche delle funzioni Java in AWS Lambda. È possibile utilizzare queste impostazioni per personalizzare il comportamento di avvio del runtime Java. Ciò può ridurre la latenza complessiva della funzione e migliorare le prestazioni complessive delle funzioni senza dover modificare il codice.

Sections

- Informazioni sulla variabile di ambiente JAVA_TOOL_OPTIONS
- Patch Log4j per Log4Shell
- Ahead-of-Time (AOT) e cache CDS

Informazioni sulla variabile di ambiente **`JAVA_TOOL_OPTIONS`**

In Java, Lambda supporta la variabile di ambiente `JAVA_TOOL_OPTIONS` per impostare variabili della linea di comando aggiuntive in Lambda. È possibile utilizzare questa variabile di ambiente in vari modi, ad esempio per personalizzare le impostazioni di compilazione a più livelli. L'esempio seguente illustra come utilizzare la variabile di ambiente `JAVA_TOOL_OPTIONS` per questo caso d'uso.

Esempio: personalizzazione delle impostazioni di compilazione a più livelli

La compilazione a livelli è una funzionalità della macchina virtuale Java (JVM). È possibile utilizzare impostazioni di compilazione specifiche a più livelli per utilizzare al meglio i compilatori JVM just-in-time (JIT). In genere, il compilatore C1 è ottimizzato per tempi di avvio rapidi. Il compilatore C2 è ottimizzato per le migliori prestazioni complessive, ma di contro utilizza più memoria e impiega più tempo per raggiungerle. Esistono 5 diversi livelli di compilazione a più livelli. Al livello 0, la JVM interpreta il bytecode Java. Al livello 4, la JVM utilizza il compilatore C2 per analizzare i dati di profilazione raccolti durante l'avvio dell'applicazione. Nel tempo, monitora l'utilizzo del codice per identificare le migliori ottimizzazioni.

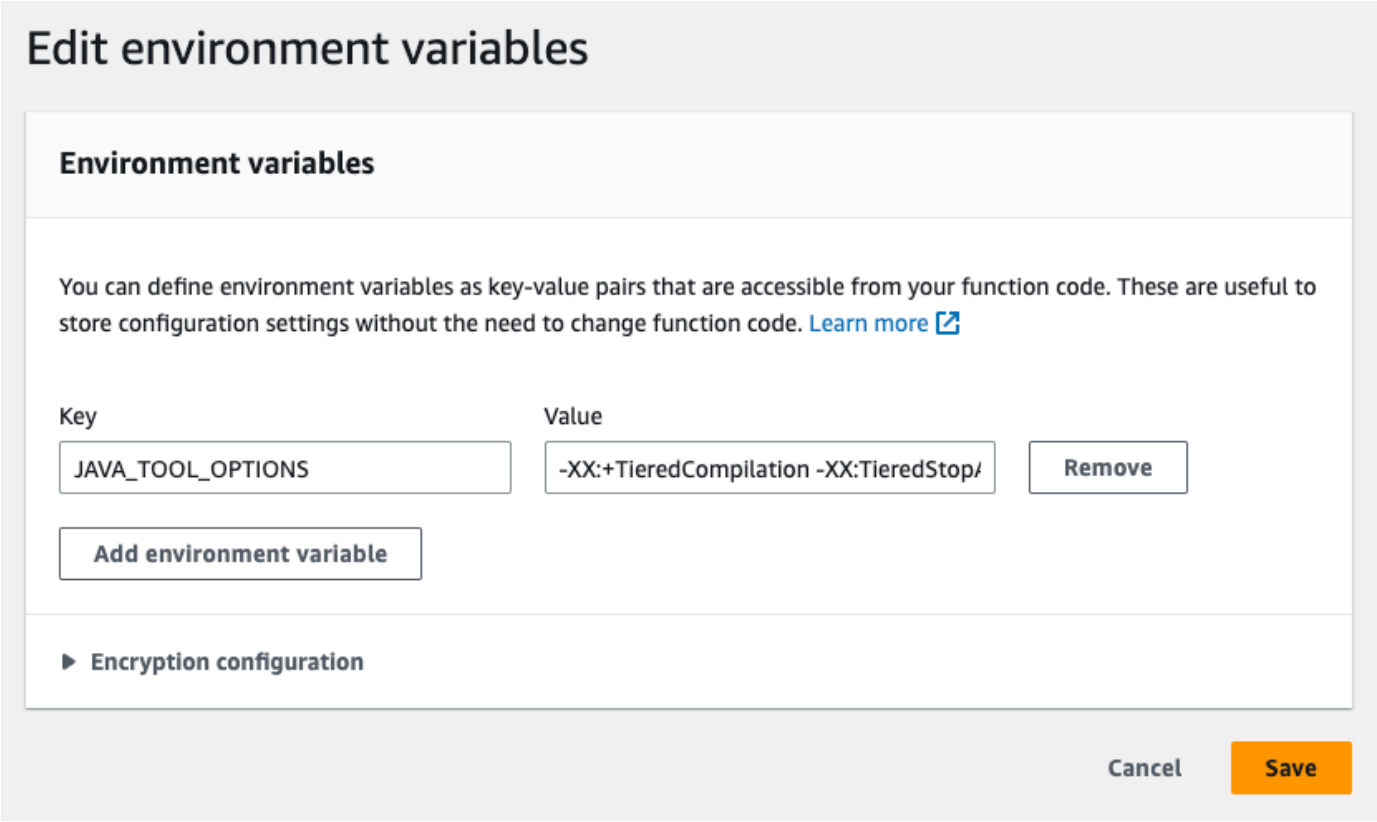
La personalizzazione del livello di compilazione a più livelli può aiutarti a ottimizzare le prestazioni delle funzioni Java. Per le piccole funzioni che vengono eseguite rapidamente, l'impostazione della compilazione a più livelli sul livello 1 può contribuire a migliorare le prestazioni di avvio a freddo facendo in modo che JVM utilizzi il compilatore C1. Questa impostazione produce rapidamente codice nativo ottimizzato, ma non genera dati di profilazione e non utilizza mai il compilatore C2. Per

le funzioni più grandi e ad alta intensità di calcolo, l'impostazione della compilazione a più livelli al livello 4 massimizza le prestazioni complessive a scapito del consumo di memoria aggiuntivo e del lavoro di ottimizzazione aggiuntivo durante le prime richiami dopo il provisioning di ogni ambiente di esecuzione Lambda.

Per i runtime Java 11 e versioni precedenti, Lambda utilizza le impostazioni di compilazione a più livelli JVM predefinite. Per Java 17 e Java 21, Lambda configura la JVM per interrompere la compilazione su più livelli al livello 1 per impostazione predefinita. A partire da Java 25, Lambda interrompe ancora la compilazione a più livelli al livello 1 per impostazione predefinita, tranne quando si utilizza SnapStart o viene fornita la concorrenza, nel qual caso vengono utilizzate le impostazioni JVM predefinite. Ciò migliora le prestazioni SnapStart e la concorrenza di Provisioned senza incorrere in penalità di avvio a freddo, poiché in questi casi la compilazione a più livelli viene eseguita al di fuori del percorso di invoke. Per massimizzare questo vantaggio, è possibile utilizzare i percorsi di codice di esecuzione priming durante l'inizializzazione della funzione per attivare JIT prima di scattare l'istantanea o quando gli ambienti di esecuzione Provisioned Concurrency sono predisposti. SnapStart Per ulteriori informazioni, consulta il post sul blog Ottimizzazione delle prestazioni di avviamento a freddo di AWS Lambda utilizzando strategie di adescamento avanzate con. SnapStart

Personalizzazione delle impostazioni di compilazione a più livelli (console)

1. Apri la pagina Funzioni della console Lambda.
2. Scegli una funzione Java per la quale desideri personalizzare la compilazione a più livelli.
3. Scegli la scheda Configurazione, quindi scegli Variabili di ambiente nel menu a sinistra.
4. Scegli Modifica.
5. Scegli Add environment variable (Aggiungi variabile d'ambiente).
6. Per la chiave, inserisci `JAVA_TOOL_OPTIONS`. Per il valore, inserisci `-XX:+TieredCompilation -XX:TieredStopAtLevel=1`.



7. Scegli Save (Salva).

> **Note**
>
> Puoi anche usare Lambda SnapStart per mitigare i problemi di avvio a freddo. SnapStartutilizza istantanee memorizzate nella cache dell'ambiente di esecuzione per migliorare significativamente le prestazioni di avvio. Per ulteriori informazioni su SnapStart funzionalità, limitazioni e aree supportate, consulta. Migliorare le prestazioni di avvio con Lambda SnapStart

Esempio: personalizzazione del comportamento del GC utilizzando JAVA_TOOL_OPTIONS

I runtime di Java 11 utilizzano Serial Garbage Collector (GC) per la rimozione di oggetti inutili (garbage collection). Per impostazione predefinita, anche i runtime di Java 17 utilizzano Serial GC. Tuttavia, con Java 17 puoi anche utilizzare la variabile di ambiente `JAVA_TOOL_OPTIONS` per modificare il GC predefinito. Puoi scegliere tra Parallel GC e Shenandoah GC.

Ad esempio, se il tuo carico di lavoro utilizza più memoria e più memoria CPUs, prendi in considerazione l'utilizzo di Parallel GC per prestazioni migliori. Puoi farlo aggiungendo quanto segue al valore della tua variabile di ambiente `JAVA_TOOL_OPTIONS`:

```
-XX:+UseParallelGC
```

Se il tuo carico di lavoro include molti oggetti di breve durata, potresti trarre vantaggio da un minore consumo di memoria abilitando la modalità generazionale dello Shenandoah Garbage Collector introdotto in Java 25. Per fare ciò, aggiungi quanto segue al valore della tua variabile di ambiente: `JAVA_TOOL_OPTIONS`

```
-XX:+UseShenandoahGC -XX:ShenandoahGCMode=generational
```

Patch Log4j per Log4Shell

I runtime Lambda per Java 8, 11, 17 e 21 includono una patch per mitigare la vulnerabilità Log4Shell (CVE-2021-44228) in Log4j, un popolare framework di registrazione Java. Questa patch comporta un sovraccarico prestazionale dovuto all'avvio a freddo. Se stai usando una versione patchata di Log4j (versione 2.17.0 o successiva), puoi disabilitare questa patch per migliorare le prestazioni di avvio a freddo. Per disabilitare la patch, imposta la variabile di ambiente su. `AWS_LAMBDA_DISABLE_CVE_2021_44228_PROTECTION` `true`

A partire da Java 25, i runtime Lambda non includono più la patch Log4Shell. È necessario verificare di utilizzare Log4j versione 2.17.0 o successiva.

Ahead-of-Time (AOT) e cache CDS

A partire da Java 25, il runtime Lambda include una cache Ahead-of-Time (AOT) per il Java Runtime Interface Client (RIC), un componente di runtime che esegue attivamente il polling degli eventi dall'API Lambda Runtime. Ciò migliora le prestazioni di avvio a freddo.

Le cache AOT sono specifiche per una build JVM. Quando Lambda aggiorna il runtime gestito, aggiorna anche la cache AOT per il RIC. Tuttavia, se distribuisci le tue cache AOT, queste potrebbero essere invalidate o causare un comportamento imprevisto a seguito di un aggiornamento del runtime. Pertanto, consigliamo vivamente di non utilizzare le cache AOT quando si utilizzano runtime gestiti. Per utilizzare le cache AOT, è necessario distribuire le funzioni utilizzando le immagini dei contenitori.

Le cache AOT non possono essere utilizzate con le cache CDS (Class Data Sharing). Se distribuisci cache CDS nella tua funzione Lambda, Lambda disabilita la cache AOT.

Utilizzo dell'oggetto di contesto Lambda per recuperare le informazioni sulla funzione Java

Quando Lambda esegue la funzione, passa un oggetto Context al gestore. Questo oggetto fornisce i metodi e le proprietà che forniscono le informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

Metodi del contesto

- `getRemainingTimeInMillis()`: restituisce il numero di millisecondi rimasti prima del timeout dell'esecuzione.
- `getFunctionName()`: restituisce il nome della funzione Lambda.
- `getFunctionVersion()`: restituisce la versione della funzione.
- `getInvokedFunctionArn()`: restituisce l'ARN (Amazon Resource Name) utilizzato per invocare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `getMemoryLimitInMB()`: restituisce la quantità di memoria allocata per la funzione.
- `getAwsRequestId()`: restituisce l'identificatore della richiesta di invocazione.
- `getLogGroupName()`: restituisce il gruppo di log per la funzione.
- `getLogStreamName()`: restituisce il flusso di log per l'istanza della funzione.
- `getIdentity()`: (app per dispositivi mobili) restituisce informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
- `getClientContext()`: (app per dispositivi mobili) restituisce il contesto client fornito a Lambda dall'applicazione client.
- `getLogger()`: restituisce l'oggetto logger per la funzione.

Nell'esempio seguente viene illustrata una funzione che utilizza l'oggetto contesto per accedere al 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;
  }
}
```

La funzione registra il tipo di classe dell'evento in entrata prima di restituire `null`.

Example Output log

```
EVENT TYPE: class java.util.LinkedHashMap
```

L'interfaccia per l'oggetto contesto è disponibile nella libreria aws-lambda-java-core. È possibile implementare questa interfaccia per creare una classe di contesto per il test. L'esempio seguente mostra una classe di contesto che restituisce valori fittizi per la maggior parte delle proprietà e un logger di test funzionante.

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();
  }

}
```

Per ulteriori informazioni sulla registrazione, consulta Registrare e monitorare funzioni Lambda in Java.

Contesto nelle applicazioni di esempio

Il repository GitHub per questa guida include applicazioni di esempio che illustrano l'uso dell'oggetto contesto. Ogni applicazione di esempio include script per semplificare la distribuzione e la pulizia, un modello AWS Serverless Application Model (AWS SAM) e risorse di supporto.

Applicazioni Lambda di esempio in Java

- example-java — Una funzione Java che dimostra come utilizzare Lambda per elaborare gli ordini. Questa funzione illustra come definire e deserializzare un oggetto evento di input personalizzato, utilizzare l'SDK e registrare l'output. AWS
- java-basic: una raccolta di funzioni Java minimali con unit test e configurazione della registrazione dei log delle variabili.
- java-events: una raccolta di funzioni Java che contengono codice skeleton per la gestione degli eventi di vari servizi, ad esempio Gateway Amazon API, Amazon SQS e Amazon Kinesis. Queste funzioni utilizzano la versione più recente della libreria aws-lambda-java-events (3.0.0 e versioni successive). Questi esempi non richiedono SDK AWS come dipendenza.
- s3-java – Una funzione Java che elabora gli eventi di notifica da Amazon S3 e utilizza la Java Class Library (JCL) per creare anteprime dai file di immagine caricati.
- layer-java — Una funzione Java che illustra come utilizzare un livello Lambda per impacchettare dipendenze separate dal codice della funzione principale.

Registrare e monitorare funzioni Lambda in Java

AWS Lambda monitora automaticamente le funzioni Lambda e invia le voci di registro ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della funzione. L'ambiente di runtime di Lambda invia al flusso di log i dettagli su ogni invocazione e altri output dal codice della funzione. Per ulteriori informazioni sui CloudWatch registri, consulta. Lambda invia automaticamente i log delle funzioni a CloudWatch Logs.

Per generare output di log dal codice della funzione, puoi utilizzare i metodi di java.lang.System o qualsiasi modulo di registrazione che scriva in stdout o stderr.

Sections

- Creazione di una funzione che restituisce i registri
- Utilizzo dei controlli di registrazione avanzati di Lambda con Java
- Implementazione della registrazione avanzata con Log4j2 e J SLF4
- Utilizzo di altri strumenti di registrazione e librerie
- Utilizzo di Powertools per AWS Lambda (Java) e per la registrazione strutturata AWS SAM
- Visualizzazione dei log nella console Lambda
- Visualizzazione dei log nella console CloudWatch
- Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI
- Eliminazione dei log
- Codice di registrazione dei log di esempio

Creazione di una funzione che restituisce i registri

Per i log di output del codice della funzione, puoi usare i metodi di java.lang.System o qualsiasi modulo di registrazione che scriva in `stdout` o `stderr`. La aws-lambda-java-corelibreria fornisce una classe logger denominata a `LambdaLogger` cui è possibile accedere dall'oggetto context. La classe di logger supporta i log multilinea.

Nell'esempio seguente viene utilizzato il logger `LambdaLogger` fornito dall'oggetto contestuale.

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 Formato dei 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
```

Il runtime di Java registra `START`, `END` e `REPORT` per ogni chiamata. La riga del report fornisce i seguenti dettagli:

Campi dati della riga REPORT

- RequestId— L'ID univoco della richiesta per la chiamata.
- Durata – La quantità di tempo che il metodo del gestore della funzione impiega durante l'elaborazione dell'evento.
- Durata fatturata – La quantità di tempo fatturata per la chiamata.
- Dimensioni memoria – La quantità di memoria allocata per la funzione.
- Quantità max utilizzata – La quantità di memoria utilizzata dalla funzione. Quando le invocazioni condividono un ambiente di esecuzione, Lambda riporta la memoria massima utilizzata in tutte le invocazioni. Questo comportamento potrebbe comportare un valore riportato superiore al previsto.
- Durata Init – Per la prima richiesta servita, la quantità di tempo impiegato dal runtime per caricare la funzione ed eseguire il codice al di fuori del metodo del gestore.
- XRAY TraceId — Per le richieste tracciate, l'ID di traccia.AWS X-Ray
- SegmentId— Per le richieste tracciate, l'ID del segmento X-Ray.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

Utilizzo dei controlli di registrazione avanzati di Lambda con Java

Per avere un maggiore controllo sul modo in cui i log delle tue funzioni vengono acquisiti, elaborati e utilizzati, puoi configurare le seguenti opzioni di registrazione per i runtime Java supportati:

- Formato di log: scegli tra i formati di testo normale e JSON strutturato per i log della funzione
- Livello di registro: per i log in formato JSON, scegli il livello di dettaglio dei log a cui Lambda invia, CloudWatch ad esempio ERROR, DEBUG o INFO
- Gruppo di log: scegli il gruppo di log a cui la CloudWatch funzione invia i log

Per ulteriori informazioni su queste opzioni di registrazione e istruzioni su come configurare la funzione per utilizzarle, consulta la pagina the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”.

Per utilizzare le opzioni del formato di log e del livello di log con le funzioni Lambda in Java, consulta le istruzioni nelle sezioni seguenti.

Utilizzo del formato di log JSON strutturato con Java

Se si seleziona JSON come formato di log della funzione, Lambda invierà gli output log utilizzando la classe `LambdaLogger` come JSON strutturati. Ogni oggetto di log JSON contiene almeno quattro coppie chiave-valore con le seguenti chiavi:

- `"timestamp"`: l'ora in cui è stato generato il messaggio di log
- `"level"`: il livello di log assegnato al messaggio
- `"message"`: il contenuto del messaggio di log
- `"AWSrequestId"`: l'ID di richiesta univoco dell'invocazione alla funzione

A seconda del metodo di registrazione di log utilizzato, gli output di log della funzione acquisiti in formato JSON possono contenere anche coppie di chiave-valore aggiuntive.

Per assegnare un livello ai log creati utilizzando il logger `LambdaLogger`, è necessario fornire un argomento `LogLevel` nel comando di registrazione, come mostrato nell'esempio seguente.

Example Codice di registrazione di Java

```
LambdaLogger logger = context.getLogger();
logger.log("This is a debug log", LogLevel.DEBUG);
```

L'output di registro di questo codice di esempio verrebbe acquisito in CloudWatch Logs come segue:

Example Record di log JSON

```
{
    "timestamp":"2023-11-01T00:21:51.358Z",
    "level":"DEBUG",
    "message":"This is a debug log",
    "AWSrequestId":"93f25699-2cbf-4976-8f94-336a0aa98c6f"
}
```

Se non assegni un livello all'output log, Lambda gli assegnerà automaticamente il livello INFO.

Se il codice utilizza già un'altra libreria di registrazione per generare log JSON strutturati, non è necessario apportare alcuna modifica. Lambda non codifica due volte i log già codificati in JSON.

Anche se configuri la tua funzione per utilizzare il formato di log JSON, i tuoi output di registrazione vengono visualizzati CloudWatch nella struttura JSON che definisci.

Utilizzo del filtraggio a livello di log con Java

Per AWS Lambda filtrare i log delle applicazioni in base al loro livello di registro, la funzione deve utilizzare log in formato JSON. Puoi farlo in due modi:

- Crea output log utilizzando il `LambdaLogger` standard e configura la tua funzione per utilizzare la formattazione dei log JSON. Successivamente, Lambda filtra gli output log utilizzando la coppia chiave-valore "livello" nell'oggetto JSON descritto in the section called “Utilizzo del formato di log JSON strutturato con Java”. Per informazioni su come configurare il formato di log della funzione, consulta la pagina the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”.
- Utilizza un'altra libreria o metodo di registrazione per creare nel codice dei log JSON strutturati che includono una coppia chiave-valore "livello" che definisce il livello dell'output log. Puoi utilizzare qualunque libreria di registrazione che sia in grado di scrivere log JSON in `stdout` o `stderr`. Ad esempio, puoi utilizzare Powertools for AWS Lambda o il pacchetto Log4j2 per generare output di log strutturati JSON dal tuo codice. Per ulteriori informazioni, consulta le pagine the section called “Utilizzo di Powertools per AWS Lambda (Java) e per la registrazione strutturata AWS SAM” e the section called “Implementazione della registrazione avanzata con Log4j2 e J SLF4”.

Quando configuri la tua funzione per utilizzare il filtraggio a livello di log, devi selezionare una delle seguenti opzioni per il livello di log che Lambda invii a Logs: CloudWatch

Livello di log	Utilizzo standard
TRACE (dettaglio massimo)	Le informazioni più dettagliate utilizzate per tracciare il percorso di esecuzione del codice
DEBUG	Informazioni dettagliate per il debug del sistema
INFO	Messaggi che registrano il normale funzionam ento della funzione
WARN	Messaggi relativi a potenziali errori che possono portare a comportamenti imprevisti se non risolti

Livello di log	Utilizzo standard
ERRORE	Messaggi relativi a problemi che impediscono al codice di funzionare come previsto
FATAL (dettaglio minimo)	Messaggi relativi a errori gravi che causano l'interruzione del funzionamento dell'appl icazione

Per consentire a Lambda di filtrare i log della funzione, è necessario includere anche una coppia chiave-valore `"timestamp"` nell'output log JSON. L'ora deve essere specificata in un formato di timestamp RFC 3339 valido. Se non fornisci un timestamp valido, Lambda assegnerà al log il livello INFO e aggiungerà un timestamp per tuo conto.

Lambda invia i log del livello selezionato e inferiore a. CloudWatch Ad esempio, se configuri un livello di log WARN, Lambda invierà i log corrispondenti ai livelli WARN, ERROR e FATAL.

Implementazione della registrazione avanzata con Log4j2 e J SLF4

> **Note**
>
> AWS Lambda non include Log4j2 nei suoi runtime gestiti o nelle immagini dei contenitori di base. Pertanto, non sono influenzati dai problemi descritti in CVE-2021-44228, CVE-2021-45046 e CVE-2021-45105.
> Per i casi in cui una funzione cliente include una versione Log4j2 interessata, abbiamo applicato una modifica ai temèi di esecuzione gestiti Lambda Java e alle immagini del container di base che aiutano a mitigare i problemi in CVE-2021-44228, CVE-2021-45046 e CVE-2021-45105. Come risultato di questa modifica, i clienti che utilizzano Log4J2 potrebbero vedere una voce di log aggiuntiva, simile a "`Transforming org/apache/logging/log4j/core/lookup/JndiLookup (java.net.URLClassLoader@...)`". Tutte le stringhe di log che fanno riferimento al mappatore jndi nell'output Log4J2 saranno sostituite con "`Patched JndiLookup::lookup()`".
> Indipendentemente da questa modifica, incoraggiamo fortemente tutti i clienti le cui funzioni includono Log4j2 ad aggiornare l'ultima versione. In particolare, i clienti che utilizzano la libreria aws-lambda-java-log 4j2 nelle proprie funzioni devono eseguire l'aggiornamento alla versione 1.5.0 (o successiva) e ridistribuire le proprie funzioni. Questa versione aggiorna le dipendenze dell'utility Log4j2 sottostanti alla versione 2.17.0 (o successiva). Il binario aws-

lambda-java-log 4j2 aggiornato è disponibile nel repository Maven e il suo codice sorgente è disponibile in Github.

Infine, tieni presente che le librerie relative a aws-lambda-java-log4j (v1.0.0 o 1.0.1) non devono essere utilizzate in nessuna circostanza. Queste librerie sono correlate alla versione 1.x di log4j, che ha raggiunto il fine vita nel 2015. Le librerie non sono supportate, non sono gestite, non sono corredate di patch e presentano vulnerabilità di sicurezza note.

Per personalizzare l'output dei log, supportare la registrazione durante i test unitari e registrare le chiamate AWS SDK, utilizzate Apache Log4j2 con SLF4 J. Log4j è una libreria di registrazione per programmi Java che consente di configurare i livelli di log e utilizzare le librerie di appender. SLF4J è una libreria di facciata che consente di modificare la libreria utilizzata senza modificare il codice della funzione.

Per aggiungere l'ID della richiesta ai log della funzione, usa l'appender nella libreria aws-lambda-java-log4j2.

Example src/main/resources/log4j2.xml — Configurazione di Appender

```
<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>
```

Puoi decidere di configurare gli output dei log di Log4j2 in testo normale o JSON specificando un layout sotto i tag `<LambdaTextFormat>` e `<LambdaJSONFormat>`.

Con questa configurazione, in modalità di testo, ogni riga è preceduta da data, ora, ID richiesta, livello di log e nome della classe. In modalità JSON, viene utilizzato `<JsonTemplateLayout>` con una configurazione fornita insieme alla libreria `aws-lambda-java-log4j2`.

SLF4J è una libreria di facciate per la registrazione del codice Java. Nel codice della funzione, si utilizza SLF4 J logger factory per recuperare un logger con metodi per i livelli di registro come e. `info()` `warn()` Nella configurazione di build, includi la libreria di registrazione e l'adattatore SLF4 J nel classpath. Modificando le librerie nella configurazione di build, è possibile modificare il tipo di logger senza modificare il codice della funzione. SLF4J è necessario per acquisire i log dall'SDK for Java.

Nel codice di esempio seguente, la classe handler utilizza SLF4 J per recuperare un logger.

Example src/main/java/example/HandlerS3.java — Registrazione con J SLF4

```
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";
    }
}
```

Questo codice genera output log simili al seguente:

Example Formato dei 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/"
}
```

La configurazione di build richiede le dipendenze di runtime dall'appender Lambda e dall'adattatore J SLF4 e le dipendenze di implementazione su Log4j2.

Example build.gradle – Registrazione delle dipendenze

```
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,)',
    ...
}
```

Quando esegui localmente il codice per i test, l'oggetto contestuale con il logger Lambda non è disponibile e non esiste alcun ID richiesta che possa essere utilizzato dall'appender Lambda. Per configurazioni di test di esempio, consulta le applicazioni di esempio nella sezione successiva.

Utilizzo di altri strumenti di registrazione e librerie

Powertools for AWS Lambda (Java) è un toolkit per sviluppatori che implementa le migliori pratiche Serverless e aumenta la velocità degli sviluppatori. L'utilità di registrazione fornisce un logger ottimizzato per Lambda che include informazioni aggiuntive sul contesto delle funzioni in tutte le funzioni con output strutturato come JSON. Utilizza l'utility per eseguire le seguenti operazioni:

- Acquisizione di campi essenziali dal contesto Lambda, avvii a freddo e output di registrazione della struttura come JSON
- Registrazione degli eventi di chiamata Lambda quando richiesto (disabilitata per impostazione predefinita)
- Stampa di tutti i log solo per una percentuale di chiamate tramite campionamento dei log (disabilitata per impostazione predefinita)
- Aggiunta di chiavi supplementari al log strutturato in qualsiasi momento
- Utilizzo di un formattatore di log personalizzato (Bring Your Own Formatter) per generare i log in una struttura compatibile con Logging RFC dell'organizzazione

Utilizzo di Powertools per AWS Lambda (Java) e per la registrazione strutturata AWS SAM

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Java Hello World di esempio con i moduli Powertools for AWS Lambda (Java) integrati utilizzando. AWS SAM Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio `hello world`.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Java 11
- AWS CLI versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS SAM

1. Inizializza l'applicazione utilizzando il modello Java Hello World.

```
sam init --app-template hello-world-powertools-java --name sam-app --package-type
 Zip --runtime java11 --no-tracing
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi `Enter`.

> **Note**
>
> Perché HelloWorldFunction potrebbe non avere un'autorizzazione definita, va bene? , assicurati di entrarey.

5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Richiama l'endpoint dell'API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere i log per la funzione, esegui sam logs. Per ulteriori informazioni, consulta l'argomento relativo all'utilizzo dei log nella Guida per sviluppatori AWS Serverless Application Model .

```
sam logs --stack-name sam-app
```

L'output del log ha la struttura seguente:

```
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. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

Gestione della conservazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, elimina il gruppo di log o configura un periodo di conservazione dopo il quale i log CloudWatch vengono eliminati automaticamente. Per configurare la conservazione dei log, aggiungi quanto segue al tuo modello: AWS SAM

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor del codice incorporato, troverai i log nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l'output del log nella sezione Dettagli.

Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (***your-function-name***/aws/lambda/).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Viene visualizzato un flusso di log quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire le chiamate simultanee. Per trovare i log per una chiamata specifica, consigliamo di strumentare la funzione con. AWS X-Ray X-Ray registra i dettagli sulla richiesta e il flusso di log nella traccia.

Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della AWS CLI versione 2.

È possibile utilizzare AWS CLI per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'`base64` utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `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
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
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
```

L'utilità `base64` è disponibile su Linux, macOS e Ubuntu su Windows. Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

Example Script get-logs.sh

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza `sed` per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.

L'cli-binary-formatopzione è obbligatoria se utilizzi la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 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 (solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il seguente comando per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
    "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"
}
```

Eliminazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, eliminare il gruppo di log o configurare un periodo di conservazione trascorso il quale i log vengono eliminati automaticamente.

Codice di registrazione dei log di esempio

L' GitHub archivio di questa guida include applicazioni di esempio che dimostrano l'uso di varie configurazioni di registrazione. Ogni applicazione di esempio include script per facilitare la distribuzione e la pulizia, un AWS SAM modello e risorse di supporto.

Applicazioni Lambda di esempio in Java

- example-java: una funzione Java che dimostra come utilizzare Lambda per elaborare gli ordini. Questa funzione illustra come definire e deserializzare un oggetto evento di input personalizzato, utilizzare l' AWS SDK e registrare l'output.
- java-basic: una raccolta di funzioni Java minimali con unit test e configurazione della registrazione dei log delle variabili.
- java-events: una raccolta di funzioni Java che contengono codice skeleton per la gestione degli eventi di vari servizi, ad esempio Gateway Amazon API, Amazon SQS e Amazon Kinesis. Queste funzioni utilizzano la versione più recente della aws-lambda-java-eventslibreria (3.0.0 e successive). Questi esempi non richiedono l' AWS SDK come dipendenza.
- s3-java – Una funzione Java che elabora gli eventi di notifica da Amazon S3 e utilizza la Java Class Library (JCL) per creare anteprime dai file di immagine caricati.
- layer-java: una funzione Java che illustra come utilizzare un livello Lambda per impacchettare dipendenze separate dal codice della funzione principale.

L'applicazione `java-basic` di esempio mostra una configurazione di registrazione minima che supporta i test di registrazione. Il codice del gestore utilizza il logger `LambdaLogger` fornito dall'oggetto contestuale. Per i test, l'applicazione utilizza una classe `TestLogger` personalizzata che implementa l'interfaccia `LambdaLogger` con un logger Log4j2. Utilizza SLF4 J come facciata per garantire la compatibilità con l' AWS SDK. Le librerie di logging sono escluse dall'output di compilazione per mantenere il pacchetto di implementazione di dimensioni ridotte.

Strumentazione del codice Java in AWS Lambda

Lambda si integra con AWS X-Ray per aiutarti a tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzare X-Ray per tracciare una richiesta mentre attraversa le risorse nell'applicazione, che possono includere funzioni Lambda e altri servizi AWS .

Per inviare dati di tracciamento a X-Ray, è possibile utilizzare una delle due librerie SDK:

- AWS Distro for OpenTelemetry (ADOT): una distribuzione sicura, pronta per la produzione e supportata dell'SDK (). AWS OpenTelemetry OTel
- SDK AWS X-Ray per Java: un SDK per generare e inviare i dati di traccia su X-Ray.
- Powertools for AWS Lambda (Java): un toolkit per sviluppatori per implementare le migliori pratiche Serverless e aumentare la velocità degli sviluppatori.

Ciascuno di essi SDKs offre modi per inviare i dati di telemetria al servizio X-Ray. Puoi quindi utilizzare X-Ray per visualizzare, filtrare e analizzare le metriche delle prestazioni dell'applicazione per identificare i problemi e le opportunità di ottimizzazione.

> **Important**
>
> X-Ray e Powertools per AWS Lambda SDKs fanno parte di una soluzione di strumentazione strettamente integrata offerta da. AWS I livelli Lambda ADOT fanno parte di uno standard di settore per la strumentazione di tracciamento che in generale raccoglie più dati, ma potrebbero non essere adatti a tutti i casi d'uso. È possibile implementare il end-to-end tracciamento in X-Ray utilizzando entrambe le soluzioni. Per saperne di più sulla scelta tra di esse, consulta Scelta tra AWS Distro for Open Telemetry e X-Ray. SDKs

Sections

- Utilizzo di Powertools per (Java) e per il AWS Lambda tracciamento AWS SAM
- Utilizzo di Powertools per AWS Lambda (Java) e AWS CDK per il tracciamento
- Utilizzo di ADOT per strumentare le funzioni Java
- Utilizzo dell'SDK X-Ray per strumentare le funzioni Java
- Attivazione del tracciamento con la console Lambda
- Attivazione del tracciamento con l'API Lambda

- Attivazione del tracciamento con CloudFormation
- Interpretazione di una traccia X-Ray
- Memorizzazione delle dipendenze di runtime in un livello (SDK X-Ray)
- Tracciamento X-Ray in applicazioni di esempio (SDK X-Ray)

Utilizzo di Powertools per (Java) e per il AWS Lambda tracciamento AWS SAM

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Java Hello World di esempio con i moduli Powertools for AWS Lambda (Java) integrati utilizzando il. AWS SAM Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio `hello world`.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Java 11 o versione successiva
- AWS CLI versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS SAM

1. Inizializza l'applicazione utilizzando il modello Java Hello World.

```
sam init --app-template hello-world-powertools-java --name sam-app --package-type
 Zip --runtime java11 --no-tracing
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi `Enter`.

> **Note**
>
> Perché HelloWorldFunction potrebbe non avere un'autorizzazione definita, va bene? , assicurati di entrarey.

5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Richiama l'endpoint dell'API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere le tracce per la funzione, esegui sam traces.

```
sam traces
```

L'output della traccia ha il seguente aspetto:

```
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. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

Utilizzo di Powertools per AWS Lambda (Java) e AWS CDK per il tracciamento

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Java Hello World di esempio con i moduli Powertools for AWS Lambda (Java) integrati utilizzando. AWS CDK Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio hello world.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- Java 11 o versione successiva

- AWS CLI versione 2
- AWS CDK versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS CDK

1. Crea una directory di progetto per la nuova applicazione.

```
mkdir hello-world
cd hello-world
```

2. Inizializza l'app.

```
cdk init app --language java
```

3. Crea un progetto maven utilizzando il comando seguente:

```
mkdir app
cd app
mvn archetype:generate -DgroupId=helloworld -DartifactId=Function -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

4. Apri `pom.xml` nella directory `hello-world\app\Function` e sostituisci il codice esistente con il codice seguente, che include dipendenze e plug-in maven per 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. Crea la directory `hello-world\app\src\main\resource` e crea `log4j.xml` per la configurazione del log.

```
mkdir -p src/main/resource
cd src/main/resource
touch log4j.xml
```

6. Apri `log4j.xml` e aggiungi il seguente codice.

```
<?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. Apri `App.java` dalla directory `hello-world\app\Function\src\main\java\helloworld` e sostituisci il codice esistente con il codice seguente. Questo è il codice per la funzione 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. Apri `HelloWorldStack.java` dalla directory `hello-world\src\main\java\com\myorg` e sostituisci il codice esistente con il codice seguente. Questo codice utilizza Lambda Constructor e ApiGatewayv2 Constructor per creare un'API REST e una funzione Lambda.

```
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. Apri `pom.xml` dalla directory `hello-world` e sostituisci il codice esistente con il codice seguente.

```
<?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. Assicurati di essere nella directory `hello-world` e implementa l'applicazione.

```
cdk deploy
```

11. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name HelloWorldStack --query
 'Stacks[0].Outputs[?OutputKey==`HttpApi`].OutputValue' --output text
```

12. Richiama l'endpoint dell'API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

    In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

13. Per ottenere le tracce per la funzione, esegui sam traces.

```
sam traces
```

L'output della traccia ha il seguente aspetto:

```
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. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
cdk destroy
```

Utilizzo di ADOT per strumentare le funzioni Java

ADOT fornisce layer Lambda completamente gestiti che racchiudono tutto il necessario per raccogliere dati di telemetria utilizzando l'SDK. OTel Usando questo livello, è possibile strumentare le funzioni Lambda senza dover modificare alcun codice funzione. Puoi anche configurare il tuo layer per eseguire l'inizializzazione personalizzata di. OTel Per ulteriori informazioni, consulta la sezione relativa alla configurazione personalizzata per ADOT Collector su Lambda nella documentazione di ADOT.

Per i runtime Java, è possibile scegliere tra due livelli da utilizzare:

- AWS layer Lambda gestito per ADOT Java (Auto-instrumentation Agent): questo livello trasforma automaticamente il codice della funzione all'avvio per raccogliere dati di tracciamento. Per istruzioni dettagliate su come utilizzare questo layer insieme all'agente Java ADOT, consulta AWS Distro for Lambda OpenTelemetry Support for Java (Auto-instrumentation Agent) nella documentazione ADOT.
- AWS layer Lambda gestito per ADOT Java — Questo livello fornisce anche strumentazione integrata per le funzioni Lambda, ma richiede alcune modifiche manuali al codice per inizializzare l'SDK. OTel Per istruzioni dettagliate su come utilizzare questo layer, consulta AWS Distro for OpenTelemetry Lambda Support for Java nella documentazione ADOT.

Utilizzo dell'SDK X-Ray per strumentare le funzioni Java

Per registrare dati sulle chiamate effettuate dalla funzione ad altre risorse e servizi nell'applicazione, è possibile aggiungere l'SDK X-Ray per Java alla configurazione di compilazione. L'esempio seguente mostra una configurazione di build di Gradle che include le librerie che attivano la strumentazione automatica dei client. AWS SDK for Java 2.x

Example build.gradle – Tracciamento delle dipendenze

```
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'
    ...
}
```

Dopo aver aggiunto le dipendenze corrette e aver apportato le modifiche necessarie al codice, attivare il tracciamento nella configurazione della funzione tramite la console Lambda o l'API.

Attivazione del tracciamento con la console Lambda

Per attivare il tracciamento attivo sulla funzione Lambda con la console, attenersi alla seguente procedura:

Per attivare il tracciamento attivo

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione) e quindi Monitoring and operations tools (Strumenti di monitoraggio e operazioni).
4. In Strumenti di monitoraggio aggiuntivi scegli Modifica.
5. In CloudWatch Application Signals e AWS X-Ray, scegli Enable for Lambda service trace.
6. Scegli Save (Salva).

Attivazione del tracciamento con l'API Lambda

Configura il tracciamento sulla tua funzione Lambda con AWS o SDK, utilizza AWS CLI le seguenti operazioni API:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Il AWS CLI comando di esempio seguente abilita il tracciamento attivo su una funzione denominata my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

La modalità di tracciamento fa parte della configurazione specifica della versione quando si pubblica una versione della funzione. Non è possibile modificare la modalità di tracciamento in una versione pubblicata.

Attivazione del tracciamento con CloudFormation

Per attivare il tracciamento su una `AWS::Lambda::Function` risorsa in un CloudFormation modello, utilizzate la proprietà. `TracingConfig`

Example function-inline.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Per una `AWS::Serverless::Function` risorsa AWS Serverless Application Model (AWS SAM), utilizzate la `Tracing` proprietà.

Example template.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretazione di una traccia X-Ray

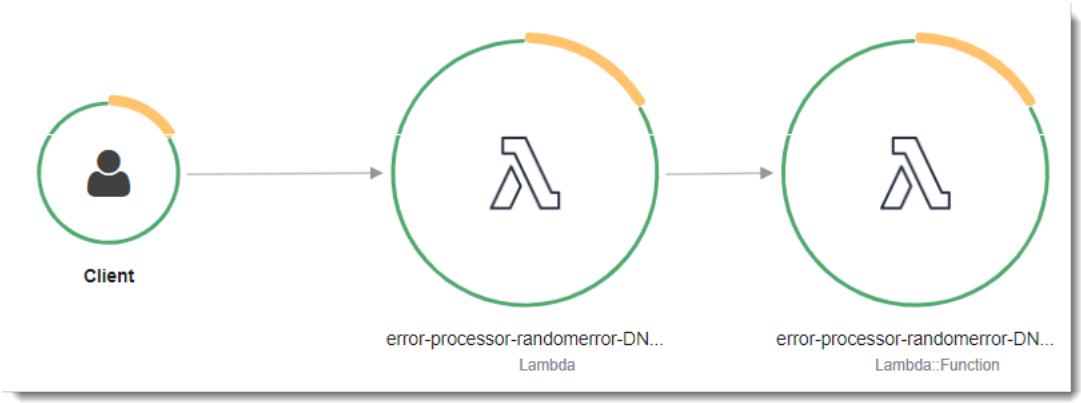
La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al ruolo di esecuzione della funzione. Altrimenti, aggiungete la AWSXRayDaemonWriteAccesspolitica al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico dei servizi X-Ray mostra informazioni sull'applicazione e tutti i suoi componenti. Il seguente esempio mostra un'applicazione con due funzioni. La funzione principale elabora gli eventi e talvolta restituisce errori. La seconda funzione in alto elabora gli errori che compaiono nel gruppo di log della prima e utilizza l' AWS SDK per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon Logs. CloudWatch

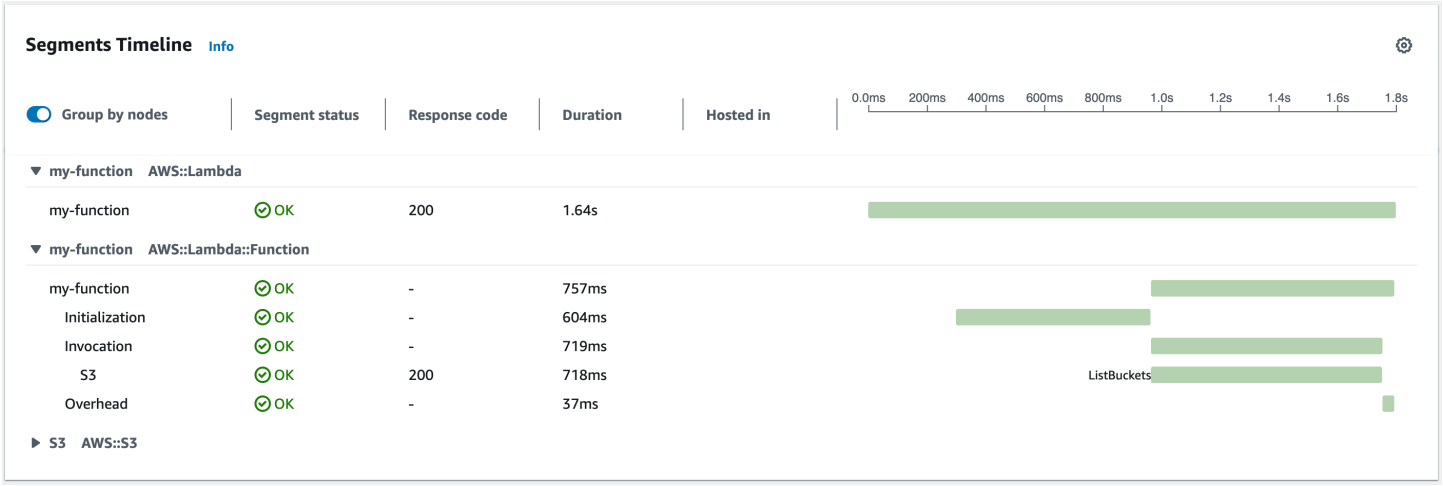


X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non è possibile configurare la frequenza di campionamento di X-Ray per le funzioni.

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda. L'esempio seguente mostra una traccia con questi 2 segmenti. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.



Questo esempio espande il segmento `AWS::Lambda::Function` per visualizzare i relativi tre sottosegmenti.

> **Note**
>
> AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. La traccia di esempio mostrata qui illustra il segmento di funzione vecchio stile. Le differenze tra i segmenti vecchio e nuovo stile sono descritte nei paragrafi seguenti.
> Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regioni, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia di nuovo formato.

Il segmento di funzioni vecchio stile contiene i seguenti sottosegmenti:

- Inizializzazione – Rappresenta il tempo trascorso a caricare la funzione e ad eseguire il codice di inizializzazione. Questo sottosegmento viene visualizzato solo per il primo evento che viene elaborato da ogni istanza della funzione.

- Chiamata: rappresenta il tempo impiegato per eseguire il codice del gestore.
- Overhead: rappresenta il tempo impiegato dal runtime Lambda per prepararsi a gestire l'evento successivo.

Il segmento di funzione di nuovo stile non contiene un sottosegmento `Invocation`. I sottosegmenti dei clienti sono invece collegati direttamente al segmento di funzioni. Per ulteriori informazioni sulla struttura dei segmenti di funzioni vecchio e nuovo stile, consulta the section called “Informazioni sui monitoraggi di X-Ray”.

> **Note**
>
> Le funzioni Lambda SnapStart includono anche un sottosegmento `Restore`. Il `Restore` sottosegmento mostra il tempo impiegato da Lambda per ripristinare un'istantanea, caricare il runtime ed eseguire eventuali hook di runtime successivi al ripristino. Il processo di ripristino degli snapshot può includere il tempo dedicato ad attività esterne alla MicroVM. Questa volta è riportato nel segmento secondario `Restore`. Non ti viene addebitato il tempo trascorso fuori dalla microVM per il ripristino di una snapshot.

È inoltre possibile strumentare i client HTTP, registrare query SQL e creare segmenti secondari personalizzati con annotazioni e metadati. Per ulteriori informazioni, consulta la sezione SDK AWS X-Ray per Java nella Guida per gli sviluppatori di AWS X-Ray .

> **Prezzi**
>
> Puoi utilizzare il tracciamento X-Ray gratuitamente ogni mese fino a un determinato limite come parte del AWS piano gratuito. Oltre la soglia, X-Ray addebita lo storage di traccia e il recupero. Per ulteriori informazioni, consultare Prezzi di AWS X-Ray.

Memorizzazione delle dipendenze di runtime in un livello (SDK X-Ray)

Se utilizzate X-Ray SDK per strumentare i client AWS SDK del codice della funzione, il pacchetto di distribuzione può diventare piuttosto grande. Per evitare di caricare dipendenze di runtime ogni volta che si aggiorna il codice della funzione, includere l'SDK X-Ray in un livello Lambda.

L'esempio seguente mostra una risorsa `AWS::Serverless::LayerVersion` che memorizza l'SDK AWS SDK per Java e X-Ray per Java.

Example template.yml – Livello delle dipendenze

```
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
```

Con questa configurazione, si aggiorna il livello della libreria solo se si modificano le dipendenze di runtime. Poiché il pacchetto di implementazione della funzione contiene solo il codice, questo può aiutare a ridurre i tempi di caricamento.

La creazione di un layer per le dipendenze richiede modifiche alla configurazione di compilazione per generare l'archivio dei layer prima della distribuzione. Per un esempio funzionante, vedete l'applicazione di esempio java-basic su. GitHub

Tracciamento X-Ray in applicazioni di esempio (SDK X-Ray)

L' GitHub archivio di questa guida include applicazioni di esempio che dimostrano l'uso del tracciamento a raggi X. Ogni applicazione di esempio include script per facilitare l'implementazione e la pulizia, un AWS SAM modello e risorse di supporto.

Applicazioni Lambda di esempio in Java

- example-java: una funzione Java che dimostra come utilizzare Lambda per elaborare gli ordini. Questa funzione illustra come definire e deserializzare un oggetto evento di input personalizzato, utilizzare l' AWS SDK e registrare l'output.
- java-basic: una raccolta di funzioni Java minimali con unit test e configurazione della registrazione dei log delle variabili.

- java-events: una raccolta di funzioni Java che contengono codice skeleton per la gestione degli eventi di vari servizi, ad esempio Gateway Amazon API, Amazon SQS e Amazon Kinesis. Queste funzioni utilizzano la versione più recente della aws-lambda-java-eventslibreria (3.0.0 e successive). Questi esempi non richiedono l' AWS SDK come dipendenza.
- s3-java – Una funzione Java che elabora gli eventi di notifica da Amazon S3 e utilizza la Java Class Library (JCL) per creare anteprime dai file di immagine caricati.
- layer-java: una funzione Java che illustra come utilizzare un livello Lambda per impacchettare dipendenze separate dal codice della funzione principale.

Tutte le applicazioni di esempio hanno il tracciamento attivo abilitato per le funzioni Lambda. Ad esempio, l'`s3-java`applicazione mostra la strumentazione automatica dei AWS SDK for Java 2.x client, la gestione dei segmenti per i test, i sottosegmenti personalizzati e l'uso dei livelli Lambda per archiviare le dipendenze di runtime.

Esempi di applicazioni Java per AWS Lambda

L' GitHub archivio di questa guida fornisce applicazioni di esempio che dimostrano l'uso di Java in AWS Lambda. Ogni applicazione di esempio include script per facilitare la distribuzione e la pulizia, un CloudFormation modello e risorse di supporto.

Applicazioni Lambda di esempio in Java

- example-java: una funzione Java che dimostra come utilizzare Lambda per elaborare gli ordini. Questa funzione illustra come definire e deserializzare un oggetto evento di input personalizzato, utilizzare l' AWS SDK e registrare l'output.
- java-basic: una raccolta di funzioni Java minimali con unit test e configurazione della registrazione dei log delle variabili.
- java-events: una raccolta di funzioni Java che contengono codice skeleton per la gestione degli eventi di vari servizi, ad esempio Gateway Amazon API, Amazon SQS e Amazon Kinesis. Queste funzioni utilizzano la versione più recente della aws-lambda-java-eventslibreria (3.0.0 e successive). Questi esempi non richiedono l' AWS SDK come dipendenza.
- s3-java – Una funzione Java che elabora gli eventi di notifica da Amazon S3 e utilizza la Java Class Library (JCL) per creare anteprime dai file di immagine caricati.
- layer-java: una funzione Java che illustra come utilizzare un livello Lambda per impacchettare dipendenze separate dal codice della funzione principale.

Esecuzione dei framework Java più diffusi su Lambda

- spring-cloud-function-samples— Un esempio tratto da Spring che mostra come utilizzare il framework Spring Cloud Function per creare funzioni AWS Lambda.
- Demo dell'applicazione Spring Boot senza server: un esempio che mostra come configurare una tipica applicazione Spring Boot in un runtime Java gestito con e senza SnapStart, o come immagine nativa GraalVM con un runtime personalizzato.
- Demo dell'applicazione Serverless Micronaut: un esempio che mostra come utilizzare Micronaut in un runtime Java gestito con e senza SnapStart, o come immagine nativa GraalVM con un runtime personalizzato. Scopri di più nelle guide Micronaut/Lambda.
- Demo dell'applicazione Quarkus senza server: un esempio che mostra come utilizzare Quarkus in un runtime Java gestito con e senza, o come immagine nativa GraalVM con un runtime personalizzato. SnapStart Scopri di più nella guida Quarkus/Lambda e nella guida Quarkus/. SnapStart

Se non hai mai utilizzato le funzioni Lambda in Java, inizia con gli esempi `java-basic`. Per iniziare con le origini eventi Lambda, consulta gli esempi `java-events`. Entrambi questi set di esempi mostrano l'uso delle librerie Java, delle variabili di ambiente, dell'SDK e dell'SDK di Lambda. AWS AWS X-Ray Questi esempi richiedono una configurazione minima e possono essere implementati dalla riga di comando in meno di un minuto.

Compilazione di funzioni Lambda con Go

Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato nativamente in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un runtime solo per il sistema operativo (la famiglia di runtime `provided`) per distribuire le funzioni Go in Lambda.

Argomenti

- Supporto per il runtime Go
- Strumenti e librerie
- Definisci i gestori di funzioni Lambda in Go
- Utilizzo dell'oggetto contestuale Lambda per recuperare le informazioni sulla funzione Go
- Distribuisci funzioni Lambda per Go con gli archivi di file .zip
- Distribuzione delle funzioni Go Lambda con immagini di container
- Utilizzo dei livelli per le funzioni Lambda in Go
- Registrare e monitorare le funzioni Lambda con Go
- Strumentazione del codice Go in AWS Lambda

Supporto per il runtime Go

Il runtime gestito Go 1.x per Lambda è obsoleto. Se hai funzioni che utilizzano il runtime Go 1.x, devi eseguire la migrazione delle tue funzioni a `provided.al2023` o `provided.al2`. I `provided.al2` runtime `provided.al2023` and offrono diversi vantaggi`go1.x`, tra cui il supporto per l'architettura arm64 (processori AWS Graviton2), binari più piccoli e tempi di invoke leggermente più rapidi.

Per questa migrazione, non sono necessarie modifiche al codice. Le uniche modifiche richieste riguardano la modalità di creazione del pacchetto di implementazione e il runtime utilizzato per creare la funzione. Per ulteriori informazioni, consulta la sezione Migrazione AWS Lambda delle funzioni dal runtime Go1.x al runtime personalizzato su Amazon Linux 2 sul blog di Compute AWS .

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Runtime solo per il sistema operativo	`provided.al2023`	Amazon Linux 2023	30 giugno 2029	31 luglio 2029	31 agosto 2029
Runtime solo per il sistema operativo	`provided.al2`	Amazon Linux 2	31 luglio 2026	31 agosto 2026	30 settembre 2026

Strumenti e librerie

Lambda fornisce i seguenti strumenti e librerie per il runtime Go:

- AWS SDK per Go v2: L' AWS SDK ufficiale per il linguaggio di programmazione Go.
- github. com/aws/aws-lambda-go/lambda: L'implementazione del modello di programmazione Lambda per Go. Questo pacchetto viene utilizzato da AWS Lambda per richiamare il gestore.
- github. com/aws/aws-lambda-go/lambdacontext: Aiutanti per accedere alle informazioni di contesto dall'oggetto context.
- github. com/aws/aws-lambda-go/events: Questa libreria fornisce definizioni dei tipi per le integrazioni di sorgenti di eventi comuni.
- github. com/aws/aws-lambda-go/cmd/build-lambda-zip: questo strumento può essere utilizzato per creare un archivio di file.zip su Windows.

Per ulteriori informazioni, vedere on. aws-lambda-go GitHub

Lambda fornisce le seguenti applicazioni di esempio per il runtime di Go:

Applicazioni Lambda di esempio in Go

- go-al2: una funzione hello world che restituisce l'indirizzo IP pubblico. Questa app utilizza il runtime personalizzato `provided.al2`.
- blank-go — Una funzione Go che mostra l'uso delle librerie Go di Lambda, la registrazione, le variabili di ambiente e l'SDK. AWS Questa app utilizza il runtime `go1.x`.

Definisci i gestori di funzioni Lambda in Go

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Questa pagina descrive come lavorare con i gestori di funzioni Lambda in Go, inclusa la configurazione del progetto, le convenzioni di denominazione e le migliori pratiche. Questa pagina include anche un esempio di funzione Go Lambda che raccoglie informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon Simple Storage Service (S3). Per informazioni su come distribuire una funzione dopo averla scritta, consulta o. the section called “Implementazione di archivi di file .zip” the section called “Implementazione di immagini di container”

Argomenti

- Configurazione di Go Handler
- Esempio di codice della funzione Lambda
- Convenzioni di denominazione dei gestori
- Definizione e accesso all'oggetto evento di input
- Accesso e utilizzo dell'oggetto contestuale Lambda
- Firme dell'handler valide per gli handler Go
- Utilizzo della AWS SDK per Go versione v2 nell'handler
- Accesso alle variabili d'ambiente
- Utilizzo dello stato globale
- Procedure consigliate di codice per le funzioni Go Lambda

Configurazione di Go Handler

Una funzione Lambda scritta in Go viene creata come eseguibile di Go. È possibile inizializzare un progetto di funzione Go Lambda nello stesso modo in cui inizializzi qualsiasi altro progetto Go utilizzando il seguente comando: `go mod init`

```
go mod init example-go
```

Ecco il nome del `example-go` modulo. Puoi sostituire con un valore qualsiasi. Questo comando inizializza il progetto e genera il `go.mod` file che elenca le dipendenze del progetto.

Utilizzate il `go get` comando per aggiungere eventuali dipendenze esterne al progetto. Nel codice della funzione Lambda è necessario includere il pacchetto github.com/aws/aws-lambda-go/lambda che implementa il modello di programmazione Lambda per Go. Installa il pacchetto con il seguente comando `go get`:

```
go get github.com/aws/aws-lambda-go
```

Il codice della funzione dovrebbe risiedere in un file Go. In questo esempio viene assegnato un nome a questo file`main.go`. In questo file, implementa la logica della funzione principale in un metodo di gestione, oltre a una funzione `main()` che chiama questo handler.

Esempio di codice della funzione Lambda

Il seguente esempio di codice della funzione Go Lambda raccoglie le informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon S3.

Example Funzione Lambda **`main.go`**

```
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)
}
```

Questo file `main.go` contiene le sezioni seguenti:

- `package main`: in Go, il pacchetto contenente la funzione `func main()` deve essere sempre denominato `main`.
- Blocco `import`: utilizza questo blocco per includere le librerie richieste dalla funzione Lambda.
- `type Order struct {}`block: definisce la forma dell'evento di input previsto in questa struttura Go.
- `var ()`block: usa questo blocco per definire tutte le variabili globali che utilizzerai nella tua funzione Lambda.
- `func init() {}`: includi in questo metodo qualsiasi codice che desideri che Lambda esegua durante la fase di inizializzazione. `init()`
- `func uploadReceiptToS3(...) {}`: questo è un metodo helper a cui fa riferimento il metodo dell'handler principale `handleRequest`.
- `func handleRequest(ctx context.Context, event json.RawMessage) error {}`: questo è il metodo dell'handler principale, che contiene la logica principale dell'applicazione.
- `func main() {}`: questo è un punto di ingresso obbligatorio per il tuo handler Lambda. L'argomento del metodo `lambda.Start()` è il metodo principale dell'handler.

Affinché questa funzione funzioni correttamente, il suo ruolo di esecuzione deve consentire l'`s3:PutObject`azione. Inoltre, assicuratevi di definire la variabile di `RECEIPT_BUCKET` ambiente. Dopo una chiamata riuscita, il bucket Amazon S3 dovrebbe contenere un file di ricevuta.

Convenzioni di denominazione dei gestori

Per le funzioni Lambda in Go, puoi usare qualsiasi nome per l'handler. In questo esempio, il nome del metodo dell'handler è `handleRequest`. Per fare riferimento al valore del gestore nel codice, puoi usare la variabile di ambiente `_HANDLER`.

Per le funzioni Go implementate mediante un pacchetto di implementazione .zip, il file eseguibile che contiene il codice della funzione deve essere denominato `bootstrap`. Il file `bootstrap` deve trovarsi nella posizione root del file .zip. Per le funzioni Go implementate mediante una immagine del container, per il file eseguibile è possibile usare qualsiasi nome.

Definizione e accesso all'oggetto evento di input

JSON è il formato di input più comune e standard per le funzioni Lambda. In questo esempio, la funzione prevede un input simile a quanto segue:

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

Quando si utilizzano le funzioni Lambda in Go, è possibile definire la forma dell'evento di input previsto come struttura Go. In questo esempio, definiamo una struttura per rappresentare un`Order`:

```
type Order struct {
    OrderID string  `json:"order_id"`
    Amount  float64 `json:"amount"`
    Item    string  `json:"item"`
}
```

Questa struttura corrisponde alla forma di input prevista. Dopo aver definito la struttura, puoi scrivere una firma dell'handler che includa un tipo JSON generico compatibile con la libreria standard encoding/json. Puoi quindi deserializzarlo nella tua struttura usando la funzione func Unmarshal. Ciò è illustrato nelle prime righe dell'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
    ...
}
```

Dopo questa deserializzazione, è possibile accedere ai campi della variabile. `order` Ad esempio, `order.OrderID` recupera il valore di `"order_id"` dall'input originale.

> **Note**
>
> Il `encoding/json` pacchetto può accedere solo ai campi esportati. Affinché siano esportati, i nomi dei campi nella struttura dell'evento devono avere l'iniziale maiuscola.

Accesso e utilizzo dell'oggetto contestuale Lambda

L'oggetto contesto: contiene informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. In questo esempio, abbiamo dichiarato questa variabile come `ctx` nella firma dell'handler:

```
func handleRequest(ctx context.Context, event json.RawMessage) error {
    ...
}
```

L'input `ctx context.Context` è un argomento facoltativo nell'handler della funzione. Per ulteriori informazioni sulle firme dell'handler accettate, consulta the section called “Firme dell'handler valide per gli handler Go”.

Se si effettuano chiamate ad altri servizi utilizzando l'AWSSDK, l'oggetto context è necessario in alcune aree chiave. Ad esempio, per inizializzare correttamente i client SDK, è possibile caricare la configurazione AWS SDK corretta utilizzando l'oggetto di contesto come segue:

```
// Load AWS SDK configuration using the default credential provider chain
    cfg, err := config.LoadDefaultConfig(ctx)
```

Le stesse chiamate SDK possono richiedere l'oggetto context come input. Ad esempio, la `s3Client.PutObject` chiamata accetta l'oggetto context come primo argomento:

```
// Upload the receipt to S3
    _, err = s3Client.PutObject(ctx, &s3.PutObjectInput{
        ...
    })
```

Oltre alle richieste AWS SDK, puoi anche utilizzare l'oggetto context per il monitoraggio delle funzioni. Per ulteriori informazioni sulla copia di oggetti, consulta la sezione the section called “Contesto”.

Firme dell'handler valide per gli handler Go

Durante la creazione di un gestore della funzione Lambda in Go sono disponibili diverse opzioni, ma è necessario attenersi alle seguenti regole:

- Il gestore deve essere una funzione.
- Il gestore può richiedere da 0 a 2 argomenti. Nel caso di due argomenti, il primo argomento deve implementare `context.Context`.
- Il gestore può restituire da 0 a 2 argomenti. Nel caso di un singolo valore restituito, deve implementare `error`. Nel caso di due valori restituiti, il secondo valore deve implementare `error`.

Di seguito sono elencate le firme del gestore valide. `TIn` e `TOut` rappresentano le tipologie compatibili con la libreria standard encoding/json. Per ulteriori informazioni su come deserializzare queste tipologie, consultare func Unmarshal.

- `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)`

Utilizzo della AWS SDK per Go versione v2 nell'handler

Spesso, utilizzerai le funzioni Lambda per interagire o aggiornare altre AWS risorse. Il modo più semplice per interfacciarsi con queste risorse è usare la AWS SDK per Go v2.

> **Note**
>
> La AWS SDK per Go (v1) è in modalità manutenzione e terminerà il supporto il 31 luglio 2025. In futuro, si consiglia di utilizzare solo la AWS SDK per Go v2.

Per aggiungere dipendenze SDK alla tua funzione, usa il `go get` comando per i client SDK specifici di cui hai bisogno. Nel codice di esempio precedente, abbiamo usato la `config` libreria e la libreria. `s3` Aggiungi queste dipendenze eseguendo i seguenti comandi nella directory che contiene i file `go.mod` e `main.go` :

```
go get github.com/aws/aws-sdk-go-v2/config
go get github.com/aws/aws-sdk-go-v2/service/s3
```

Quindi, importa le dipendenze di conseguenza nel blocco di importazione della tua funzione:

```
import (
    ...
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)
```

Quando utilizzi l'SDK nel tuo handler, configura i tuoi client con le impostazioni corrette. Il modo più semplice per farlo è utilizzare la catena di provider di credenziali predefinita. Questo esempio illustra un modo per caricare questa configurazione:

```
// 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
    }
```

Dopo aver caricato questa configurazione nella variabile `cfg`, è possibile passare la variabile nelle istanze del client. Il codice di esempio crea un'istanza di un client Amazon S3 come segue:

```
// Create an S3 client
    s3Client := s3.NewFromConfig(cfg)
```

In questo esempio, abbiamo inizializzato il nostro client Amazon S3 nella `init()` funzione per evitare di doverlo inizializzare ogni volta che richiamiamo la nostra funzione. Il problema è che nella `init()` funzione, Lambda non ha accesso all'oggetto context. Come soluzione alternativa, puoi inserire un segnaposto come `context.TODO()` durante la fase di inizializzazione. Successivamente, quando effettui una chiamata utilizzando il client, inserisci l'oggetto contestuale completo. Questa soluzione alternativa è descritta anche inthe section called “Utilizzo del contesto nelle inizializzazioni e nelle AWS chiamate dei client SDK”.

Dopo aver configurato e inizializzato il client SDK, puoi utilizzarlo per interagire con altri servizi. AWS Il codice di esempio richiama l'`PutObject`API Amazon S3 nel modo seguente:

```
_, err = s3Client.PutObject(ctx, &s3.PutObjectInput{
    Bucket: &bucketName,
    Key:    &key,
    Body:   strings.NewReader(receiptContent),
})
```

Accesso alle variabili d'ambiente

Nel codice dell'handler, puoi fare riferimento a qualsiasi variabile di ambiente utilizzando il metodo `os.Getenv()`. In questo esempio, facciamo riferimento alla variabile di `RECEIPT_BUCKET` ambiente definita utilizzando la seguente riga di codice:

```
// 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")
    }
```

Utilizzo dello stato globale

Per evitare di creare nuove risorse ogni volta che richiami la funzione, puoi dichiarare e modificare le variabili globali all'esterno del codice dell'handler della funzione Lambda. Queste variabili globali vengono definite in un blocco o in un'`var`istruzione. Inoltre, l'handler può dichiarare una funzione `init()` che viene eseguita durante la fase di inizializzazione. Il metodo `init` si comporta allo stesso modo in AWS Lambda come nei programmi Go standard.

Procedure consigliate di codice per le funzioni Go Lambda

Segui le linee guida riportate nell'elenco seguente per utilizzare le best practice di codifica durante la creazione delle funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. In questo modo è possibile creare una funzione di cui è più semplice eseguire l'unit test.
- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del contesto di esecuzione.
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. In questo modo viene ridotta la quantità di tempo necessaria per il download del pacchetto e per la relativa decompressione prima dell'invocazione.

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente,

utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedere Riutilizzo delle connessioni con Keep-Alive in Node.js.

Utilizzare le variabili di ambiente per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su `0` per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare API non documentate e non pubbliche nel codice della funzione Lambda. Per i tempi di esecuzione gestiti AWS Lambda, Lambda applica periodicamente aggiornamenti di sicurezza e funzionalità alle API interne di Lambda. Questi aggiornamenti API interni potrebbero essere incompatibili con le versioni precedenti, causando conseguenze indesiderate come errori di chiamata se la funzione ha una dipendenza su queste API non pubbliche. Consulta il riferimento all'API per un elenco di API disponibili pubblicamente.

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda?.

Utilizzo dell'oggetto contestuale Lambda per recuperare le informazioni sulla funzione Go

Questo oggetto fornisce i metodi e le proprietà con informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. Quando Lambda esegue la funzione, passa un oggetto Context al gestore. Per utilizzare l'oggetto context nel tuo handler, puoi facoltativamente dichiararlo come parametro di input per il tuo handler. L'oggetto contesto è necessario se desideri eseguire le operazioni indicate di seguito nell'handler:

- È necessario accedere a tutte le variabili, i metodi o le proprietà globali offerti dall'oggetto context. Questi metodi e proprietà sono utili per attività come determinare l'entità che ha richiamato la funzione o misurare il tempo di invocazione della funzione, come illustrato in. the section called “Accesso alle informazioni relative al contesto di invocazione”
- È necessario utilizzare il per effettuare chiamate AWS SDK per Go verso altri servizi. L'oggetto context è un parametro di input importante per la maggior parte di queste chiamate. Per ulteriori informazioni, consulta the section called “Utilizzo del contesto nelle inizializzazioni e nelle AWS chiamate dei client SDK”.

Argomenti

- Variabili supportate, metodi e proprietà nell'oggetto contesto
- Accesso alle informazioni relative al contesto di invocazione
- Utilizzo del contesto nelle inizializzazioni e nelle AWS chiamate dei client SDK

Variabili supportate, metodi e proprietà nell'oggetto contesto

La libreria contesto Lambda offre le variabili globali, i metodi e le proprietà indicate di seguito.

Variabili globali

- `FunctionName`: il nome della funzione Lambda.
- `FunctionVersion`: la versione della funzione.
- `MemoryLimitInMB`: la quantità di memoria allocata per la funzione.
- `LogGroupName`: il gruppo di log per la funzione.
- `LogStreamName`: il flusso di log per l'istanza della funzione.

Metodi del contesto

- `Deadline`: restituisce la data del timeout dell'esecuzione in millisecondi Unix.

Proprietà del contesto

- `InvokedFunctionArn`: l'ARN (Amazon Resource Name) utilizzato per richiamare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `AwsRequestID`: l'identificatore della richiesta di invocazione.
- `Identity`: (app per dispositivi mobili) Informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
- `ClientContext`: (app per dispositivi mobili) Contesto client fornito a Lambda dall'applicazione client.

Accesso alle informazioni relative al contesto di invocazione

Le funzioni Lambda hanno accesso ai metadata relativi al loro ambiente e alla richiesta di invocazione. È possibile accedervi mediante il Package context. Qualora il gestore includa `context.Context` come parametro, Lambda inserirà le informazioni relative alla funzione nella proprietà `Value` del contesto. È necessario importare la libreria `lambdacontext` per accedere ai contenuti dell'oggetto `context.Context`.

```
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)
```

```
}
```

Nell'esempio sopra riportato, `lc` è la variabile utilizzata per usufruire delle informazioni acquisite dall'oggetto contesto e `log.Print(lc.Identity.CognitoIdentityPoolID)` stampa le informazioni, in questo caso il CognitoPoolID.

L'esempio che segue illustra come utilizzare l'oggetto contesto per monitorare il tempo impiegato per il completamento della funzione Lambda. In tal modo è possibile analizzare le aspettative prestazionali e, se necessario modificare la funzione di conseguenza.

```
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)
}
```

Utilizzo del contesto nelle inizializzazioni e nelle AWS chiamate dei client SDK

Se l'handler deve utilizzare il per effettuare chiamate AWS SDK per Go ad altri servizi, includi l'oggetto context come input per l'handler. In AWS, è consigliabile passare l'oggetto del contesto nella maggior parte delle chiamate all'SDK AWS. Ad esempio, la `PutObject` chiamata Amazon S3 accetta l'oggetto context (`ctx`) come primo argomento:

```
// Upload an object to S3
    _, err = s3Client.PutObject(ctx, &s3.PutObjectInput{
        ...
    })
```

Per inizializzare correttamente i client SDK, puoi anche utilizzare l'oggetto del contesto per caricare la configurazione corretta prima di passare l'oggetto di configurazione al client:

```
// Load AWS SDK configuration using the default credential provider chain
    cfg, err := config.LoadDefaultConfig(ctx)
    ...
    s3Client = s3.NewFromConfig(cfg)
```

Se desideri inizializzare i client SDK al di fuori dell'handler principale (ad esempio durante la fase di inizializzazione), puoi passare un oggetto del contesto segnaposto:

```
func init() {
 // Initialize the S3 client outside of the handler, during the init phase
 cfg, err := config.LoadDefaultConfig(context.TODO())
 ...
 s3Client = s3.NewFromConfig(cfg)
}
```

Se inizializzi i client in questo modo, assicurati di inserire l'oggetto contestuale corretto nelle chiamate SDK dall'handler principale.

Distribuisci funzioni Lambda per Go con gli archivi di file .zip

Il codice della funzione AWS Lambda è costituito da script o programmi compilati e dalle relative dipendenze. Utilizza un pacchetto di implementazione per distribuire il codice della funzione a Lambda. Lambda supporta due tipi di pacchetti di implementazione: immagini di container e archivi di file .zip.

In questa pagina viene descritto come creare un file .zip come pacchetto di implementazione per il runtime Go e utilizzarlo quindi per implementare il codice della funzione in AWS Lambda tramite la Console di gestione AWS, l'AWS Command Line Interface (AWS CLI) e AWS Serverless Application Model (AWS SAM).

Nota che Lambda utilizza le autorizzazioni dei file POSIX, quindi potresti aver bisogno di impostare le autorizzazioni per la cartella del pacchetto di implementazione prima di creare l'archivio di file .zip.

Sezioni

- Creazione di un file .zip su macOS e Linux
- Creazione di un file .zip su Windows
- Creazione e aggiornamento delle funzioni Lambda di Go utilizzando file .zip

Creazione di un file .zip su macOS e Linux

I passaggi seguenti mostrano come compilare il file eseguibile utilizzando il comando `go build` e creare un pacchetto di implementazione con file .zip per Lambda. Prima di compilare il codice, assicurati di aver installato il pacchetto lambda da GitHub. Questo modulo fornisce un'implementazione dell'interfaccia di runtime, che gestisce l'interazione tra Lambda e il codice della funzione. Per scaricare questa libreria, esegui il comando seguente.

```
go get github.com/aws/aws-lambda-go/lambda
```

Se la funzione utilizza AWS SDK per Go, scarica il set standard di moduli SDK, insieme a tutti i client API di servizio AWS richiesti dall'applicazione. Per informazioni su come installare l'SDK for Go, consulta la Guida introduttiva a AWS SDK per Go versione 2.

Utilizzo della famiglia di runtime forniti

Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato nativamente in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un runtime

solo per il sistema operativo (la famiglia di runtime `provided`) per distribuire le funzioni Go in Lambda.

Creazione di un pacchetto di implementazione .zip (macOS/Linux)

1. Nella directory del progetto contenente il file `main.go` dell'applicazione, compila il tuo eseguibile. Tieni presente quanto segue:

   - L'eseguibile deve essere denominato `bootstrap`. Per ulteriori informazioni, consulta Convenzioni di denominazione dei gestori.
   - Imposta l'architettura del set di istruzioni di destinazione. I runtime del sistema operativo supportano sia arm64 sia x86_64.
   - Puoi usare il tag facoltativo `lambda.norpc` per escludere il componente Remote Procedure Call (RPC) della libreria lambda. Il componente RPC è richiesto solo quando si utilizza il runtime Go 1.x obsoleto. L'esclusione dell'RPC riduce le dimensioni del pacchetto di implementazione.

   Per l'architettura arm64:

   ```
   GOOS=linux GOARCH=arm64 go build -tags lambda.norpc -o bootstrap main.go
   ```

   Per l'architettura x86_64:

   ```
   GOOS=linux GOARCH=amd64 go build -tags lambda.norpc -o bootstrap main.go
   ```

2. (Opzionale) Potrebbe essere necessario compilare pacchetti con `CGO_ENABLED=0` impostato su Linux:

   ```
   GOOS=linux GOARCH=arm64 CGO_ENABLED=0 go build -o bootstrap -tags lambda.norpc
    main.go
   ```

   Questo comando crea un pacchetto binario stabile per le versioni standard della libreria C (`libc`) che possono essere diverse su Lambda e su altri dispositivi.

3. Crea un pacchetto di distribuzione comprimendo l'eseguibile in un file .zip.

   ```
   zip myFunction.zip bootstrap
   ```

> **Note**
>
> Il file `bootstrap` deve trovarsi nella posizione root del file .zip.

4. Crea la funzione . Tieni presente quanto segue:

   - Il file binario deve essere denominato `bootstrap` ma il nome del gestore può essere qualsiasi cosa. Per ulteriori informazioni, consulta Convenzioni di denominazione dei gestori.
   - L'opzione `--architectures` è necessaria solo se si utilizza arm64. Il valore predefinito è x86_64.
   - Per `--role`, specifica il nome della risorsa Amazon (ARN) del ruolo di esecuzione.

```
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
```

Creazione di un file .zip su Windows

La seguente procedura mostra come scaricare lo strumento build-lambda-zip per Windows da GitHub, compilare il tuo eseguibile e creare un pacchetto di implementazione .zip.

> **Note**
>
> Se non è già presente, è necessario installare git e quindi aggiungere il percorso dell'eseguibile `git` alla variabile di ambiente di Windows `%PATH%`.

Prima di compilare il codice, assicurati di aver installato la libreria lambda da GitHub. Per scaricare questa libreria, esegui il comando seguente.

```
go get github.com/aws/aws-lambda-go/lambda
```

Se la funzione utilizza AWS SDK per Go, scarica il set standard di moduli SDK, insieme a tutti i client API di servizio AWS richiesti dall'applicazione. Per informazioni su come installare l'SDK for Go, consulta la Guida introduttiva a AWS SDK per Go versione 2.

Utilizzo della famiglia di runtime forniti

Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato nativamente in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un runtime solo per il sistema operativo (la famiglia di runtime `provided`) per distribuire le funzioni Go in Lambda.

Creazione di un pacchetto di implementazione .zip (Windows)

1. Scarica lo strumento build-lambda-zip da GitHub.

   ```
   go install github.com/aws/aws-lambda-go/cmd/build-lambda-zip@latest
   ```

2. Utilizza lo strumento dal tuo `GOPATH` per creare un file .zip. Se si dispone di un'installazione predefinita di Go, lo strumento si trova solitamente in `%USERPROFILE%\Go\bin`. Altrimenti, vai alla posizione in cui è stato installato il runtime Go ed esegui una delle seguenti operazioni:

   cmd.exe

   In cmd.exe, esegui una delle operazioni riportate, a seconda della tua architettura del set di istruzioni di destinazione. I runtime del sistema operativo supportano sia arm64 sia x86_64.

   Puoi usare il tag facoltativo `lambda.norpc` per escludere il componente Remote Procedure Call (RPC) della libreria lambda. Il componente RPC è richiesto solo quando si utilizza il runtime Go 1.x obsoleto. L'esclusione dell'RPC riduce le dimensioni del pacchetto di implementazione.

   Example - Per l'architettura 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 - Per l'architettura 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

In PowerShell, esegui una delle operazioni riportate, a seconda della tua architettura del set di istruzioni di destinazione. I runtime del sistema operativo supportano sia arm64 sia x86_64.

Puoi usare il tag facoltativo `lambda.norpc` per escludere il componente Remote Procedure Call (RPC) della libreria lambda. Il componente RPC è richiesto solo quando si utilizza il runtime Go 1.x obsoleto. L'esclusione dell'RPC riduce le dimensioni del pacchetto di implementazione.

Per l'architettura 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
```

Per l'architettura 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. Crea la funzione . Tieni presente quanto segue:

- Il file binario deve essere denominato `bootstrap` ma il nome del gestore può essere qualsiasi cosa. Per ulteriori informazioni, consulta Convenzioni di denominazione dei gestori.

- L'opzione `--architectures` è necessaria solo se si utilizza arm64. Il valore predefinito è x86_64.
- Per `--role`, specifica il nome della risorsa Amazon (ARN) del ruolo di esecuzione.

```
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
```

Creazione e aggiornamento delle funzioni Lambda di Go utilizzando file .zip

Dopo aver creato il pacchetto di implementazione .zip, puoi utilizzarlo per creare una nuova funzione Lambda o aggiornarne una esistente. È possibile caricare il pacchetto .zip utilizzando la console Lambda, l'AWS Command Line Interface e l'API Lambda. Puoi anche creare e aggiornare le funzioni Lambda usando AWS Serverless Application Model (AWS SAM) e CloudFormation.

La dimensione massima per un pacchetto di implementazione .zip per Lambda è di 250 MB (dopo l'estrazione). Nota che questo limite si applica alla dimensione combinata di tutti i file caricati, inclusi eventuali livelli Lambda.

Il runtime Lambda necessita dell'autorizzazione per leggere i file nel pacchetto di distribuzione. Nella notazione ottale delle autorizzazioni di Linux, Lambda richiede 644 autorizzazioni per i file non eseguibili (rw-r--r--) e 755 autorizzazioni (rwxr-xr-x) per le directory e i file eseguibili.

In Linux e macOS, utilizza il comando `chmod` per modificare le autorizzazioni file su file e directory nel pacchetto di implementazione. Ad esempio, per assegnare a un file non eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 644 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta Set, View, Change, or Remove Permissions on an Object nella documentazione di Microsoft Windows.

> **Note**
>
> Se non concedi a Lambda le autorizzazioni necessarie per accedere alle directory del pacchetto di implementazione, Lambda imposta le autorizzazioni per tali directory su 755 (rwxr-xr-x).

Creazione e aggiornamento delle funzioni con file .zip utilizzando la console

Per creare una nuova funzione, devi prima creare la funzione nella console, quindi devi caricare il tuo archivio .zip. Per aggiornare una funzione esistente, apri la pagina relativa alla funzione, quindi segui la stessa procedura per aggiungere il file .zip aggiornato.

Se il file .zip ha dimensioni inferiori a 50 MB, è possibile creare o aggiornare una funzione caricando il file direttamente dal computer locale. Per i file .zip di dimensioni superiori a 50 MB, prima è necessario caricare il pacchetto in un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando la Console di gestione AWS, consulta la sezione Nozioni di base su Amazon S3. Per caricare file utilizzando la AWS CLI, consulta la sezione Spostamento di oggetti nella Guida per l'utente di AWS CLI.

> **Note**
>
> Non è possibile convertire una funzione di immagine di container esistente per utilizzare un archivio .zip. È necessario creare una nuova funzione.

Creazione di una nuova funzione (console)

1. Apri la pagina Funzioni della console Lambda e scegli Crea funzione.
2. Scegli Author from scratch (Crea da zero).
3. In Basic information (Informazioni di base) eseguire queste operazioni:
   a. In Nome funzione, inserisci il nome della funzione.
   b. In Runtime, selezionare `provided.al2023`.
4. (Opzionale) In Autorizzazioni espandere Modifica ruolo di esecuzione predefinito. Puoi creare un nuovo ruolo di esecuzione o utilizzare un ruolo esistente.
5. Scegli Crea funzione. Lambda crea una funzione di base "Hello world" utilizzando il runtime scelto.

Caricamento di un archivio .zip dal computer locale (console)

1. Nella pagina Funzioni della console Lambda, scegli la funzione per cui vuoi caricare il file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli File .zip.
5. Per caricare il file .zip, procedi come segue:

    a. Seleziona Carica, quindi seleziona il tuo file .zip nel selettore di file.
    b. Seleziona Apri.
    c. Selezionare Salva.

Caricamento di un archivio .zip da un bucket Amazon S3 (console)

1. Nella pagina Funzioni della console Lambda, scegli la funzione per cui vuoi caricare un nuovo file .zip.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, scegli Carica da.
4. Scegli Posizione Amazon S3.
5. Incolla l'URL del link Amazon S3 del tuo file .zip e scegli Salva.

Creazione e aggiornamento delle funzioni con file .zip utilizzando la AWS CLI

È possibile utilizzare la AWS CLI per creare una nuova funzione o aggiornare una funzione esistente mediante un file .zip. Utilizza i comandi create-function e update-function-code per implementare il tuo pacchetto .zip. Se il file .zip ha dimensioni inferiori a 50 MB, è possibile caricare il pacchetto .zip da una posizione di file nella macchina di compilazione locale. Per i file di dimensioni maggiori, è necessario caricare il pacchetto .zip da un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando la AWS CLI, consulta la sezione Spostamento di oggetti nella Guida per l'utente di AWS CLI.

> **Note**
>
> Se carichi il file .zip da un bucket Amazon S3 utilizzando la AWS CLI, il bucket deve trovarsi nella stessa Regione AWS della funzione.

Per creare una nuova funzione mediante un file .zip con la AWS CLI, è necessario specificare quanto segue:

- Il nome della funzione (`--function-name`)
- Il runtime della tua funzione (`--runtime`)
- Il nome della risorsa Amazon (ARN) del ruolo di esecuzione della funzione (`--role`)
- Il nome del metodo del gestore nel codice della funzione (`--handler`)

È inoltre necessario specificare la posizione del file .zip. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
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
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza l'opzione `--code` illustrata nel seguente comando di esempio. È necessario utilizzare il parametro `S3ObjectVersion` solo per gli oggetti con controllo delle versioni.

```
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
```

Per aggiornare una funzione esistente mediante la CLI, specifica il nome della funzione utilizzando il parametro `--function-name`. È inoltre necessario specificare la posizione del file .zip che desideri utilizzare per aggiornare il codice della funzione. Se il file .zip si trova in una cartella sulla macchina di compilazione locale, utilizza l'opzione `--zip-file` per specificare il percorso del file, come mostrato nel seguente comando di esempio.

```
aws lambda update-function-code --function-name myFunction \
--zip-file fileb://myFunction.zip
```

Per specificare la posizione del file .zip in un bucket Amazon S3, utilizza le opzioni `--s3-bucket` e `--s3-key` come illustrato nel seguente comando di esempio. È necessario utilizzare il parametro `--s3-object-version` solo per gli oggetti con controllo delle versioni.

```
aws lambda update-function-code --function-name myFunction \
--s3-bucket amzn-s3-demo-bucket --s3-key myFileName.zip --s3-object-version myObject
 Version
```

Creazione e aggiornamento delle funzioni con file .zip utilizzando l'API Lambda

Per creare e aggiornare le funzioni mediante un archivio di file .zip, utilizza le seguenti operazioni API:

- CreateFunction
- UpdateFunctionCode

Creazione e aggiornamento delle funzioni con file .zip utilizzando AWS SAM

AWS Serverless Application Model (AWS SAM) è un toolkit che aiuta a semplificare il processo di creazione ed esecuzione di applicazioni serverless su AWS. Definisci le risorse per la tua applicazione in un modello YAML o JSON e usa l'interfaccia della linea di comando AWS SAM (AWS SAM CLI) per compilare, creare pacchetti e implementare le tue applicazioni. Quando crei una funzione Lambda da un modello AWS SAM, AWS SAM crea automaticamente un pacchetto di implementazione .zip o un'immagine di container con il codice della funzione e le dipendenze specificate. Per ulteriori informazioni sull'utilizzo di AWS SAM per creare e implementare funzioni Lambda, consulta la pagina Nozioni di base su AWS SAM nella Guida introduttiva per gli sviluppatori di AWS Serverless Application Model.

Puoi anche utilizzare AWS SAM per creare una funzione Lambda mediante un archivio di file .zip esistente. Per creare una funzione Lambda utilizzando AWS SAM, puoi salvare il tuo file .zip in un bucket Amazon S3 o in una cartella locale sul tuo computer di compilazione. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando la AWS CLI, consulta la sezione Spostamento di oggetti nella Guida per l'utente di AWS CLI.

Nel modello AWS SAM, la risorsa `AWS::Serverless::Function` specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `CodeUri`: imposta il valore sull'URI Amazon S3, sul percorso della cartella locale o sull'oggetto FunctionCode del codice della funzione
- `Runtime`: imposta il runtime prescelto

Con AWS SAM, se il file .zip è superiore a 50 MB, non è necessario caricarlo prima in un bucket Amazon S3. AWS SAM può caricare pacchetti .zip fino alla dimensione massima consentita di 250 MB (dopo l'estrazione) da una posizione sulla macchina di compilazione locale.

Per ulteriori informazioni sull'implementazione di funzioni mediante il file .zip in AWS SAM, consulta la sezione AWS::Serverless::Function nella Guida per gli sviluppatori di AWS SAM.

Esempio: utilizzo di AWS SAM per creare una funzione Go con provided.al2023

1. Crea un modello AWS SAM con le seguenti proprietà:

   - BuildMethod: specifica il compilatore per l'applicazione. Utilizza `go1.x`.
   - Runtime: utilizza `provided.al2023`.
   - CodeUri: inserisci il percorso del tuo codice.
   - Architetture: usa `[arm64]` per l'architettura arm64. Per l'architettura del set di istruzioni x86_64, utilizza `[amd64]` oppure rimuovi la proprietà `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. Usa il comando sam build per compilare l'eseguibile.

   ```
   sam build
   ```

3. Utilizza il comando sam deploy per implementare la funzione su Lambda.

   ```
   sam deploy --guided
   ```

Creazione e aggiornamento delle funzioni con file .zip utilizzando CloudFormation

È possibile utilizzare CloudFormation per creare una funzione Lambda mediante un archivio di file .zip. Per creare una funzione Lambda da un file .zip, devi prima caricare il file su un bucket Amazon S3. Per istruzioni su come caricare un file in un bucket Amazon S3 utilizzando la AWS CLI, consulta la sezione Spostamento di oggetti nella Guida per l'utente di AWS CLI.

Nel modello CloudFormation, la risorsa `AWS::Lambda::Function` specifica la funzione Lambda. In questa risorsa, imposta le seguenti proprietà per creare una funzione utilizzando un archivio di file .zip:

- `PackageType`: imposta il valore su `Zip`
- `Code`: inserisci il nome del bucket Amazon S3 e il nome del file .zip nei campi `S3Bucket` e `S3Key`
- `Runtime`: imposta il runtime prescelto

Il file .zip che CloudFormation genera non può superare i 4 MB. Per ulteriori informazioni sull'implementazione di funzioni mediante un file .zip in CloudFormation, consulta la sezione AWS::Lambda::Function nella Guida per gli sviluppatori di CloudFormation.

Distribuzione delle funzioni Go Lambda con immagini di container

Esistono due modi per creare un'immagine di container per una funzione Lambda in Go:

- Utilizzo di un'immagine di AWS base solo per il sistema operativo

  Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato nativamente in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un'immagine di base solo per il sistema operativo per creare immagini Go per Lambda. Per rendere l'immagine compatibile con Lambda, devi includere il pacchetto `aws-lambda-go/lambda` nell'immagine.
- Utilizzo di un'immagine non di base AWS

  È possibile utilizzare un'immagine di base alternativa da un altro registro del container, come ad esempio Alpine Linux o Debian. Puoi anche utilizzare un'immagine personalizzata creata dalla tua organizzazione. Per rendere l'immagine compatibile con Lambda, devi includere il pacchetto `aws-lambda-go/lambda` nell'immagine.

> **Tip**
>
> Per ridurre il tempo necessario all'attivazione delle funzioni del container Lambda, consulta Utilizzo di compilazioni a più fasi nella documentazione Docker. Per creare immagini di container efficienti, segui le best practice per scrivere file Docker.

Questa pagina spiega come creare, testare e implementare le immagini di container per Lambda.

AWS immagini di base per l'implementazione delle funzioni Go

Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato nativamente in un file binario eseguibile, non richiede un runtime linguistico dedicato. Usa un'immagine di base solo per il sistema operativo per distribuire le funzioni Go su Lambda.

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
Runtime solo per il sistema operativo	`provided.al2023`	Amazon Linux 2023	30 giugno 2029	31 luglio 2029	31 agosto 2029
Runtime solo per il sistema operativo	`provided.al2`	Amazon Linux 2	31 luglio 2026	31 agosto 2026	30 settembre 2026

Galleria pubblica di Amazon Elastic Container Registry: gallery.ecr. aws/lambda/provided

Client di interfaccia di runtime per Go

Il pacchetto `aws-lambda-go/lambda` include un'implementazione dell'interfaccia di runtime. Per esempi di come utilizzare `aws-lambda-go/lambda` nell'immagine, consulta le sezioni Utilizzo di un'immagine di AWS base solo per il sistema operativo o Utilizzo di un'immagine non di base AWS.

Utilizzo di un'immagine di AWS base solo per il sistema operativo

Go è implementato in modo diverso rispetto ad altri runtime gestiti. Poiché Go viene compilato nativamente in un file binario eseguibile, non richiede un runtime linguistico dedicato. Utilizza un'immagine di base solo per il sistema operativo per creare immagini di container per le funzioni Go.

Tag	Runtime	Sistema operativo	Dockerfile	Raggiunta obsolescenza
al2023	Runtime solo per il sistema operativo	Amazon Linux 2023	Dockerfile per Runtime solo per sistema operativo su GitHub	30 giugno 2029
al2	Runtime solo per	Amazon Linux 2	Dockerfile per Runtime solo per sistema operativo su GitHub	31 luglio 2026

Tag	Runtime	Sistema operativo	Dockerfile	Raggiunta obsolescenza
	il sistema operativo			

Per ulteriori informazioni su queste immagini di base, consulta la documentazione fornita nella galleria pubblica di Amazon ECR.

È necessario includere il pacchetto aws-lambda-go/lambda nel gestore Go. Questo pacchetto implementa il modello di programmazione per Go, inclusa l'interfaccia di runtime.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS CLI versione 2
- Docker (versione minima 25.0.0)
- Il plugin buildx di Docker.
- Go

Creazione di un'immagine dall'immagine di base provided.al2023

Creazione e implementazione di una funzione Go con l'immagine di base **`provided.al2023`**

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir hello
cd hello
```

2. Inizializza un nuovo modulo Go.

```
go mod init example.com/hello-world
```

3. Aggiungi la libreria lambda come dipendenza del nuovo modulo.

```
go get github.com/aws/aws-lambda-go/lambda
```

4. Crea un file denominato `main.go`, quindi aprilo in un editor di testo. Questo è il codice per la funzione Lambda. A fini di test, puoi utilizzare il codice di esempio seguente o sostituirlo con il tuo codice personalizzato.

```
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. Utilizza un editor di testo per creare un file Dockerfile nella directory del progetto.

   - Il seguente Dockerfile di esempio utilizza una build multi-fase. Ciò consente di utilizzare un'immagine di base diversa in ogni passaggio. Puoi utilizzare un'immagine, ad esempio un'immagine di base Go, per compilare il codice e creare il file binario eseguibile. È quindi possibile utilizzare un'immagine diversa, ad esempio `provided.al2023`, nell'istruzione `FROM` finale per definire l'immagine da implementare in Lambda. Il processo di compilazione è separato dall'immagine di implementazione finale, quindi l'immagine finale contiene solo i file necessari per eseguire l'applicazione.
   - Puoi usare il tag facoltativo `lambda.norpc` per escludere il componente Remote Procedure Call (RPC) della libreria lambda. Il componente RPC è richiesto solo quando si utilizza il runtime Go 1.x obsoleto. L'esclusione dell'RPC riduce le dimensioni del pacchetto di implementazione.
   - Nota che l'esempio Dockerfile non include un'istruzione USER. Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux

predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione USER.

Example - Dockerfile di compilazione multi-fase

> **Note**
>
> Assicurati che la versione di Go specificata nel tuo Dockerfile (ad esempio `golang:1.20`) sia la stessa versione di Go che hai usato per creare l'applicazione.

```
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. Crea l'immagine Docker con il comando docker build. L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il tag `test`. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'`--provenance=false`opzione.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Se intendi creare una funzione Lambda utilizzando l'architettura del set di ARM64 istruzioni, assicurati di modificare il comando per utilizzare invece l'`--platform linux/arm64`opzione.

(Facoltativo) Test dell'immagine in locale

Usa il simulatore dell'interfaccia di runtime per testare l'immagine in locale. Il simulatore dell'interfaccia di runtime è incluso nell'immagine di base `provided.al2023`.

Esecuzione del simulatore dell'interfaccia di runtime sul computer locale

1. Avvia l'immagine Docker con il comando docker run. Tenere presente quanto segue:

   - `docker-image` è il nome dell'immagine e `test` è il tag.
   - `./main` è l'`ENTRYPOINT` del Dockerfile.

   ```
   docker run -d -p 9000:8080 \
   --entrypoint /usr/local/bin/aws-lambda-rie \
   docker-image:test ./main
   ```

   Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

2. Da una nuova finestra di terminale, invia un evento al seguente endpoint utilizzando un comando curl:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Alcune funzioni potrebbero richiedere un payload JSON. Esempio:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

3. Ottieni l'ID del container.

   ```
   docker ps
   ```

4. Utilizza il comando docker kill per arrestare il container. In questo comando, sostituisci `3766c4ab331c` con l'ID del container del passaggio precedente.

   ```
   docker kill 3766c4ab331c
   ```

Implementazione dell'immagine

Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Esegui il get-login-passwordcomando per autenticare la CLI Docker nel tuo registro Amazon ECR.

   - Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
   - Sostituiscilo `111122223333` con il tuo ID. Account AWS

   ```
   aws ecr get-login-password --region us-east-1 | docker login --username AWS --
   password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
   ```

2. Crea un repository in Amazon ECR utilizzando il commando create-repository.

   ```
   aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
   scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
   ```

   > **Note**
   >
   > Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS

   In caso di esito positivo, dovresti ottenere una risposta simile a questa:

   ```
   {
       "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. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando docker tag per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:
   - `docker-image:test` è il nome e tag dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
   - Sostituisci l'`<ECRrepositoryUri>` con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

   Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

5. Esegui il comando docker push per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crea un ruolo di esecuzione per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'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**
>
> È possibile creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta Autorizzazioni multiaccount Amazon ECR.

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nell'archivio Amazon ECR e quindi utilizzare il update-function-codecomando per distribuire l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per distribuire la nuova immagine nella stessa funzione Lambda, è necessario utilizzare update-function-codeil comando, anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Utilizzo di un'immagine non di base AWS

Puoi creare un'immagine contenitore per Go da un'immagine non di AWS base. Nei passaggi di esempio che seguono, Dockerfile utilizza un'immagine di base Alpine.

È necessario includere il pacchetto aws-lambda-go/lambda nel gestore Go. Questo pacchetto implementa il modello di programmazione per Go, inclusa l'interfaccia di runtime.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- AWS CLI versione 2
- Docker (versione minima 25.0.0)
- Il plugin buildx di Docker.
- Go

Creazione di un'immagine da un'immagine di base alternativa

Creazione e implementazione di una funzione Go con un'immagine di base Alpine

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir hello
cd hello
```

2. Inizializza un nuovo modulo Go.

```
go mod init example.com/hello-world
```

3. Aggiungi la libreria lambda come dipendenza del nuovo modulo.

```
go get github.com/aws/aws-lambda-go/lambda
```

4. Crea un file denominato `main.go`, quindi aprilo in un editor di testo. Questo è il codice per la funzione Lambda. A fini di test, puoi utilizzare il codice di esempio seguente o sostituirlo con il tuo codice personalizzato.

```
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. Utilizza un editor di testo per creare un file Dockerfile nella directory del progetto. Nell'esempio che segue, Dockerfile utilizza un'immagine di base Alpine. Nota che l'esempio Dockerfile non include un'istruzione USER. Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente `root` quando non viene fornita alcuna istruzione `USER`.

   Example Dockerfile

   > **Note**
   >
   > Assicurati che la versione di Go specificata nel tuo Dockerfile (ad esempio `golang:1.20`) sia la stessa versione di Go che hai usato per creare l'applicazione.

```
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. Crea l'immagine Docker con il comando docker build. L'esempio seguente assegna un nome all'immagine `docker-image` e le assegna il tag `test`. Per rendere l'immagine compatibile con Lambda, è necessario utilizzare l'`--provenance=false`opzione.

```
docker buildx build --platform linux/amd64 --provenance=false -t docker-image:test
 .
```

> **Note**
>
> Il comando specifica l'opzione `--platform linux/amd64` per garantire che il container sia compatibile con l'ambiente di esecuzione di Lambda, indipendentemente dall'architettura della macchina di sviluppo. Se intendi creare una funzione Lambda utilizzando l'architettura del set di ARM64 istruzioni, assicurati di modificare il comando per utilizzare invece l'`--platform linux/arm64`opzione.

(Facoltativo) Test dell'immagine in locale

Usa il simulatore dell'interfaccia di runtime per testare l'immagine in locale. Puoi creare l'emulatore nella tua immagine o seguire la procedura riportata e installarlo sul tuo computer locale.

Installazione ed esecuzione dell'emulatore di interfaccia di runtime sul computer locale

1. Dalla directory del progetto, esegui il comando seguente per scaricare l'emulatore di interfaccia di runtime (architettura x86-64) GitHub e installarlo sul computer locale.

   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
   ```

   Per installare l'emulatore arm64, sostituisci l'URL del GitHub repository nel comando precedente con il seguente:

```
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
```

Per installare l'emulatore arm64, sostituisci `$downloadLink` con quanto segue:

```
https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/
download/aws-lambda-rie-arm64
```

2. Avvia l'immagine Docker con il comando docker run. Tenere presente quanto segue:

   - `docker-image` è il nome dell'immagine e `test` è il tag.
   - `/main` è l'ENTRYPOINT del Dockerfile.

   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
```

Questo comando esegue l'immagine come container e crea un endpoint locale in `localhost:9000/2015-03-31/functions/function/invocations`.

> **Note**
>
> Se hai creato l'immagine Docker per l'architettura del set di ARM64 istruzioni, assicurati di utilizzare l'`--platform linux/arm64`opzione invece di. `--platform linux/amd64`

3. Pubblica un evento nell'endpoint locale.

   Linux/macOS

   Su MacOS o Linux, esegui il comando seguente `curl`:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
   ```

   Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

   ```
   curl "http://localhost:9000/2015-03-31/functions/function/invocations" -d
    '{"payload":"hello world!"}'
   ```

   PowerShell

   In PowerShell, esegui il seguente `Invoke-WebRequest` comando:

   ```
   Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
   invocations" -Method Post -Body '{}' -ContentType "application/json"
   ```

   Questo comando richiama la funzione con un evento vuoto e restituisce una risposta. Se stai utilizzando il tuo codice della funzione anziché quello di esempio, potresti voler richiamare la funzione con un payload JSON. Esempio:

```
Invoke-WebRequest -Uri "http://localhost:9000/2015-03-31/functions/function/
invocations" -Method Post -Body '{"payload":"hello world!"}' -ContentType
 "application/json"
```

4. Ottieni l'ID del container.

```
docker ps
```

5. Utilizza il comando docker kill per arrestare il container. In questo comando, sostituisci `3766c4ab331c` con l'ID del container del passaggio precedente.

```
docker kill 3766c4ab331c
```

Implementazione dell'immagine

Caricamento dell'immagine su Amazon ECR e creazione della funzione Lambda

1. Esegui il get-login-passwordcomando per autenticare la CLI Docker nel tuo registro Amazon ECR.
   - Imposta il `--region` valore nel Regione AWS punto in cui desideri creare il repository Amazon ECR.
   - Sostituiscilo `111122223333` con il tuo ID. Account AWS

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-east-1.amazonaws.com
```

2. Crea un repository in Amazon ECR utilizzando il comando create-repository.

```
aws ecr create-repository --repository-name hello-world --region us-east-1 --image-
scanning-configuration scanOnPush=true --image-tag-mutability MUTABLE
```

> **Note**
>
> Il repository Amazon ECR deve corrispondere alla funzione Lambda. Regione AWS

In caso di esito positivo, dovresti ottenere una risposta simile a questa:

```
{
    "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. Copia il `repositoryUri` dall'output del passaggio precedente.
4. Esegui il comando docker tag per etichettare l'immagine locale nel repository Amazon ECR come versione più recente. In questo comando:

   - `docker-image:test` è il nome e tag dell'immagine Docker. Si tratta del nome e del tag dell'immagine specificati nel comando `docker build`.
   - Sostituisci l'`<ECRrepositoryUri>` con l'`repositoryUri` copiato. Assicurati di includere `:latest` alla fine dell'URI.

```
docker tag docker-image:test <ECRrepositoryUri>:latest
```

Esempio:

```
docker tag docker-image:test 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-
world:latest
```

5. Esegui il comando docker push per implementare la tua immagine locale sul repository Amazon ECR. Assicurati di includere `:latest` alla fine dell'URI del repository.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

6. Crea un ruolo di esecuzione per la funzione, se non lo hai già fatto. Il nome della risorsa Amazon (ARN) del ruolo ti occorrerà nel passaggio successivo.
7. Creazione della funzione Lambda Per `ImageUri`, specifica l'URI del repository creato in precedenza. Assicurati di includere `:latest` alla fine dell'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**
>
> È possibile creare una funzione utilizzando un'immagine in un AWS account diverso, purché l'immagine si trovi nella stessa regione della funzione Lambda. Per ulteriori informazioni, consulta Autorizzazioni multiaccount Amazon ECR.

8. Richiama la funzione.

```
aws lambda invoke --function-name hello-world response.json
```

Dovresti ottenere una risposta simile a questa:

```
{
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

9. Per vedere l'output della funzione, controlla il file `response.json`.

Per aggiornare il codice della funzione, devi creare nuovamente l'immagine, caricare la nuova immagine nell'archivio Amazon ECR e quindi utilizzare il update-function-codecomando per distribuire l'immagine nella funzione Lambda.

Lambda risolve il tag dell'immagine in un digest di immagine specifico. Ciò significa che se punti il tag immagine utilizzato per implementare la funzione su una nuova immagine in Amazon ECR, Lambda non aggiorna automaticamente la funzione per utilizzare la nuova immagine.

Per distribuire la nuova immagine nella stessa funzione Lambda, è necessario utilizzare update-function-codeil comando, anche se il tag dell'immagine in Amazon ECR rimane lo stesso. Nell'esempio seguente, l'opzione `--publish` crea una nuova versione della funzione utilizzando l'immagine del container aggiornata.

```
aws lambda update-function-code \
  --function-name hello-world \
  --image-uri 111122223333.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest \
  --publish
```

Utilizzo dei livelli per le funzioni Lambda in Go

Consigliamo di non utilizzare i livelli per gestire le dipendenze per le funzioni Lambda scritte in Go. Questo perché le funzioni Lambda in Go vengono compilate in un unico eseguibile, che viene fornito a Lambda quando si distribuisce la funzione. Questo eseguibile contiene il codice di funzione compilato, insieme a tutte le sue dipendenze. L'uso dei livelli non solo complica questo processo, ma comporta anche un aumento dei tempi di avvio a freddo, poiché le funzioni devono caricare manualmente assiemi aggiuntivi in memoria durante la fase di inizializzazione.

Per utilizzare dipendenze esterne con i gestori Go, includile direttamente nel pacchetto di implementazione. In questo modo, semplifichi il processo di implementazione e sfrutti anche le ottimizzazioni integrate del compilatore Go. Per un esempio di come importare e utilizzare una dipendenza come l'SDK AWS per Go nella tua funzione, consulta the section called “Gestore”.

Registrare e monitorare le funzioni Lambda con Go

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della funzione. L'ambiente del runtime Lambda invia i dettagli su ogni richiamo al flusso di log e inoltra i log e l'output del codice della funzione. Per ulteriori informazioni, consulta Lambda invia automaticamente i log delle funzioni a CloudWatch Logs..

Questa pagina descrive come produrre un output di registro dal codice della funzione Lambda e accedere ai log utilizzando AWS Command Line Interface la console Lambda o la console. CloudWatch

Sections

- Creazione di una funzione che restituisce i registri
- Visualizzazione dei log nella console Lambda
- Visualizzazione dei log nella console CloudWatch
- Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI
- Eliminazione dei log

Creazione di una funzione che restituisce i registri

Per i log di output del codice della funzione, puoi usare i metodi del pacchetto fmt o qualsiasi libreria di registrazione che scriva in `stdout` o `stderr`. Nell'esempio seguente viene utilizzato il pacchetto log.

Example main.go – Registrazione

```
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 Formato dei 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
```

Il runtime di Go registra `START`, `END` e `REPORT` per ogni chiamata. La riga del report fornisce i seguenti dettagli.

Campi dati della riga REPORT

- RequestId— L'ID univoco della richiesta per la chiamata.
- Durata – La quantità di tempo che il metodo del gestore della funzione impiega durante l'elaborazione dell'evento.
- Durata fatturata – La quantità di tempo fatturata per la chiamata.
- Dimensioni memoria – La quantità di memoria allocata per la funzione.

- Quantità max utilizzata – La quantità di memoria utilizzata dalla funzione. Quando le invocazioni condividono un ambiente di esecuzione, Lambda riporta la memoria massima utilizzata in tutte le invocazioni. Questo comportamento potrebbe comportare un valore riportato superiore al previsto.
- Durata Init – Per la prima richiesta servita, la quantità di tempo impiegato dal runtime per caricare la funzione ed eseguire il codice al di fuori del metodo del gestore.
- XRAY TraceId — Per le richieste tracciate, l'ID di traccia.AWS X-Ray
- SegmentId— Per le richieste tracciate, l'ID del segmento X-Ray.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor del codice incorporato, troverai i log nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l'output del log nella sezione Dettagli.

Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (***your-function-name***/aws/lambda/).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Viene visualizzato un flusso di log quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire le chiamate simultanee. Per trovare i log per una chiamata specifica, consigliamo di strumentare la funzione con. AWS X-Ray X-Ray registra i dettagli sulla richiesta e il flusso di log nella traccia.

Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della AWS CLI versione 2.

È possibile utilizzare AWS CLI per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'`base64` utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `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
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-`

`base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
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
```

L'utilità `base64` è disponibile su Linux, macOS e Ubuntu su Windows. Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

Example Script get-logs.sh

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza `sed` per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.

L'cli-binary-formatopzione è obbligatoria se utilizzi la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 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 (solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il seguente comando per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
    "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"
}
```


Eliminazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, eliminare il gruppo di log o configurare un periodo di conservazione trascorso il quale i log vengono eliminati automaticamente.

Strumentazione del codice Go in AWS Lambda

Lambda si integra con AWS X-Ray per aiutarti a tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzare X-Ray per tracciare una richiesta mentre attraversa le risorse nell'applicazione, che possono includere funzioni Lambda e altri servizi AWS.

Per inviare dati di tracciamento a X-Ray, è possibile utilizzare una delle due librerie SDK:

- AWS Distro for OpenTelemetry (ADOT): una distribuzione supportata da AWS sicura, pronta per la produzione, dell'SDK OpenTelemetry (OTel).
- SDK AWS X-Ray per Go: un SDK per generare e inviare i dati di traccia su X-Ray.

Ciascun SDK offre dei modi per inviare i dati di telemetria al servizio X-Ray. Puoi quindi utilizzare X-Ray per visualizzare, filtrare e analizzare le metriche delle prestazioni dell'applicazione per identificare i problemi e le opportunità di ottimizzazione.

> **Important**
>
> Gli SDK X-Ray e Powertools per AWS Lambda fanno parte di una soluzione di strumentazione strettamente integrata offerta da AWS. I livelli Lambda ADOT fanno parte di uno standard di settore per la strumentazione di tracciamento che in generale raccoglie più dati, ma potrebbero non essere adatti a tutti i casi d'uso. È possibile implementare il tracciamento end-to-end di X-Ray utilizzando entrambe le soluzioni. Per ulteriori informazioni sulla scelta più adatta, consulta Scegliere tra le AWS Distro per OpenTelemetry e SDK X-Ray.

Sezioni

- Utilizzo di ADOT per strumentare le funzioni Go
- Utilizzo dell'SDK X-Ray per strumentare le funzioni Go
- Attivazione del tracciamento con la console Lambda
- Attivazione del tracciamento con l'API Lambda
- Attivazione del tracciamento con CloudFormation
- Interpretazione di una traccia X-Ray

Utilizzo di ADOT per strumentare le funzioni Go

ADOT fornisce livelli Lambda completamente gestiti che mettono insieme tutto il necessario per raccogliere i dati di telemetria utilizzando l'SDK OTel. Usando questo livello, è possibile strumentare le funzioni Lambda senza dover modificare alcun codice funzione. È inoltre possibile configurare il livello per eseguire l'inizializzazione personalizzata di OTel. Per ulteriori informazioni, consulta la sezione relativa alla configurazione personalizzata per ADOT Collector su Lambda nella documentazione di ADOT.

Per i runtime Go, puoi aggiungere il livello Lambda gestito da AWS per ADOT Go per strumentare automaticamente le tue funzioni. Per istruzioni dettagliate su come aggiungere questo livello, consulta Supporto Lambda di AWS Distro for OpenTelemetry per Go nella documentazione di ADOT.

Utilizzo dell'SDK X-Ray per strumentare le funzioni Go

Per registrare i dettagli sulle chiamate effettuate dalla funzione Lambda ad altre risorse nell'applicazione, puoi utilizzare anche l'SDK AWS X-Ray per Go. Per ottenere l'SDK, scaricalo dal relativo repository GitHub con `go get`:

```
go get github.com/aws/aws-xray-sdk-go
```

Per strumentare i client AWS SDK, passare il client al metodo `xray.AWS()`. Quindi potrai tracciare le chiamate utilizzando la versione `WithContext` del metodo.

```
svc := s3.New(session.New())
xray.AWS(svc.Client)
...
svc.ListBucketsWithContext(ctx aws.Context, input *ListBucketsInput)
```

Dopo aver aggiunto le dipendenze corrette e aver apportato le modifiche necessarie al codice, attivare il tracciamento nella configurazione della funzione tramite la console Lambda o l'API.

Attivazione del tracciamento con la console Lambda

Per attivare il tracciamento attivo sulla funzione Lambda con la console, attenersi alla seguente procedura:

Per attivare il tracciamento attivo

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione) e quindi Monitoring and operations tools (Strumenti di monitoraggio e operazioni).
4. Nel riquadro Strumenti di monitoraggio aggiuntivi scegli Modifica.
5. In CloudWatch Application Signals AWS X-Ray e, scegli Enable for Lambda service trace.
6. Selezionare Salva.

Attivazione del tracciamento con l'API Lambda

Configurare il tracciamento sulla funzione Lambda con la AWS CLI o SDK AWS e utilizzare le seguenti operazioni API:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Il seguente comando AWS CLI di esempio consente il tracciamento attivo su una funzione denominata my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

La modalità di tracciamento fa parte della configurazione specifica della versione quando si pubblica una versione della funzione. Non è possibile modificare la modalità di tracciamento in una versione pubblicata.

Attivazione del tracciamento con CloudFormation

Per attivare il tracciamento su una risorsa `AWS::Lambda::Function` in un modello CloudFormation, utilizzare la proprietà `TracingConfig`.

Example function-inline.yml – Configurazione del tracciamento

```
Resources:
```

```
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Per una risorsa AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, utilizzare la proprietà `Tracing`.

Example template.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretazione di una traccia X-Ray

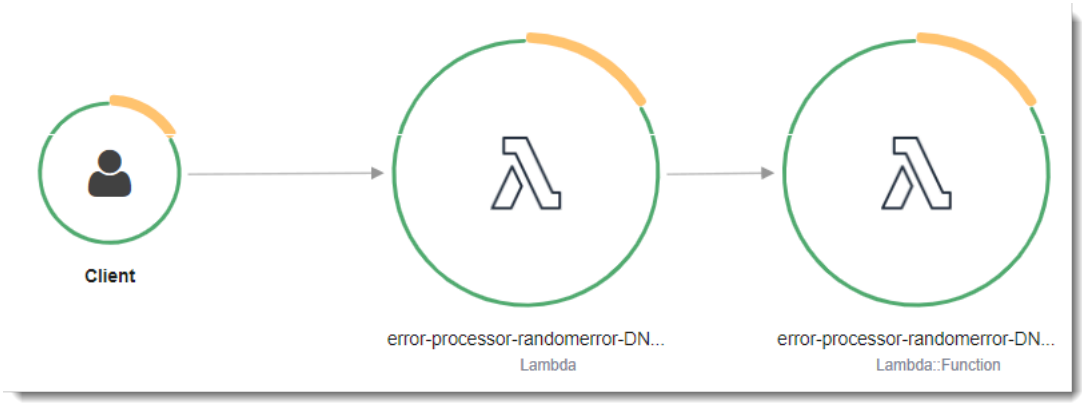
La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al ruolo di esecuzione della funzione. In caso contrario, aggiungere la policy AWSXRayDaemonWriteAccess al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico dei servizi X-Ray mostra informazioni sull'applicazione e tutti i suoi componenti. Il seguente esempio mostra un'applicazione con due funzioni. La funzione principale elabora gli eventi e talvolta restituisce errori. La seconda funzione elabora gli errori che appaiono nel gruppo di log della prima funzione e utilizza l'SDK AWS per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon CloudWatch Logs.

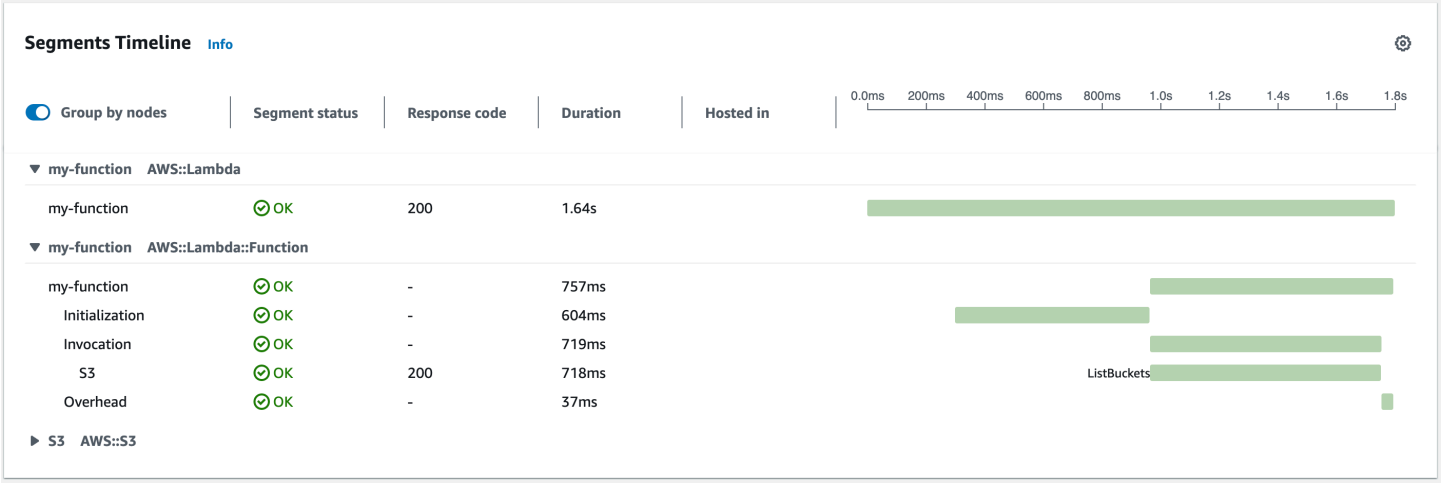


X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non è possibile configurare la frequenza di campionamento di X-Ray per le funzioni.

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda. L'esempio seguente mostra una traccia con questi 2 segmenti. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.



Questo esempio espande il segmento `AWS::Lambda::Function` per visualizzare i relativi tre sottosegmenti.

> **Note**
>
> AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. La traccia di esempio mostrata qui illustra il segmento di funzione vecchio stile. Le differenze tra i segmenti vecchio e nuovo stile sono descritte nei paragrafi seguenti.
> Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni in tutte le Regioni AWS, ad eccezione delle regioni Cina e GovCloud, passeranno all'utilizzo dei messaggi di log e dei segmenti di traccia di nuovo formato.

Il segmento di funzioni vecchio stile contiene i seguenti sottosegmenti:

- Inizializzazione – Rappresenta il tempo trascorso a caricare la funzione e ad eseguire il codice di inizializzazione. Questo sottosegmento viene visualizzato solo per il primo evento che viene elaborato da ogni istanza della funzione.
- Chiamata: rappresenta il tempo impiegato per eseguire il codice del gestore.
- Overhead: rappresenta il tempo impiegato dal runtime Lambda per prepararsi a gestire l'evento successivo.

Il segmento di funzione di nuovo stile non contiene un sottosegmento `Invocation`. I sottosegmenti dei clienti sono invece collegati direttamente al segmento di funzioni. Per ulteriori informazioni sulla

struttura dei segmenti di funzioni vecchio e nuovo stile, consulta the section called “Informazioni sui monitoraggi di X-Ray”.

È inoltre possibile strumentare i client HTTP, registrare query SQL e creare segmenti secondari personalizzati con annotazioni e metadati. Per ulteriori informazioni, consulta SDK AWS X-Ray per Go nella Guida per gli sviluppatori di AWS X-Ray.

> **Prezzi**
>
> È possibile utilizzare il tracciamento X-Ray gratuitamente ogni mese fino a un certo limite come parte del piano gratuito di AWS. Oltre la soglia, X-Ray addebita lo storage di traccia e il recupero. Per ulteriori informazioni, consulta Prezzi di AWS X-Ray.

Compilazione di funzioni Lambda con C#

Puoi eseguire l'applicazione.NET in Lambda utilizzando il runtime gestito.NET 8, un runtime personalizzato o un'immagine del contenitore. Dopo aver compilato il codice dell'applicazione, potrai implementarlo su Lambda come file .zip o come immagine di container. Lambda fornisce i runtime seguenti per i linguaggi .NET:

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
.NET 10	`dotnet10`	Amazon Linux 2023	14 novembre 2028	14 dicembre 2028	15 gennaio 2029
.NET 9 (solo contenitore)	`dotnet9`	Amazon Linux 2023	10 novembre 2026	Non programmato	Non programmato
.NET 8	`dotnet8`	Amazon Linux 2023	10 novembre 2026	10 dicembre 2026	11 gennaio 2027

Configurazione dell'ambiente di sviluppo .NET

Per sviluppare e creare le funzioni Lambda, puoi utilizzare uno qualsiasi degli ambienti di sviluppo integrati.NET comunemente disponibili (IDEs), inclusi Microsoft Visual Studio, Visual Studio Code e JetBrains Rider. Per semplificare l'esperienza di sviluppo, AWS fornisce un set di modelli di progetto.NET e l'interfaccia a riga di `Amazon.Lambda.Tools` comando (CLI).

Emetti i seguenti comandi .NET della CLI per installare questi modelli di progetto e strumenti da riga di comando.

Installazione dei modelli di progetto .NET

Per installare i modelli di progetto, esegui il comando riportato:

```
dotnet new install Amazon.Lambda.Templates
```

Installazione e aggiornamento degli strumenti della CLI

Esegui i comandi riportati qui di seguito per installare, aggiornare e disinstallare la CLI di `Amazon.Lambda.Tools`.

Per installare gli strumenti a riga di comando:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Per aggiornare gli strumenti a riga di comando:

```
dotnet tool update -g Amazon.Lambda.Tools
```

Per disinstallare gli strumenti a riga di comando:

```
dotnet tool uninstall -g Amazon.Lambda.Tools
```

Definire l'handler della funzione Lambda in C#

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Questa pagina descrive come lavorare con i gestori di funzioni Lambda in C# per operare con il runtime gestito da .NET, incluse le opzioni per la configurazione del progetto, le convenzioni di denominazione e le best practice. Questa pagina include anche un esempio di funzione Lambda C# che raccoglie informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon Simple Storage Service (S3). Per informazioni su come distribuire una funzione dopo averla scritta, consulta o. the section called “Pacchetto di implementazione” the section called “Implementazione di immagini di container”

Argomenti

- Configurazione del progetto del gestore C#
- Esempio di codice della funzione Lambda C#
- Gestori di librerie di classi
- Gestori di assembly eseguibili
- Firme del gestore valide per le funzioni C#
- Convenzioni di denominazione dei gestori
- Serializzazione nelle funzioni Lambda C#
- Funzioni basate su file
- Accesso e utilizzo dell'oggetto contestuale Lambda
- Usando la SDK per .NET v3 nel tuo gestore
- Accesso alle variabili d'ambiente
- Utilizzo dello stato globale
- Semplificazione del codice delle funzioni con il framework Lambda Annotations
- Best practice di codice per le funzioni Lambda con C#

Configurazione del progetto del gestore C#

Quando lavori con le funzioni Lambda in C#, il processo prevede la scrittura del codice, quindi l’implementazione del codice in Lambda. Esistono due diversi modelli di esecuzione per

l'implementazione delle funzioni Lambda in .NET: l'approccio della libreria di classi e l'approccio dell'assembly eseguibile.

Quando segui l'approccio della libreria di classi, impacchetti il codice della funzione come assembly .NET (`.dll`) e lo implementi in Lambda con il runtime gestito da .NET (`dotnet8`). Per il nome del gestore, Lambda si aspetta una stringa nel formato `AssemblyName::Namespace.Classname::Methodname`. Durante la fase di inizializzazione della funzione, la classe della funzione viene inizializzata e viene eseguito qualsiasi codice nel costruttore.

Quando segui l'approccio dell'assembly eseguibile, utilizzi la funzionalità delle istruzioni di primo livello introdotta per la prima volta in C# 9. Questo approccio genera un assembly eseguibile che Lambda esegue ogni volta che riceve un comando invoke per la tua funzione. Con questo approccio, utilizzi anche il runtime gestito da .NET (`dotnet8`). Per il nome del gestore, fornisci a Lambda il nome dell'assembly eseguibile da eseguire.

L'esempio principale in questa pagina illustra l'approccio delle librerie di classi. Puoi inizializzare il progetto C# Lambda in vari modi, ma il modo più semplice è utilizzare l'interfaccia CLI .NET con la CLI `Amazon.Lambda.Tools`. Configura la CLI `Amazon.Lambda.Tools` seguendo i passaggi indicati in the section called "Ambiente di sviluppo". Quindi, inizializza il progetto con il seguente comando.

```
dotnet new lambda.EmptyFunction --name ExampleCS
```

Questo comando genera la seguente struttura di file:

```
/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
```

In questa struttura di file, la logica del gestore principale per la funzione risiede nel file `Function.cs`.

Esempio di codice della funzione Lambda C#

Il seguente esempio di codice della funzione Lambda C# raccoglie le informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon S3.

Example Funzione Lambda **`Function.cs`**

```
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);
        }
    }
}
```

Questo file `Function.cs` contiene le sezioni seguenti:

- Istruzioni `using`: usale per importare le classi C# richieste dalla funzione Lambda.

- `[assembly: LambdaSerializer(...)]`: `LambdaSerializer` è un attributo assembly che indica a Lambda di convertire automaticamente i payload di eventi JSON in oggetti C# prima di passarli alla funzione.
- `namespace ExampleLambda`: definisce il namespace. In C#, il nome del namespace non deve corrispondere al nome del file.
- `public class Order {...}`: definisce la forma dell'evento di input previsto.
- `public class OrderHandler {...}`: definisce la tua classe C#. Al suo interno, definirai il metodo del gestore principale e qualsiasi altro metodo di supporto.
- `private static readonly AmazonS3Client s3Client = new();`: inizializza un client Amazon S3 con la catena di provider delle credenziali predefinita, al di fuori del metodo del gestore principale. Ciò fa sì che Lambda esegua questo codice durante la fase di inizializzazione.
- `public async ... HandleRequest (Order order, ILambdaContext context)`: questo è il metodo dell'handler principale, che contiene la logica principale dell'applicazione.
- `private async Task UploadReceiptToS3(...) {}`: questo è un metodo helper a cui fa riferimento il metodo dell'handler principale `handleRequest`.

Poiché questa funzione richiede un client SDK Amazon S3, devi aggiungerlo alle dipendenze del progetto. Puoi farlo andando in `src/ExampleCS` e impartendo il comando seguente:

```
dotnet add package AWSSDK.S3
```

Aggiungi informazioni sui metadati a.json aws-lambda-tools-defaults

Per impostazione predefinita, il file `aws-lambda-tools-defaults.json` generato non contiene informazioni `profile` o `region` sulla tua funzione. Inoltre, aggiorna la stringa `function-handler` al valore corretto (`ExampleCS::ExampleLambda.OrderHandler::HandleRequest`). Puoi effettuare questo aggiornamento manualmente e aggiungere i metadati necessari per utilizzare un profilo di credenziali e una regione specifici per la tua funzione. Ad esempio, il file `aws-lambda-tools-defaults.json` dovrebbe essere simile all'esempio seguente:

```
{
  "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"
}
```

Affinché questa funzione funzioni correttamente, il suo ruolo di esecuzione deve consentire l'`s3:PutObject`azione. Inoltre, assicuratevi di definire la variabile di `RECEIPT_BUCKET` ambiente. Dopo una chiamata riuscita, il bucket Amazon S3 dovrebbe contenere un file di ricevuta.

Gestori di librerie di classi

Il codice di esempio principale in questa pagina illustra un gestore di librerie di classi. I gestori di librerie di classi hanno la seguente struttura:

```
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerializer

namespace NAMESPACE;

...

public class CLASSNAME {
    public async Task<string> METHODNAME (...) {
    ...
    }
}
```

Quando crei una funzione Lambda, devi fornire a Lambda informazioni sul gestore della funzione sotto forma di stringa nel campo Gestore. Questo indica a Lambda quale metodo del codice eseguire quando la funzione viene richiamata. In C#, per i gestori di librerie di classi, il formato della stringa del gestore è `ASSEMBLY::TYPE::METHOD`, dove:

- `ASSEMBLY` è il nome del file di assembly .NET per l'applicazione. Se stai usando `Amazon.Lambda.Tools` CLI per creare l'applicazione e non specifichi il nome dell'assembly con la proprietà `AssemblyName` nel file `.csproj`, allora `ASSEMBLY` è semplicemente il nome del file `.csproj`.
- `TYPE` è il nome completo del tipo di gestore, cioè `NAMESPACE.CLASSNAME`.
- `METHOD` è il nome del metodo del gestore principale nel codice, cioè `METHODNAME`.

Per il codice di esempio principale, se l'assembly è denominato `ExampleCS`, la stringa completa del gestore è `ExampleCS::ExampleLambda.OrderHandler::HandleRequest`.

Gestori di assembly eseguibili

È inoltre possibile definire le funzioni Lambda in C# come assembly eseguibile. I gestori di assembly eseguibili utilizzano la funzionalità delle istruzioni di primo livello di C#, in cui il compilatore genera il metodo `Main()` e inserisce il codice della funzione al suo interno. Quando si utilizzano assembly eseguibili, è necessario avviare il runtime Lambda. Per far ciò, utilizza il metodo `LambdaBootstrapBuilder.Create` nel codice. Gli input per questo metodo sono la funzione del gestore principale e il serializzatore Lambda da utilizzare. Di seguito viene illustrato un esempio di gestore di assembly eseguibile in 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)
    };
```

```
};
```

Nel campo Gestore per i gestori di assembly eseguibili, la stringa del gestore che indica a Lambda come eseguire il codice è il nome dell'assembly. In questo esempio è `GetProductHandler`.

Firme del gestore valide per le funzioni C#

In C#, le firme del gestore Lambda valide richiedono da 0 a 2 argomenti. In genere, la firma del gestore ha due argomenti, come mostrato nell'esempio principale:

```
public async Task<string> HandleRequest(Order order, ILambdaContext context)
```

Quando si forniscono due argomenti, il primo argomento deve essere l'input dell'evento e il secondo deve essere l'oggetto del contesto Lambda. Entrambi gli argomenti sono facoltativi. Ad esempio, le seguenti sono anche firme valide di gestori Lambda in C#:

- `public async Task<string> HandleRequest()`
- `public async Task<string> HandleRequest(Order order)`
- `public async Task<string> HandleRequest(ILambdaContext context)`

Oltre alla sintassi di base della firma del gestore, esistono alcune restrizioni aggiuntive:

- Non è possibile utilizzare la parola chiave `unsafe` nella firma del gestore. Tuttavia, puoi utilizzare il contesto `unsafe` all'interno del metodo del gestore e delle sue dipendenze. Per ulteriori informazioni, consulta unsafe (Riferimenti per C#) sul sito Web di Microsoft.
- Il gestore potrebbe non utilizzare la parola chiave `params`, o usare `ArgIterator` come parametro di input o di ritorno. Queste parole chiave supportano un numero variabile di parametri. Il gestore può accettare al massimo due argomenti.
- Il gestore potrebbe non essere un metodo generico. In altre parole, non può utilizzare parametri di tipo generico come <T>.
- Lambda non supporta gestori asincroni con `async void` nella firma.

Convenzioni di denominazione dei gestori

I gestori Lambda in C# non hanno restrizioni di denominazione rigide. Tuttavia, devi assicurarti di fornire la stringa del gestore corretta a Lambda quando implementi la funzione. La stringa del gestore

corretta dipende da se stai implementando un gestore di libreria di classi o un gestore di assembly eseguibile.

Sebbene sia possibile utilizzare qualsiasi nome per il gestore, i nomi delle funzioni in C# sono generalmente in. PascalCase Inoltre, sebbene non sia necessario che il nome del file corrisponda al nome della classe o del gestore, in genere è consigliabile utilizzare un nome di file come `OrderHandler.cs` se il nome della classe è `OrderHandler`. Ad esempio, è possibile modificare il nome del file in questo esempio da `Function.cs` a `OrderHandler.cs`.

Serializzazione nelle funzioni Lambda C#

JSON è il formato di input più comune e standard per le funzioni Lambda. In questo esempio, la funzione prevede un input simile a quanto segue:

```
{
    "orderId": "12345",
    "amount": 199.99,
    "item": "Wireless Headphones"
}
```

In C#, puoi definire la forma dell'evento di input previsto in una classe. In questo esempio, definiamo la classe `Order` per modellare questo input:

```
public class Order
{
    public string OrderId { get; set; } = string.Empty;
    public double Amount { get; set; }
    public string Item { get; set; } = string.Empty;
}
```

Se la funzione Lambda utilizza i tipi di input/output diversi da un oggetto `Stream`, è necessario aggiungere una libreria di serializzazione all'applicazione. Ciò ti consente di convertire l'input JSON in un'istanza della classe definita. Esistono due metodi di serializzazione per le funzioni C# in Lambda: serializzazione basata sulla riflessione e serializzazione generata dal codice sorgente.

Serializzazione basata sulla riflessione

AWS fornisce librerie predefinite che è possibile aggiungere rapidamente all'applicazione. Queste librerie implementano la serializzazione utilizzando la riflessione. Utilizza uno dei seguenti pacchetti per implementare la serializzazione basata sulla riflessione:

- `Amazon.Lambda.Serialization.SystemTextJson`: nel backend, questo pacchetto utilizza `System.Text.Json` per eseguire attività di serializzazione.
- `Amazon.Lambda.Serialization.Json`: nel backend, questo pacchetto utilizza `Newtonsoft.Json` per eseguire attività di serializzazione.

È possibile inoltre creare una libreria di serializzazione personalizzata implementando l'interfaccia `ILambdaSerializer`, disponibile come parte della libreria `Amazon.Lambda.Core`. L'interfaccia definisce due metodi:

- `T Deserialize<T>(Stream requestStream);`

  Questo metodo viene implementato per deserializzare il payload della richiesta dall'API `Invoke` nell'oggetto passato al gestore della funzione Lambda.
- `T Serialize<T>(T response, Stream responseStream);`

  Questo metodo viene implementato per serializzare il risultato restituito dal gestore della funzione Lambda nel payload della risposta restituito dall'operazione dell'API `Invoke`.

L'esempio principale in questa pagina utilizza la serializzazione basata sulla riflessione. La serializzazione basata su Reflection funziona immediatamente AWS Lambda e non richiede alcuna configurazione aggiuntiva, il che la rende una buona scelta in termini di semplicità. Tuttavia, richiede un maggiore utilizzo della memoria funzionale. È inoltre possibile che si verifichino latenze di funzioni più elevate a causa della riflessione del runtime.

Serializzazione generata dal codice sorgente

Con la serializzazione generata dal codice sorgente, il codice di serializzazione viene generato in fase di compilazione. Ciò elimina la necessità di riflessione e può migliorare le prestazioni della tua funzione. Per utilizzare la serializzazione generata dal codice sorgente nella tua funzione, devi procedere come indicato di seguito:

- Crea una nuova classe parziale che eredita da `JsonSerializerContext`, aggiungendo attributi `JsonSerializable` per tutti i tipi che richiedono la serializzazione o la deserializzazione.
- Configura il `LambdaSerializer` per utilizzare un `SourceGeneratorLambdaJsonSerializer<T>`.
- Aggiorna qualsiasi serializzazione e deserializzazione manuale nel codice dell'applicazione per utilizzare la classe appena creata.

L'esempio seguente mostra come puoi modificare l'esempio principale di questa pagina, che utilizza la serializzazione basata sulla riflessione, in modo da utilizzare invece la serializzazione generata dal codice sorgente.

```
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);

    ...

    }

}
```

La serializzazione generata dal codice sorgente richiede più impostazioni rispetto alla serializzazione basata sulla riflessione. Tuttavia, le funzioni che utilizzano la generazione da codice sorgente tendono a utilizzare meno memoria e offrono prestazioni migliori grazie alla generazione di codice in fase di compilazione. Per contribuire a eliminare gli avvii a freddo delle funzioni, prendi in considerazione la possibilità di passare alla serializzazione generata dal codice sorgente.

> **Note**
>
> Se desideri utilizzare la ahead-of-time compilazione nativa (AOT) con Lambda, devi utilizzare la serializzazione generata dal codice sorgente.

Funzioni basate su file

Introdotte in .NET 10, le app basate su file consentono di creare applicazioni.NET da un singolo `.cs` file, senza una `.csproj` struttura di file o directory. Lambda supporta funzioni basate su file, a partire da.NET 10. Offrono un modo semplice e leggero per creare funzioni Lambda in C#.

Il modo più veloce per iniziare a creare una funzione Lambda basata su file C# consiste nell'utilizzare il pacchetto. `Amazon.Lambda.Templates` Per installare il pacchetto, esegui il comando seguente:

```
dotnet new install Amazon.Lambda.Templates
```

Quindi, crea una funzione Lambda di esempio basata su file C#:

```
dotnet new lambda.FileBased -n MyLambdaFunction
```

Le funzioni basate su file utilizzano gestori di assembly eseguibili. È quindi necessario includere il `Amazon.Lambda.RuntimeSupport` NuGet pacchetto e utilizzare il `LambdaBootstrapBuilder.Create` metodo per registrare la funzione di gestore.NET per il tipo di evento e avviare il runtime client .NET Lambda.

Le funzioni basate su file utilizzano.NET Native AOT per impostazione predefinita, che richiede la serializzazione generata dal codice sorgente. È possibile disabilitare Native AOT specificandolo nel file sorgente. `#:property PublishAot=false` Per ulteriori informazioni sull'utilizzo di Native AOT in Lambda, consulta. the section called “Compilazione AOT nativa”

Accesso e utilizzo dell'oggetto contestuale Lambda

L'oggetto contesto: contiene informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. In questo esempio, l'oggetto contesto è di tipo `Amazon.Lambda.Core.ILambdaContext` ed è il secondo argomento della funzione del gestore principale.

```
public async Task<string> HandleRequest(Order order, ILambdaContext context) {
```

```
    ...
}
```

L'oggetto contesto è un input opzionale. Per ulteriori informazioni sulle firme valide accettate del gestore, consulta the section called “Firme del gestore valide per le funzioni C#”.

L'oggetto context è utile per produrre log di funzioni su Amazon CloudWatch. Puoi usare il metodo `context.getLogger()` per ottenere un oggetto `LambdaLogger` per la registrazione. In questo esempio, possiamo usare il logger per registrare il log di un messaggio di errore se l'elaborazione fallisce per qualche motivo:

```
context.Logger.LogError($"Failed to process order: {ex.Message}");
```

Oltre alla registrazione di log, puoi anche utilizzare l'oggetto contesto per il monitoraggio delle funzioni. Per ulteriori informazioni sulla copia di oggetti, consulta la sezione the section called “Contesto”.

Usando la SDK per .NET v3 nel tuo gestore

Spesso, utilizzerai le funzioni Lambda per interagire o aggiornare altre AWS risorse. Il modo più semplice per interfacciarsi con queste risorse è usare la SDK per .NET v3.

> **Note**
>
> La SDK per .NET (v2) è obsoleta. Ti consigliamo di utilizzare solo la versione v3. SDK per .NET

Puoi aggiungere dipendenze SDK al tuo progetto usando il comando `Amazon.Lambda.Tools` seguente:

```
dotnet add package <package_name>
```

Ad esempio, nell'esempio principale di questa pagina, è necessario utilizzare l'API Amazon S3 per caricare una ricevuta su S3. Possiamo importare il client SDK Amazon S3 con il seguente comando:

```
dotnet add package AWSSDK.S3
```

Questo comando aggiunge la dipendenza al tuo progetto. Dovresti anche vedere una riga simile alla seguente nel file `.csproj` del tuo progetto:

```
<PackageReference Include="AWSSDK.S3" Version="3.7.2.18" />
```

Quindi, importa le dipendenze direttamente nel tuo codice C#:

```
using Amazon.S3;
using Amazon.S3.Model;
```

Il codice di esempio inizializza quindi un client Amazon S3 (utilizzando la catena di provider delle credenziali predefinita) come indicato di seguito:

```
private static readonly AmazonS3Client s3Client = new();
```

In questo esempio, abbiamo inizializzato il nostro client Amazon S3 all'esterno della funzione del gestore principale per evitare di doverlo inizializzare ogni volta che invochiamo la nostra funzione. Dopo aver inizializzato il client SDK, puoi utilizzarlo per interagire con altri servizi. AWS Il codice di esempio richiama l'API `PutObject` Amazon S3 nel modo seguente:

```
var putRequest = new PutObjectRequest
{
    BucketName = bucketName,
    Key = key,
    ContentBody = receiptContent,
    ContentType = "text/plain"
};

await s3Client.PutObjectAsync(putRequest);
```

Accesso alle variabili d'ambiente

Nel codice dell'handler, puoi fare riferimento a qualsiasi variabile di ambiente utilizzando il metodo `System.Environment.GetEnvironmentVariable`. In questo esempio, facciamo riferimento alla variabile di ambiente `RECEIPT_BUCKET` definita utilizzando le seguenti righe di codice:

```
string? bucketName = Environment.GetEnvironmentVariable("RECEIPT_BUCKET");
if (string.IsNullOrWhiteSpace(bucketName))
{
```

```
    throw new ArgumentException("RECEIPT_BUCKET environment variable is not set");
}
```

Utilizzo dello stato globale

Lambda esegue il codice statico e il costruttore della classe durante la fase di inizializzazione prima di richiamare la funzione per la prima volta. Le risorse create durante l'inizializzazione restano in memoria tra le invocazioni, in modo da evitare di doverle creare ogni volta che si invoca la funzione.

Nel codice di esempio, il codice di inizializzazione del client S3 non rientra nel metodo del gestore principale. Il runtime inizializza il client prima che la funzione gestisca il suo primo evento, il che può portare a tempi di elaborazione più lunghi. Gli eventi successivi sono molto più veloci perché Lambda non ha bisogno di inizializzare nuovamente il client.

Semplificazione del codice delle funzioni con il framework Lambda Annotations

Lambda Annotations è un framework per .NET 8 che semplifica la scrittura di funzioni Lambda utilizzando C#. Il framework Annotations utilizza generatori di codice sorgente per generare codice che si traduce dal modello di programmazione Lambda al codice semplificato. Con il framework Annotations, è possibile sostituire gran parte del codice in una funzione Lambda scritta utilizzando il normale modello di programmazione. Il codice scritto utilizzando il framework utilizza espressioni più semplici che consentono di concentrarsi sulla logica aziendale. Per alcuni esempi, consulta Amazon.Lambda.Annotations nella documentazione NuGet.

Per un esempio di applicazione completa che utilizza le annotazioni Lambda, consulta PhotoAssetManagerl'esempio nel repository. `awsdocs/aws-doc-sdk-examples` GitHub Il file `Function.cs` principale nella directory `PamApiAnnotations` usa Lambda Annotations. Per fare un confronto, la directory `PamApi` contiene file equivalenti scritti utilizzando il normale modello di programmazione Lambda.

Iniezione delle dipendenze con il framework Lambda Annotations

Puoi utilizzare il framework Lambda Annotations anche per aggiungere l'iniezione di dipendenze alle tue funzioni Lambda utilizzando una sintassi che conosci. Quando aggiungi un attributo `[LambdaStartup]` a un file `Startup.cs`, il framework Lambda Annotations genererà il codice richiesto in fase di compilazione.

```
[LambdaStartup]
```

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddSingleton<IDatabaseRepository, DatabaseRepository>();
    }
}
```

La tua funzione Lambda può iniettare servizi utilizzando l'iniezione del costruttore o iniettandoli in metodi individuali utilizzando l'attributo `[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);
    }
}
```

Best practice di codice per le funzioni Lambda con C#

Segui le linee guida riportate nell'elenco seguente per utilizzare le best practice di codifica durante la creazione delle funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. In questo modo è possibile creare una funzione di cui è più semplice eseguire l'unit test.

- Controllare le dipendenze nel pacchetto di distribuzione della funzione. L'ambiente di esecuzione AWS Lambda contiene diverse librerie. Per abilitare il set di caratteristiche e aggiornamenti della sicurezza più recenti, Lambda aggiorna periodicamente tali librerie. Tali aggiornamenti possono introdurre lievi modifiche al comportamento della funzione Lambda. Per mantenere il controllo completo delle dipendenze utilizzate dalla funzione, inserire tutte le dipendenze nel pacchetto di implementazione.
- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del contesto di esecuzione.
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. In questo modo viene ridotta la quantità di tempo necessaria per il download del pacchetto e per la relativa decompressione prima dell'invocazione. Per le funzioni create in .NET, evitate di caricare l'intera libreria AWS SDK come parte del pacchetto di distribuzione. Utilizzare invece in modo selettivo i moduli che prelevano i componenti dell'SDK necessari (ad esempio i moduli SDK Dynamo DB e Amazon S3 e le librerie di base Lambda).

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedere Riutilizzo delle connessioni con Keep-Alive in Node.js.

Utilizzare le variabili di ambiente per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare

un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su `0` per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare documenti non documentati e non pubblici APIs nel codice della funzione Lambda. Per i runtime AWS Lambda gestiti, Lambda applica periodicamente aggiornamenti di sicurezza e funzionalità all'interno di Lambda. APIs Questi aggiornamenti delle API interne possono essere incompatibili con le versioni precedenti e portare a conseguenze indesiderate, come errori di chiamata se la funzione dipende da questi elementi non pubblici. APIs Consulta il riferimento alle API per un elenco di quelle disponibili al pubblico. APIs

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda?.

Crea e implementa le funzioni Lambda C# con gli archivi di file .zip

Un pacchetto di implementazione .NET (archivio di file .zip) contiene l'assembly compilato della funzione insieme a tutte le dipendenze dell'assembly stesso. Il pacchetto contiene inoltre un file *proj*`.deps.json`, Ciò indica al runtime di .NET tutte le dipendenze della funzione e un file *proj*`.runtimeconfig.json` utilizzato per configurare il runtime.

Per implementare singole funzioni Lambda, puoi utilizzare l'interfaccia a riga di comando globale Lambda .NET `Amazon.Lambda.Tools`. L'utilizzo del comando `dotnet lambda deploy-function` crea automaticamente un pacchetto di implementazione .zip e lo distribuisce su Lambda. Tuttavia, ti consigliamo di utilizzare framework come AWS Serverless Application Model (AWS SAM) o AWS Cloud Development Kit (AWS CDK) per implementare le applicazioni .NET su AWS.

Le applicazioni serverless di solito comprendono una combinazione di funzioni Lambda e altri Servizi AWS gestiti che interagiscono per eseguire una particolare attività aziendale. AWS SAM e AWS CDK semplificano la creazione e l'implementazione di funzioni Lambda con altri Servizi AWS su larga scala. La specifica del modello AWS SAM fornisce una sintassi semplice e chiara per descrivere le funzioni Lambda, le API, le autorizzazioni, le configurazioni e altre risorse AWS che costituiscono l'applicazione serverless. Con il AWS CDK si definisce l'infrastruttura cloud come codice per creare applicazioni affidabili, scalabili e convenienti nel cloud utilizzando linguaggi di programmazione e framework di programmazione moderni come .NET. Sia AWS CDK che AWS SAM utilizzano la CLI globale Lambda di .NET per creare pacchetti di funzioni.

Anche se è possibile utilizzare livelli Lambda con funzioni in C# tramite la CLI di .NET Core, tale scelta non è preferibile. Le funzioni in C# che utilizzano livelli caricano manualmente gli assembly condivisi in memoria durante il Fase di init, per cui i tempi di avvio a freddo possono aumentare. Includi invece tutto il codice condiviso in fase di compilazione per evitare che il caricamento degli assembly al runtime comprometta le prestazioni.

Nelle sezioni seguenti sono riportate le istruzioni per la creazione e l'implementazione di funzioni Lambda .NET tramite AWS SAM, il AWS CDK e la CLI globale di Lambda di .NET.

Argomenti

- Utilizzo della CLI globale Lambda di .NET
- Implementazione di funzioni Lambda C# tramite AWS SAM
- Implementazione di funzioni Lambda C# tramite AWS CDK
- Implementa applicazioni ASP.NET

Utilizzo della CLI globale Lambda di .NET

La CLI .NET e l'estensione degli strumenti globali Lambda .NET (`Amazon.Lambda.Tools`) offrono un modo multi-piattaforma per creare applicazioni Lambda basate su .NET, impacchettarle e implementarle in Lambda. In questa sezione, scoprirai come creare nuovi progetti Lambda .NET utilizzando la CLI .NET e i modelli Amazon Lambda e come impacchettarli e implementarli tramite `Amazon.Lambda.Tools`

Argomenti

- Prerequisiti
- Creazione di progetti .NET con la CLI .NET
- Implementazione di progetti .NET con la CLI .NET
- Utilizzo dei livelli Lambda con la CLI di .NET

Prerequisiti

SDK .NET 8

Se non l'hai già fatto, installa l'SDK .NET 8 e Runtime.

Modelli di progetto .NET di AWS Amazon.Lambda.Templates

Per generare il codice della funzione Lambda, usa il pacchetto Amazon.Lambda.Templates NuGet. Per installare questo pacchetto del modello, esegui il comando riportato:

```
dotnet new install Amazon.Lambda.Templates
```

Strumenti della CLI globale .NET AWSAmazon.Lambda.Tools

Per creare le funzioni Lambda, usa l'estensione degli strumenti globali .NET Amazon.Lambda.Tools. Per installare Amazon.Lambda.Tools, esegui il comando riportato:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Per ulteriori informazioni sull'estensione della CLI .NET Amazon.Lambda.Tools, consulta il repository Estensioni AWS per la CLI .NET su GitHub.

Creazione di progetti .NET con la CLI .NET

Nella CLI .NET, utilizzi il comando `dotnet new` per creare progetti .NET da una riga di comando. Lambda offre modelli aggiuntivi che utilizzano il pacchetto NuGet `Amazon.Lambda.Templates`.

Dopo aver installato questo pacchetto, esegui il comando di seguito per visualizzare un elenco di modelli disponibili.

```
dotnet new list
```

Per esaminare i dettagli relativi a un modello, utilizzare l'opzione `help`. Ad esempio, per visualizzare i dettagli del modello `lambda.EmptyFunction`, emetti il seguente comando.

```
dotnet new lambda.EmptyFunction --help
```

Per creare un modello di base per una funzione Lambda .NET, usa il modello `lambda.EmptyFunction`. In questo modo viene creata una funzione semplice che accetta una stringa come input e la converte in lettere maiuscole utilizzando il metodo `ToUpper`. Questo modello supporta le seguenti opzioni:

- `--name`: il nome della funzione.
- `--region`: la regione AWS in cui creare la funzione.
- `--profile`: il nome di un profilo nel file delle credenziali AWS SDK per .NET. Per ulteriori informazioni sui profili di credenziali in .NET, consulta Configurazione delle credenziali di AWS nella Guida per gli sviluppatori di SDK AWS per .NET.

In questo esempio, creiamo una nuova funzione vuota denominata `myDotnetFunction` utilizzando il profilo e le impostazioni di Regione AWS predefiniti:

```
dotnet new lambda.EmptyFunction --name myDotnetFunction
```

Questo comando crea i seguenti file e directory nella directory di progetto.

```
### myDotnetFunction
    ### src
    #   ### myDotnetFunction
    #       ### Function.cs
    #       ### Readme.md
```

```
    #       ### aws-lambda-tools-defaults.json
    #       ### myDotnetFunction.csproj
    ### test
        ### myDotnetFunction.Tests
            ### FunctionTest.cs
            ### myDotnetFunction.Tests.csproj
```

Nella directory `src/myDotnetFunction` esaminare i file seguenti:

- aws-lambda-tools-defaults.json: file in cui si specificano le opzioni della riga di comando quando si distribuisce la funzione Lambda. Ad esempio:

```
  "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: codice della funzione del gestore Lambda. Si tratta di un modello C# che include la libreria `Amazon.Lambda.Core` e un attributo `LambdaSerializer` predefiniti. Per ulteriori informazioni sui requisiti e sulle opzioni di serializzazione, consulta Serializzazione nelle funzioni Lambda C#. Include anche una funzione di esempio che è possibile modificare per applicare il codice della funzione Lambda.

```
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();
    }
}
```

- myDotnetFunction.csproj: un file MSBuild in cui sono elencati i file e gli assembly che costituiscono l'applicazione.

```
<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: file che consente di documentare la funzione Lambda.

Nella directory `myfunction/test` esaminare i file seguenti:

- myDotnetFunction.Tests.csproj: come indicato in precedenza, si tratta di un file MSBuild in cui sono elencati file e assembly che costituiscono il progetto di test. Si noti anche che nel file è inclusa la libreria `Amazon.Lambda.Core`, che consente di integrare in modo semplice qualsiasi modello Lambda necessario per eseguire il test della funzione.

```
<Project Sdk="Microsoft.NET.Sdk">
   ...

    <PackageReference Include="Amazon.Lambda.Core" Version="2.2.0 " />
   ...
```

- FunctionTest.cs: lo stesso file di modello del codice C# incluso nella directory `src`. Modificare questo file per eseguire il mirroring del codice di produzione della funzione e per eseguirne il test prima di caricare la funzione Lambda in un ambiente di produzione.

```
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);
        }
    }
}
```

Implementazione di progetti .NET con la CLI .NET

Per creare il pacchetto di implementazione e implementarlo in Lambda, utilizzi gli strumenti della CLI `Amazon.Lambda.Tools`. Per implementare la funzione dai file creati nei passaggi precedenti, per prima cosa accedi alla cartella contenente il file `.csproj` della funzione.

```
cd myDotnetFunction/src/myDotnetFunction
```

Per implementare il codice in Lambda come pacchetto di implementazione .zip, emetti il comando seguente. Scegli il nome della tua funzione.

```
dotnet lambda deploy-function myDotnetFunction
```

Durante l'implementazione, la procedura guidata chiede di selezionare un the section called “Ruolo di esecuzione (autorizzazioni per le funzioni per accedere ad altre risorse)”. Per questo esempio, seleziona `lambda_basic_role`.

Dopo aver implementato la funzione, puoi testarla nel cloud con il comando `dotnet lambda invoke-function`. Per il codice di esempio nel modello `lambda.EmptyFunction`, puoi testare la tua funzione inserendo una stringa utilizzando l'opzione `--payload`.

```
dotnet lambda invoke-function myDotnetFunction --payload "Just checking if everything
 is OK"
```

Se la funzione è stata implementata con successo, dovresti vedere un output simile al seguente.

```
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
```

Utilizzo dei livelli Lambda con la CLI di .NET

> **Note**
>
> Anche se è possibile utilizzare livelli con funzioni in .NET, tale scelta non è preferibile. Le funzioni in .NET che utilizzano livelli caricano manualmente gli assembly condivisi in memoria durante la fase `Init`, per cui i tempi di avvio a freddo possono aumentare. Includi, invece, tutto il codice condiviso in fase di compilazione per sfruttare le ottimizzazioni integrate del compilatore .NET.

La CLI di .NET supporta comandi che facilitano la pubblicazione di livelli e l'implementazione di funzioni C# che utilizzano livelli. Per pubblicare un livello in un bucket Amazon S3 specificato, utilizza il comando seguente nella stessa directory del tuo file `.csproj`:

```
dotnet lambda publish-layer <layer_name> --layer-type runtime-package-store --s3-
bucket <s3_bucket_name>
```

Quando implementi la funzione utilizzando la CLI di .NET, quindi, specifica l'ARN utilizzato dal livello nel seguente comando:

```
dotnet lambda deploy-function <function_name> --function-layers arn:aws:lambda:us-
east-1:123456789012:layer:layer-name:1
```

Per un esempio completo di una funzione Hello World, consulta l'esempio blank-csharp-with-layer.

Implementazione di funzioni Lambda C# tramite AWS SAM

AWS Serverless Application Model (AWS SAM) è un toolkit che aiuta a semplificare il processo di creazione ed esecuzione di applicazioni serverless su AWS. Definisci le risorse per la tua applicazione in un modello YAML o JSON e usa l'interfaccia della linea di comando AWS SAM (AWS SAM CLI) per compilare, creare pacchetti e implementare le tue applicazioni. Quando crei una funzione Lambda da un modello AWS SAM, AWS SAM crea automaticamente un pacchetto di implementazione .zip o un'immagine di container con il codice della funzione e le dipendenze specificate.AWS SAM quindi implementa la funzione utilizzando uno stack CloudFormation. Per ulteriori informazioni sull'utilizzo di AWS SAM per creare e implementare funzioni Lambda, consulta la pagina Nozioni di base su AWS SAM nella Guida introduttiva per gli sviluppatori di AWS Serverless Application Model.

La seguente procedura mostra come scaricare, creare e implementare un'applicazione Hello World .NET di esempio con AWS SAM. Questa applicazione di esempio utilizza una funzione Lambda e un endpoint Gateway Amazon API per implementare un backend API di base. Quando invii una richiesta HTTP GET all'endpoint Gateway API, Gateway API richiama la funzione Lambda. La funzione restituisce un messaggio "hello world", insieme all'indirizzo IP dell'istanza della funzione Lambda che elabora la richiesta.

Quando crei e implementi l'applicazione utilizzando AWS SAM, dietro le quinte, la CLI AWS SAM utilizza il comando `dotnet lambda package` per impacchettare i singoli bundle di codici delle funzioni Lambda.

Prerequisiti

SDK .NET 8

Installa l'SDK .NET 8 e Runtime.

CLI AWS SAM versione 1.39 o successiva

Per informazioni su come installare l'ultima versione della CLI AWS SAM, consulta Installazione della CLI AWS SAM.

Implementare un'applicazione AWS SAM di esempio

1. Inizializza l'applicazione utilizzando il modello .NET Hello world con il comando riportato di seguito.

```
sam init --app-template hello-world --name sam-app \
--package-type Zip --runtime dotnet8
```

Questo comando crea i seguenti file e directory nella directory di progetto.

```
### 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. Passa alla directory contenente il `template.yaml file`. Questo file è un modello che definisce le risorse AWS per l'applicazione, tra cui la funzione Lambda e un'API di Gateway API.

```
cd sam-app
```

3. Per creare il codice sorgente dell'applicazione, digita il comando riportato di seguito.

```
sam build
```

4. Per implementare l'applicazione su AWS, emetti il comando riportato di seguito:

```
sam deploy --guided
```

Questo comando impacchetta e implementa l'applicazione con la seguente serie di prompt. Per accettare le opzioni predefinite, premi Invio.

> **Note**
>
> Per HelloWorldFunction potrebbe non avere l'autorizzazione definita, va bene?, assicurati di inserire y.

- Nome dello stack: il nome dello stack da implementare su CloudFormation. Questo nome deve essere univoco per l'Account AWS e la Regione AWS.
- Regione AWS: la Regione AWS in cui desideri implementare l'app.
- Conferma le modifiche prima dell'implementazione: seleziona Sì per rivedere manualmente tutti i set di modifiche prima che AWS SAM implementi modifiche all'applicazione. Selezionando No, la CLI AWS SAM implementa automaticamente le modifiche alle applicazioni.
- Consenti la creazione di ruoli IAM per la CLI SAM: numerosi modelli AWS SAM, incluso quello Hello world in questo esempio, creano ruoli AWS Identity and Access Management (IAM) per

consentire alle funzioni Lambda di accedere ad altri Servizi AWS. Seleziona Sì per fornire l'autorizzazione a distribuire uno stack CloudFormation che crea o modifica i ruoli IAM.

- Disabilita il rollback: per impostazione predefinita, se AWS SAM riporta un errore durante la creazione o l'implementazione dello stack, lo stack torna alla versione precedente. Seleziona No per accettare questa impostazione predefinita.
- Per HelloWorldFunction potrebbe non avere l'autorizzazione definita, va bene?, immetti `y`.
- Salva gli argomenti in samconfig.toml: seleziona Sì per salvare le tue scelte di configurazione. In futuro, potrai eseguire nuovamente `sam deploy` senza parametri per implementare le modifiche all'applicazione.

5. Una volta completata l'implementazione dell'applicazione, la CLI restituisce il nome della risorsa Amazon (ARN) della funzione Lambda Hello World e il ruolo IAM creato per essa. Consente inoltre di visualizzare anche l'endpoint dell'API di Gateway API. Per testare l'applicazione, apri l'endpoint in un browser. Si avrà una risposta simile alla seguente.

```
{"message":"hello world","location":"34.244.135.203"}
```

6. Per eliminare le risorse, emetti il seguente comando. Tieni presente che l'endpoint dell'API che hai creato è un endpoint pubblico accessibile su Internet. È consigliabile eliminare questo endpoint dopo il test.

```
sam delete
```

Passaggi successivi

Per ulteriori informazioni su come utilizzare AWS SAM per creare e implementare le funzioni Lambda con .NET, consulta le seguenti risorse:

- La Guida per gli sviluppatori di AWS Serverless Application Model (AWS SAM)
- Creazione di applicazioni .NET serverless con AWS Lambda e la CLI SAM

Implementazione di funzioni Lambda C# tramite AWS CDK

AWS Cloud Development Kit (AWS CDK) è un framework di sviluppo software open source per definire l'infrastruttura cloud come codice con linguaggi di programmazione e framework di programmazione moderni come .NET. I progetti AWS CDK vengono eseguiti per generare modelli CloudFormation che vengono poi utilizzati per implementare il codice.

Per creare e implementare un'applicazione .NET Hello world di esempio utilizzando ilAWS CDK, segui le istruzioni nelle sezioni seguenti. L'applicazione di esempio implementa un backend di API di base che consiste di un endpoint Gateway API e di una funzione Lambda. Quando si invia una richiesta HTTP GET all'endpoint, Gateway API richiama la funzione Lambda. La funzione restituisce un messaggio "hello world", insieme all'indirizzo IP dell'istanza Lambda che elabora la richiesta.

Prerequisiti

SDK .NET 8

Installa l'SDK .NET 8 e Runtime.

AWS CDK versione 2

Per informazioni su come installare la versione più recente di AWS CDK, consulta la Nozioni di base sul AWS CDK nella Guida per gli sviluppatori di AWS Cloud Development Kit (AWS CDK) v2.

Implementare un'applicazione AWS CDK di esempio

1. Crea una directory di progetto per l'applicazione di esempio e passa ad essa.

```
mkdir hello-world
cd hello-world
```

2. Inizializza una nuova applicazione AWS CDK eseguendo il comando riportato di seguito.

```
cdk init app --language csharp
```

Questo comando crea i seguenti file e directory nella directory di progetto.

```
### README.md
### cdk.json
### src
    ### HelloWorld
    #   ### GlobalSuppressions.cs
    #   ### HelloWorld.csproj
    #   ### HelloWorldStack.cs
    #   ### Program.cs
    ### HelloWorld.sln
```

3. Apri la directory `src` e crea una nuova funzione Lambda utilizzando la CLI .NET. Questa è la funzione che implementerai utilizzando il AWS CDK. In questo esempio, viene creata una funzione Hello world denominata `HelloWorldLambda` utilizzando il modello `lambda.EmptyFunction`.

```
cd src
dotnet new lambda.EmptyFunction -n HelloWorldLambda
```

Dopo questo passaggio, la struttura di directory all'interno della directory del progetto dovrebbe avere un aspetto simile al seguente.

```
### 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. Apri il file `HelloWorldStack.cs` nella directory `src/HelloWorld`. Sostituisci il contenuto del file con il seguente codice.

```
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
                }),
            });
        }
    }
}
```

Questo è il codice per compilare e raggruppare il codice dell'applicazione, nonché la definizione della funzione Lambda stessa. L'oggetto `BundlingOptions` consente di creare un file zip, insieme a una serie di comandi utilizzati per generare il contenuto del file zip. In questa istanza, il comando `dotnet lambda package` viene utilizzato per compilare e generare il file zip.

5. Per implementare l'applicazione, emetti il comando riportato di seguito.

```
cdk deploy
```

6. Invoca la funzione Lambda implementata utilizzando la CLI .NET Lambda.

```
dotnet lambda invoke-function HelloWorldFunction -p "hello world"
```

7. Una volta terminato il test, potrai eliminare le risorse create (a meno che non si desideri mantenerle). Per eliminare le risorse, emetti il seguente comando.

```
cdk destroy
```

Passaggi successivi

Per ulteriori informazioni su come utilizzare AWS CDK per creare e implementare le funzioni Lambda con .NET, consulta le seguenti risorse:

- Operazioni con il CDK AWS in C#
- Crea, impacchetta e pubblica funzioni Lambda C# .NET CDK AWS

Implementa applicazioni ASP.NET

Oltre a ospitare funzioni basate sugli eventi, puoi utilizzare .NET con Lambda anche per ospitare applicazioni ASP.NET leggere. Puoi creare e implementare applicazioni ASP.NET utilizzando il pacchetto `Amazon.Lambda.AspNetCoreServer` NuGet. In questa sezione, imparerai come distribuire un'API Web ASP.NET in Lambda utilizzando gli strumenti della CLI .NET Lambda.

Argomenti

- Prerequisiti
- Implementazione di un'API Web ASP.NET in Lambda
- Implementazione di API minime ASP.NET su Lambda

Prerequisiti

SDK .NET 8

Installa l'SDK .NET 8 e ASP.NET Core Runtime.

Amazon.Lambda.Tools

Per creare le funzioni Lambda, usa l'estensione degli strumenti globali .NET Amazon.Lambda.Tools. Per installare Amazon.Lambda.Tools, esegui il comando riportato:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Per ulteriori informazioni sull'estensione della CLI .NET Amazon.Lambda.Tools, consulta il repository Estensioni AWS per la CLI .NET su GitHub.

Amazon.Lambda.Templates

Per generare il codice della funzione Lambda, usa il pacchetto Amazon.Lambda.Templates NuGet. Per installare questo pacchetto del modello, esegui il comando riportato:

```
dotnet new --install Amazon.Lambda.Templates
```

Implementazione di un'API Web ASP.NET in Lambda

Per implementare un'API Web utilizzando ASP.NET, puoi utilizzare i modelli Lambda .NET per creare un nuovo progetto di API Web. Utilizza il comando seguente per inizializzare un nuovo progetto di API Web ASP.NET. Nel comando di esempio, diamo un nome al progetto `AspNetOnLambda`.

```
dotnet new serverless.AspNetCoreWebAPI -n AspNetOnLambda
```

Questo comando crea i seguenti file e directory nella directory di progetto.

```
.
### 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
```

Quando Lambda richiama la funzione, il punto di ingresso che utilizza è il file `LambdaEntryPoint.cs`. Il file creato dal modello Lambda .NET contiene il seguente codice.

```
namespace AspNetOnLambda;

public class LambdaEntryPoint : Amazon.Lambda.AspNetCoreServer.APIGatewayProxyFunction
{
    protected override void Init(IWebHostBuilder builder)
    {
        builder
            .UseStartup#Startup#();
    }

    protected override void Init(IHostBuilder builder)
    {
    }
}
```

Il punto di ingresso utilizzato da Lambda deve ereditare da una delle tre classi base del pacchetto `Amazon.Lambda.AspNetCoreServer`. Queste tre classi base sono:

- `APIGatewayProxyFunction`
- `APIGatewayHttpApiV2ProxyFunction`
- `ApplicationLoadBalancerFunction`

La classe predefinita utilizzata quando si crea il file `LambdaEntryPoint.cs` utilizzando il modello Lambda .NET fornito è `APIGatewayProxyFunction`. La classe base che usi nella tua funzione dipende dal livello API che precede la tua funzione Lambda.

Ciascuna delle tre classi base contiene un metodo pubblico denominato `FunctionHandlerAsync`. Il nome di questo metodo farà parte della stringa del gestore che Lambda usa per richiamare la tua funzione. Il metodo `FunctionHandlerAsync` trasforma il payload dell'evento in entrata nel formato ASP.NET corretto e la risposta ASP.NET in un payload di risposta Lambda. Per il progetto `AspNetOnLambda` di esempio mostrato, la stringa del gestore sarebbe la seguente.

```
AspNetOnLambda::AspNetOnLambda.LambdaEntryPoint::FunctionHandlerAsync
```

Per implementare l'API in Lambda, emetti i seguenti comandi per navigare nella directory contenente il file del codice sorgente e implementa la funzione utilizzando CloudFormation.

```
cd AspNetOnLambda/src/AspNetOnLambda
dotnet lambda deploy-serverless
```

> **Tip**
>
> Quando implementi un'API utilizzando il comando **`dotnet lambda deploy-serverless`**, CloudFormation assegna alla funzione Lambda un nome basato sul nome dello stack specificato durante l'implementazione. Per assegnare un nome personalizzato alla funzione Lambda, modifica il file `serverless.template` in modo da aggiungere una proprietà `FunctionName` alla risorsa `AWS::Serverless::Function`. Consulta Tipo di nome nella Guida per l'utente di CloudFormation per ulteriori informazioni.

Implementazione di API minime ASP.NET su Lambda

Per implementare un'API minima ASP.NET, puoi utilizzare i modelli Lambda .NET per creare un nuovo progetto di API minima. Utilizza il comando seguente per inizializzare un nuovo progetto di API minima ASP.NET. In questo esempio, al progetto assegnamo il nome `MinimalApiOnLambda`.

```
dotnet new serverless.AspNetCoreMinimalAPI -n MinimalApiOnLambda
```

Questo comando crea i seguenti file e directory nella directory di progetto.

```
### MinimalApiOnLambda
    ### src
        ### MinimalApiOnLambda
            ### Controllers
            #   ### CalculatorController.cs
            ### MinimalApiOnLambda.csproj
            ### Program.cs
            ### Readme.md
            ### appsettings.Development.json
            ### appsettings.json
            ### aws-lambda-tools-defaults.json
            ### serverless.template
```

Il file `Program.cs` contiene il seguente codice.

```
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();
```

Per configurare l'API minima da eseguire su Lambda, potrebbe essere necessario modificare questo codice in modo che le richieste e le risposte tra Lambda e ASP.NET Core vengano tradotte correttamente. Per impostazione predefinita, la funzione è configurata per un'origine di eventi della REST API. Per un'API HTTP o un Application Load Balancer, sostituisci `(LambdaEventSource.RestApi)` con una delle seguenti opzioni:

- `(LambdaEventSource.HttpAPi)`
- `(LambdaEventSource.ApplicationLoadBalancer)`

Per implementare l'API minima in Lambda, emetti i seguenti comandi per navigare nella directory contenente il file del codice sorgente e implementa la funzione utilizzando CloudFormation.

```
cd MinimalApiOnLambda/src/MinimalApiOnLambda
dotnet lambda deploy-serverless
```

Utilizzo dei livelli per le funzioni Lambda .NET

Consigliamo di non utilizzare i livelli per gestire le dipendenze per le funzioni Lambda scritte in .NET. Il motivo è che .NET è un linguaggio compilato, e le funzioni devono comunque caricare manualmente gli assembly condivisi in memoria durante la fase di inizializzazione, per cui i tempi di avvio a freddo possono aumentare. L'uso dei livelli non solo complica il processo di implementazione, ma impedisce anche di sfruttare le ottimizzazioni integrate del compilatore.

Per utilizzare dipendenze esterne con i gestori .NET, includile direttamente nel pacchetto di implementazione durante la fase di compilazione. In questo modo, semplifichi il processo di implementazione e sfrutti anche le ottimizzazioni integrate del compilatore .NET. Per un esempio di come importare e utilizzare dipendenze come pacchetti NuGet nella tua funzione, consulta the section called “Gestore”.

Distribuzione delle funzioni .NET Lambda con immagini di container

Esistono tre modi per creare un'immagine di container per una funzione Lambda in .NET:

- Utilizzo di un'immagine AWS di base per.NET

  Le immagini di base AWS sono precaricate con un runtime in linguaggio, un client di interfaccia di runtime per gestire l'interazione tra Lambda e il codice della funzione e un emulatore di interfaccia di runtime per i test locali.
- Utilizzo di un'immagine di AWS base solo per il sistema operativo

  AWS Le immagini di base solo per il sistema operativo contengono una distribuzione Amazon Linux e l'emulatore di interfaccia di runtime. Queste immagini vengono comunemente utilizzate per creare immagini di container per linguaggi compilati, come Go e Rust, e per un linguaggio o una versione di linguaggio per cui Lambda non fornisce un'immagine di base, come Node.js 19. Puoi anche utilizzare immagini di base solo per il sistema operativo al fine di implementare un runtime personalizzato. Per rendere l'immagine compatibile con Lambda, devi includere il client di interfaccia di runtime per .NET nell'immagine.
- Utilizzo AWS di un'immagine non di base

  È possibile utilizzare un'immagine di base alternativa da un altro registro del container, come ad esempio Alpine Linux o Debian. Puoi anche utilizzare un'immagine personalizzata creata dalla tua organizzazione. Per rendere l'immagine compatibile con Lambda, devi includere il client di interfaccia di runtime per .NET nell'immagine.

> **Tip**
>
> Per ridurre il tempo necessario all'attivazione delle funzioni del container Lambda, consulta Utilizzo di compilazioni a più fasi nella documentazione Docker. Per creare immagini di container efficienti, segui le best practice per scrivere file Docker.

Questa pagina spiega come creare, testare e implementare le immagini di container per Lambda.

Argomenti

- AWS immagini di base per.NET
- Utilizzo di un'immagine AWS di base per.NET

- Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

AWS immagini di base per.NET

AWS fornisce le seguenti immagini di base per.NET:

Tag	Runtime	Sistema operativo	Dockerfile	Raggiunta obsolescenza
10	.NET 10	Amazon Linux 2023	Dockerfile per.NET 10 su GitHub	14 novembre 2028
9	.NET 9	Amazon Linux 2023	Dockerfile per.NET 9 su GitHub	10 novembre 2026
8	.NET 8	Amazon Linux 2023	Dockerfile per.NET 8 su GitHub	10 novembre 2026

Archivio Amazon ECR: gallery.ecr. aws/lambda/dotnet

Utilizzo di un'immagine AWS di base per.NET

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- SDK .NET: i passaggi seguenti utilizzano l'immagine di base .NET 8. Assicurati che la tua versione di .NET corrisponda alla versione dell'immagine di base specificata nel tuo Dockerfile.
- Docker (versione minima 25.0.0)
- Il plugin buildx di Docker.

Creazione e implementazione di un'immagine utilizzando un'immagine di base

Nei passaggi seguenti, utilizzerai Amazon.Lambda.Templates e Amazon.Lambda.Tools per creare un progetto .NET. Quindi, crei un'immagine Docker, carichi l'immagine su Amazon ECR e la implementi su una funzione Lambda.

1. Installa il pacchetto Amazon.Lambda.Templates. NuGet

```
dotnet new install Amazon.Lambda.Templates
```

2. Crea un progetto .NET utilizzando il modello `lambda.image.EmptyFunction`.

```
dotnet new lambda.image.EmptyFunction --name MyFunction --region us-east-1
```

I file di progetto sono archiviati nella directory *MyFunction*`/src/`*MyFunction*:

- aws-lambda-tools-defaults.json: specifica le opzioni della riga di comando per la distribuzione della funzione Lambda.
- Function.cs: codice della funzione del gestore Lambda. Si tratta di un modello C# che include la libreria `Amazon.Lambda.Core` e un attributo `LambdaSerializer` predefiniti. Per ulteriori informazioni sui requisiti e sulle opzioni di serializzazione, consulta la pagina Serializzazione nelle funzioni Lambda C#. A fini di test, puoi utilizzare il codice fornito o sostituirlo con il tuo codice personalizzato.
- MyFunction.csproj: un file di progetto.NET, che elenca i file e gli assembly che compongono l'applicazione.
- Dockerfile: a fini di test, puoi utilizzare il Dockerfile fornito o sostituirlo con un file personalizzato. Se utilizzi un file personalizzato, assicurati di:
  - Imposta la proprietà `FROM` sull'URI dell'immagine di base. L'immagine di base e la `TargetFramework` nel file `MyFunction.csproj` devono utilizzare entrambe la stessa versione di .NET. Ad esempio, per utilizzare .NET 9:
    - File Docker: `FROM` *public.ecr.aws/lambda/dotnet:9*
    - MyFunction.csproj: `<TargetFramework>`*net9.0*`</TargetFramework>`
  - Imposta l'argomento `CMD` specificando il gestore della funzione Lambda. Questo dovrebbe corrispondere a `image-command` in `aws-lambda-tools-defaults.json`.

3. Installa lo strumento globale .NET Amazon.Lambda.Tools.

```
dotnet tool install -g Amazon.Lambda.Tools
```

Se Amazon.Lambda.Tools è già installato, assicurati di disporre della versione più recente.

```
dotnet tool update -g Amazon.Lambda.Tools
```

4. Passa alla directory *MyFunction*/src/*MyFunction*, se non lo hai ancora fatto.

```
cd src/MyFunction
```

5. Utilizza Amazon.Lambda.Tools per creare l'immagine Docker, trasferirla in un nuovo repository Amazon ECR e implementare la funzione Lambda.

   Per `--function-role`, specifica il nome del ruolo, non il nome della risorsa Amazon (ARN), del ruolo di esecuzione della funzione. Ad esempio, `lambda-role`.

```
dotnet lambda deploy-function MyFunction --function-role lambda-role
```

   Per ulteriori informazioni sullo strumento globale Amazon.Lambda.Tools, consulta l'archivio Extensions AWS for .NET CLI su. GitHub

6. Richiama la funzione.

```
dotnet lambda invoke-function MyFunction --payload "Testing the function"
```

   in caso di esito positivo, vedrai una risposta simile alla seguente:

```
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. Elimina la funzione Lambda.

```
dotnet lambda delete-function MyFunction
```

Utilizzo di un'immagine di base alternativa con il client di interfaccia di runtime

Se utilizzi un'immagine di base solo per il sistema operativo o un'immagine di base alternativa, devi includere il client dell'interfaccia di runtime nell'immagine. Il client dell'interfaccia di runtime estende l'API Runtime, che gestisce l'interazione tra Lambda e il codice della funzione.

L'esempio seguente mostra come creare un'immagine contenitore per.NET utilizzando un'immagine non di AWS base e come aggiungere Amazon.Lambda. RuntimeSupport pacchetto, che è il client dell'interfaccia runtime Lambda per .NET. Il Dockerfile di esempio utilizza l'immagine di base Microsoft .NET 8.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- .NET SDK: i passaggi seguenti utilizzano un'immagine di base.NET 9. Assicurati che la tua versione di .NET corrisponda alla versione dell'immagine di base specificata nel tuo Dockerfile.
- Docker (versione minima 25.0.0)
- Il plugin buildx di Docker.

Creazione e implementazione di un'immagine utilizzando un'immagine di base alternativa

1. Installa il pacchetto NuGet Amazon.Lambda.Templates.

   ```
   dotnet new install Amazon.Lambda.Templates
   ```

2. Crea un progetto .NET utilizzando il modello `lambda.CustomRuntimeFunction`. Questo modello include Amazon.Lambda. RuntimeSupportpacchetto.

   ```
   dotnet new lambda.CustomRuntimeFunction --name MyFunction --region us-east-1
   ```

3. Passa alla directory *MyFunction*`/src/`*MyFunction*. Qui sono archiviati i file del progetto. Esamina i seguenti file di log:

   - aws-lambda-tools-defaults.json: in questo file si specificano le opzioni della riga di comando quando si distribuisce la funzione Lambda.

- Function.cs: il codice contiene una classe con un metodo `Main` che inizializza la libreria `Amazon.Lambda.RuntimeSupport` come bootstrap. Il metodo `Main` sarà il punto di ingresso per il processo della funzione. Il metodo `Main` racchiude il gestore della funzione in un wrapper con cui il bootstrap può funzionare. Per ulteriori informazioni, consulta Using Amazon.Lambda. RuntimeSupport come libreria di classi nel repository. GitHub
- MyFunction.csproj — Un file di progetto.NET, che elenca i file e gli assembly che compongono l'applicazione.
- Readme.md: questo file contiene ulteriori informazioni sulla funzione Lambda di esempio.

4. Apri il file `aws-lambda-tools-defaults.json` e aggiungi le righe seguenti.

```
  "package-type": "image",
  "docker-host-build-output-dir": "./bin/Release/lambda-publish"
```

- package-type: definisce il pacchetto di implementazione come immagine di container.
- docker-host-build-output-dir: imposta la directory di output per il processo di compilazione.

Example aws-lambda-tools-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. Crea un Dockerfile nella directory *MyFunction*/src/*MyFunction*. Il seguente Dockerfile di esempio utilizza un'immagine di base Microsoft .NET anziché un'immagine di base AWS.

   - Imposta la proprietà FROM sull'identificativo dell'immagine di base. L'immagine di base e la TargetFramework nel file MyFunction.csproj devono utilizzare entrambe la stessa versione di .NET.
   - Utilizza il comando COPY per copiare la funzione nella directory /var/task.
   - Imposta l'ENTRYPOINT sul modulo su cui desideri che il container Docker venga eseguito all'avvio. In questo caso, il modulo è il bootstrap, che inizializza la libreria Amazon.Lambda.RuntimeSupport.

   Nota che l'esempio Dockerfile non include un'istruzione USER. Quando implementi un'immagine di container su Lambda, Lambda definisce automaticamente un utente Linux predefinito con autorizzazioni con privilegi minimi. Questo è diverso dal comportamento standard di Docker, che per impostazione predefinita è l'utente root quando non viene fornita alcuna istruzione USER.

   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. Installa lo strumento globale .NET Core Amazon.Lambda.Tools.

```
dotnet tool install -g Amazon.Lambda.Tools
```

   Se Amazon.Lambda.Tools è già installato, assicurati di disporre della versione più recente.

```
dotnet tool update -g Amazon.Lambda.Tools
```

7. Utilizza Amazon.Lambda.Tools per creare l'immagine Docker, trasferirla in un nuovo repository Amazon ECR e implementare la funzione Lambda.

   Per `--function-role`, specifica il nome del ruolo, non il nome della risorsa Amazon (ARN), del ruolo di esecuzione della funzione. Ad esempio, `lambda-role`.

   ```
   dotnet lambda deploy-function MyFunction --function-role lambda-role
   ```

   Per ulteriori informazioni sull'estensione CLI di Amazon.Lambda.Tools, consulta l'archivio Extensions for .NET CLI su.AWS GitHub

8. Richiama la funzione.

   ```
   dotnet lambda invoke-function MyFunction --payload "Testing the function"
   ```

   In caso di esito positivo, viene visualizzato quanto segue:

   ```
   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. Elimina la funzione Lambda.

   ```
   dotnet lambda delete-function MyFunction
   ```

Compilare il codice della funzione Lambda .NET in un formato di runtime nativo

.NET 8 supporta la compilazione nativa in anticipo (AOT). Con AOT nativa, puoi compilare il codice della funzione Lambda in un formato di runtime nativo, che elimina la necessità di compilare il codice .NET in fase di runtime. La compilazione AOT nativa può ridurre il tempo di avvio a freddo delle funzioni Lambda scritte in .NET. Per ulteriori informazioni, consulta il post Introduzione del runtime .NET 8 per AWS Lambda sul blog AWS Compute.

Sezioni

- Runtime Lambda
- Prerequisiti
- Nozioni di base
- Serializzazione
- Rifinitura
- Risoluzione dei problemi

Runtime Lambda

Per implementare una funzione Lambda creata con la compilazione AOT nativa, usa il runtime Lambda .NET 8 gestito. Questo runtime supporta l'uso sia di architetture x86_64 che di architetture arm64.

Quando si implementa una funzione Lambda .NET, l'applicazione viene prima compilata in codice IL (Intermediate Language). Al runtime, il compilatore just-in-time (JIT) nel runtime Lambda prende il codice IL e lo compila in codice macchina secondo le necessità. Con una funzione Lambda compilata in anticipo con AOT nativo, si compila il codice in codice macchina quando si implementa la funzione in modo da non dipendere dal runtime .NET o dall'SDK nel runtime Lambda per compilare il codice prima che venga eseguito.

Una limitazione di AOT è che il codice dell'applicazione deve essere compilato in un ambiente con lo stesso sistema operativo Amazon Linux 2023 (AL2023) utilizzando dal runtime .NET 8. La CLI Lambda .NET offre funzionalità per compilare l'applicazione in un container Docker utilizzando un'immagine AL2023.

Per evitare potenziali problemi di compatibilità tra le architetture, consigliamo vivamente di compilare il codice in un ambiente con la stessa architettura di processore configurata per la funzione. Per ulteriori informazioni sui limiti della compilazione tra architetture diverse, consulta Compilazione incrociata nella documentazione di Microsoft .NET.

Prerequisiti

Docker

Per utilizzare l'AOT nativo, il codice della funzione deve essere compilato in un ambiente con lo stesso sistema operativo AL2023 del runtime .NET 8. I comandi della CLI .NET nelle seguenti sezioni utilizzano Docker per sviluppare e creare funzioni Lambda in un ambiente AL2023.

SDK .NET 8

La compilazione AOT nativa è una funzionalità di .NET 8. È necessario installare l'SDK .NET 8 sulla macchina di compilazione, non solo il runtime.

Amazon.Lambda.Tools

Per creare le funzioni Lambda, usa l'estensione degli strumenti globali .NET Amazon.Lambda.Tools. Per installare Amazon.Lambda.Tools, esegui il comando riportato:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Per ulteriori informazioni sull'estensione della CLI .NET Amazon.Lambda.Tools, consulta il repository Estensioni AWS per la CLI .NET su GitHub.

Amazon.Lambda.Templates

Per generare il codice della funzione Lambda, usa il pacchetto Amazon.Lambda.Templates NuGet. Per installare questo pacchetto del modello, esegui il comando riportato:

```
dotnet new install Amazon.Lambda.Templates
```

Nozioni di base

Sia la CLI globale .NET che AWS Serverless Application Model (AWS SAM) forniscono modelli introduttivi per la creazione di applicazioni utilizzando AOT nativo. Per creare la tua prima funzione Lambda AOT nativa, completa le operazioni riportate nelle seguenti istruzioni.

Inizializzazione e distribuzione di una funzione Lambda compilata in AOT nativa

1. Inizializza un nuovo progetto utilizzando il modello AOT nativo, quindi naviga nella directory contenente i file `.cs` e `.csproj` creati. In questo esempio, diamo un nome alla nostra funzione `NativeAotSample`.

```
dotnet new lambda.NativeAOT -n NativeAotSample
cd ./NativeAotSample/src/NativeAotSample
```

   Il file `Function.cs` creato dal modello AOT nativo contiene il seguente codice di funzione.

```
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
```

AOT nativo compila l'applicazione in un unico binario nativo. Il punto di ingresso di quel binario è il metodo `static Main`. All'interno di `static Main`, viene avviato il runtime Lambda e viene impostato il metodo `FunctionHandler`. Come parte del bootstrap di runtime, un serializzatore generato dal codice sorgente viene configurato utilizzando `new SourceGeneratorLambdaJsonSerializer<LambdaFunctionJsonSerializerContext>()`

2. Per distribuire l'applicazione in Lambda, assicurati che Docker sia in esecuzione nel tuo ambiente locale ed esegui il comando riportato.

```
dotnet lambda deploy-function
```

Dietro le quinte, la CLI globale .NET scarica un'immagine Docker di AL2023 e compila il codice dell'applicazione all'interno di un container in esecuzione. Il file binario compilato viene restituito al file system locale prima di essere implementato su Lambda.

3. Verifica la tua funzione eseguendo il comando riportato. Sostituisci `<FUNCTION_NAME>` con il nome scelto per la funzione nella procedura guidata di implementazione.

```
dotnet lambda invoke-function <FUNCTION_NAME> --payload "hello world"
```

La risposta della CLI include dettagli sulle prestazioni per l'avvio a freddo (durata di inizializzazione) e il runtime totale per l'invocazione della funzione.

4. Per eliminare le risorse AWS create seguendo i passaggi precedenti, esegui il comando riportato. Sostituisci `<FUNCTION_NAME>` con il nome scelto per la funzione nella procedura guidata di implementazione. Eliminando le risorse AWS che non si utilizzano più, è possibile evitare addebiti superflui sul proprio Account AWS.

```
dotnet lambda delete-function <FUNCTION_NAME>
```

Serializzazione

Per implementare le funzioni in Lambda utilizzando AOT nativo, il codice della funzione deve utilizzare la serializzazione generata dal codice sorgente. Invece di utilizzare la riflessione in fase

di esecuzione per raccogliere i metadati necessari per accedere alle proprietà degli oggetti per la serializzazione, i generatori di codice sorgente generano file sorgente C# che vengono compilati durante la creazione dell'applicazione. Per configurare correttamente il serializzatore generato dal codice sorgente, assicurati di includere tutti gli oggetti di input e output utilizzati dalla funzione, nonché tutti i tipi personalizzati. Ad esempio, una funzione Lambda che riceve eventi da una REST API di Gateway API e restituisce un tipo `Product` personalizzato includerebbe un serializzatore definito come segue.

```
[JsonSerializable(typeof(APIGatewayProxyRequest))]
[JsonSerializable(typeof(APIGatewayProxyResponse))]
[JsonSerializable(typeof(Product))]
public partial class CustomSerializer : JsonSerializerContext
{
}
```

Rifinitura

L'AOT nativo rifinisce il codice dell'applicazione come parte della compilazione per garantire che il file binario sia il più piccolo possibile. .NET 8 for Lambda offre un supporto di rifinitura migliorato rispetto alle versioni precedenti di .NET. È stato aggiunto il supporto alle librerie di runtime Lambda, all'SDK AWS .NET alle annotazioni Lambda .NET e a .NET 8 stesso.

Questi miglioramenti offrono la possibilità di eliminare gli avvisi di rifinitura in fase di compilazione, ma .NET non sarà mai completamente esente da tale problema. Ciò significa che parti delle librerie su cui si basa la funzione possono essere eliminate durante la fase di compilazione. Puoi gestire questo problema definendo `TrimmerRootAssemblies` come parte del tuo file `.csproj`, come mostrato nell'esempio seguente.

```
<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>
```

Tieni presente che quando ricevi un avviso di rifinitura, l'aggiunta della classe che genera l'avviso `TrimmerRootAssembly` potrebbe non risolvere il problema. Un avviso di rifinitura indica che la

classe sta provando ad accedere a un'altra classe che non può essere determinata fino al runtime. Per evitare errori di runtime, aggiungi questa seconda classe a `TrimmerRootAssembly`.

Per ulteriori informazioni sulla gestione degli avvisi di rifinitura, consulta Introduzione agli avvisi di rifinitura nella documentazione di Microsoft .NET.

Risoluzione dei problemi

Errore: la compilazione nativa tra sistemi operativi non è supportata.

> La tua versione dello strumento globale .NET Core di Amazon.Lambda.Tools non è aggiornata. Esegui l'aggiornamento alla versione più recente e riprova.

Docker: il sistema operativo di immagini "linux" non può essere utilizzato su questa piattaforma.

> Docker sul tuo sistema è configurato per utilizzare container Windows. Passa ai container Linux per eseguire l'ambiente della compilazione AOT nativa.

Per ulteriori informazioni sugli errori più comuni, consulta il repository AWSNativeAOT per .NET su GitHub.

Utilizzo dell'oggetto del contesto Lambda per recuperare le informazioni sulla funzione C#

Quando Lambda esegue la funzione, passa un oggetto Context al gestore. Questo oggetto fornisce proprietà con informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

Proprietà del contesto

- `FunctionName`: il nome della funzione Lambda.
- `FunctionVersion`: la versione della funzione.
- `InvokedFunctionArn`: l'Amazon Resource Name (ARN) utilizzato per richiamare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `MemoryLimitInMB`: la quantità di memoria allocata per la funzione.
- `AwsRequestId`: l'identificatore della richiesta di invocazione.
- `LogGroupName`: il gruppo di log per la funzione.
- `LogStreamName`: il flusso di log per l'istanza della funzione.
- `RemainingTime` (`TimeSpan`): il numero di millisecondi rimasti prima del timeout dell'esecuzione.
- `Identity`: (app per dispositivi mobili) Informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
- `ClientContext`: (app per dispositivi mobili) Contesto client fornito a Lambda dall'applicazione client.
- `Logger` L'oggetto logger per la funzione.

Puoi utilizzare le informazioni riportate nell'oggetto `ILambdaContext` per generare informazioni sull'invocazione della funzione a scopo di monitoraggio. Il seguente codice fornisce un esempio di come aggiungere informazioni di contesto a un framework di registrazione strutturato. In questo esempio, la funzione aggiunge `AwsRequestId` agli output di log. La funzione utilizza anche la proprietà `RemainingTime` per annullare un'attività in transito se il timeout della funzione Lambda sta per essere raggiunto.

```
[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();
        }
    }
```

```
}
```

Registrare e monitorare le funzioni Lambda con C#

AWS Lambda monitora automaticamente le funzioni Lambda e invia le voci di registro ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della funzione. L'ambiente di runtime di Lambda invia al flusso di log i dettagli su ogni invocazione e altri output dal codice della funzione. Per ulteriori informazioni sui CloudWatch registri, consulta. Lambda invia automaticamente i log delle funzioni a CloudWatch Logs.

Sections

- Creazione di una funzione che restituisce i registri
- Utilizzo dei controlli di registrazione avanzati di Lambda con .NET
- Strumenti e librerie di registrazione aggiuntivi
- Utilizzo di Powertools per AWS Lambda (.NET) e per la registrazione strutturata AWS SAM
- Visualizzazione dei log nella console Lambda
- Visualizzazione dei log nella console CloudWatch
- Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI
- Eliminazione dei log

Creazione di una funzione che restituisce i registri

Per generare i log dal codice della funzione, potete utilizzare il ILambdaLogger sull'oggetto context, i metodi sulla classe Console o qualsiasi libreria di registrazione che scrive su o. `stdout` `stderr`

Il runtime di .NET registra `START`, `END` e `REPORT` per ogni chiamata. La riga del report fornisce i seguenti dettagli.

Campi dati della riga REPORT

- RequestId— L'ID univoco della richiesta per la chiamata.
- Durata – La quantità di tempo che il metodo del gestore della funzione impiega durante l'elaborazione dell'evento.
- Durata fatturata – La quantità di tempo fatturata per la chiamata.
- Dimensioni memoria – La quantità di memoria allocata per la funzione.

- Quantità max utilizzata – La quantità di memoria utilizzata dalla funzione. Quando le invocazioni condividono un ambiente di esecuzione, Lambda riporta la memoria massima utilizzata in tutte le invocazioni. Questo comportamento potrebbe comportare un valore riportato superiore al previsto.
- Durata Init – Per la prima richiesta servita, la quantità di tempo impiegato dal runtime per caricare la funzione ed eseguire il codice al di fuori del metodo del gestore.
- XRAY TraceId — Per le richieste tracciate, l'ID di traccia.AWS X-Ray
- SegmentId— Per le richieste tracciate, l'ID del segmento X-Ray.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

Utilizzo dei controlli di registrazione avanzati di Lambda con .NET

Per avere un maggiore controllo sul modo in cui i log delle tue funzioni vengono acquisiti, elaborati e utilizzati, puoi configurare le seguenti opzioni di registrazione per i runtime .NET supportati:

- Formato di log: scegli tra i formati di testo normale e JSON strutturato per i log della funzione
- Livello di registro: per i log in formato JSON, scegli il livello di dettaglio dei log a cui Lambda invia, CloudWatch ad esempio ERROR, DEBUG o INFO
- Gruppo di log: scegli il gruppo di log a cui la CloudWatch funzione invia i log

Per ulteriori informazioni su queste opzioni di registrazione e istruzioni su come configurare la funzione per utilizzarle, consulta la pagina the section called “Configurazione dei controlli di registrazione avanzati per le funzioni Lambda”.

Per utilizzare le opzioni del formato di log e del livello di log con le funzioni Lambda in .NET, consulta le istruzioni nelle sezioni seguenti.

Utilizzo del formato di log JSON strutturato con .NET

Se si seleziona JSON per il formato di registro della funzione, Lambda invierà l'output dei log ILambdautilizzando Logger come JSON strutturato. Ogni oggetto di log JSON contiene almeno cinque coppie chiave-valore con le seguenti chiavi:

- `"timestamp"`: l'ora in cui è stato generato il messaggio di log
- `"level"`: il livello di log assegnato al messaggio
- `"requestId"`: l'ID di richiesta univoco dell'invocazione alla funzione

- `"traceId"`: la variabile di ambiente `_X_AMZN_TRACE_ID`
- `"message"`: il contenuto del messaggio di log

L'istanza `ILambdaLogger` può aggiungere ulteriori coppie chiave-valore, ad esempio durante la registrazione delle eccezioni. È inoltre possibile fornire parametri aggiuntivi personalizzati come descritto nella sezione the section called “Parametri di log forniti dal cliente”.

> **Note**
>
> Se il codice utilizza già un'altra libreria di registrazione per generare log in formato JSON, assicurati che il formato di log della funzione sia impostato su testo semplice. L'impostazione del formato di log su JSON comporterà la doppia codifica degli output dei log.

Il seguente comando di registrazione di esempio mostra come scrivere un messaggio di log con il livello `INFO`.

Example codice di registrazione .NET

```
context.Logger.LogInformation("Fetching cart from database");
```

Inoltre, puoi usare un metodo di log generico che utilizza il livello di log come argomento come mostrato nell'esempio seguente.

```
context.Logger.Log(LogLevel.Information, "Fetching cart from database");
```

L'output di registro di questi frammenti di codice di esempio verrebbe acquisito in Logs come segue: CloudWatch

Example Record di 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**
>
> Se configuri il formato di log della funzione per utilizzare testo semplice anziché JSON, il livello di log acquisito nel messaggio segue la convenzione di Microsoft di utilizzare un'etichetta di quattro caratteri. Ad esempio, un livello di log di `Debug` è rappresentato nel messaggio come `dbug`.
> Quando si configura la funzione per l'utilizzo di log in formato JSON, il livello di log acquisito nel log utilizza l'etichetta completa, come mostrato nel record di log JSON di esempio.

Se non assegni un livello all'output log, Lambda gli assegnerà automaticamente il livello INFO.

Registrazione delle eccezioni in JSON

Quando si utilizza la registrazione JSON strutturata con `ILambdaLogger`, è possibile registrare le eccezioni dei log nel codice come illustrato nell'esempio seguente.

Example utilizzo della registrazione delle eccezioni

```
try
{
    connection.ExecuteQuery(query);
}
catch(Exception e)
{
    context.Logger.LogWarning(e, "Error executing query");
}
```

Il formato di log generato da questo codice è mostrato nell'esempio JSON seguente. Si noti che la proprietà `message` in JSON viene compilata utilizzando l'argomento del messaggio fornito nella chiamata `LogWarning`, mentre la proprietà `errorMessage` proviene dalla proprietà `Message` dell'eccezione stessa.

Example Record di 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>"]
}
```

Se il formato di registrazione della funzione è impostato su JSON, Lambda emette anche messaggi di log in formato JSON quando il codice genera un'eccezione non rilevata. Il frammento di codice e il messaggio di log di esempio seguenti mostrano come vengono registrate le eccezioni non rilevate.

Example codice di eccezione

```
throw new ApplicationException("Invalid data");
```

Example Record di 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>"]
}
```

Parametri di log forniti dal cliente

Con i messaggi di log in formato JSON, è possibile fornire parametri di log aggiuntivi e includerli nel log `message`. Il seguente esempio di frammento di codice mostra un comando per aggiungere due parametri forniti dall'utente etichettati `retryAttempt` e `uri`. Nell'esempio, il valore di questi parametri deriva dagli argomenti `retryAttempt` e `uriDestination` passati al comando di registrazione.

Example Comando di registrazione JSON con parametri aggiuntivi

```
context.Logger.LogInformation("Starting retry {retryAttempt} to make GET request to
 {uri}", retryAttempt, uriDestination);
```

L'output del messaggio di log di questo comando è mostrato nell'esempio JSON seguente.

Example Record di 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

Quando si specificano parametri aggiuntivi, è inoltre possibile utilizzare proprietà posizionali anziché nomi. A titolo illustrativo, il comando di registrazione nell'esempio precedente potrebbe essere scritto anche come segue:

```
context.Logger.LogInformation("Starting retry {0} to make GET request to {1}",
 retryAttempt, uriDestination);
```

Tieni presente che quando fornisci parametri di registrazione aggiuntivi, Lambda li acquisisce come proprietà di primo livello nel record di log JSON. Questo approccio è diverso da alcune popolari librerie di registrazione .NET come `Serilog`, che acquisiscono parametri aggiuntivi in un oggetto secondario separato.

Se l'argomento fornito per un parametro aggiuntivo è un oggetto complesso, per impostazione predefinita Lambda utilizza il metodo `ToString()` per fornire il valore. Per indicare che un argomento deve essere serializzato in JSON, utilizza il prefisso `@` come mostrato nel seguente frammento di codice. In questo esempio, `User` è un oggetto con proprietà `FirstName` e `LastName`.

Example Comando di registrazione JSON con oggetto serializzato JSON

```
context.Logger.LogInformation("User {@user} logged in", User);
```

L'output del messaggio di log di questo comando è mostrato nell'esempio JSON seguente.

Example Record di 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"
    }
}
```

Se l'argomento per un parametro aggiuntivo è un array o implementa `IList` o`IDictionary`, Lambda aggiunge l'argomento al messaggio di log JSON come matrice, come mostrato nell'esempio seguente di record di log JSON. In questo esempio, `{users}` accetta un argomento `IList` contenente istanze della proprietà `User` con lo stesso formato dell'esempio precedente. Lambda converte `IList` in un array, con ogni valore creato utilizzando il metodo `ToString`.

Example Record di log JSON con un argomento **`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"
    ]
}
```

Puoi serializzare l'elenco in JSON anche utilizzando il prefisso `@` nel comando di registrazione. Nel seguente esempio di record di log JSON, la proprietà `users` è serializzata in JSON.

Example Record di log JSON con un argomento **`IList`** serializzato in 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"
        }
    ]
}
```

Utilizzo del filtraggio a livello di log con .NET

Configurando il filtraggio a livello di registro, è possibile scegliere di inviare a Logs solo i log con un determinato livello di dettaglio o inferiore. CloudWatch Per informazioni su come configurare il filtraggio a livello di log della funzione, consulta la pagina the section called “Filtraggio a livello di log”.

Per AWS Lambda filtrare i messaggi di registro in base al livello di registro, è possibile utilizzare registri in formato JSON o utilizzare i metodi.NET per generare messaggi di registro. `Console` Per creare log in formato JSON, configura il tipo di log della funzione su JSON e usa l'istanza `ILambdaLogger`.

Con i log in formato JSON, Lambda filtra gli output dei log utilizzando la coppia chiave-valore "livello" nell'oggetto JSON descritto in the section called “Utilizzo del formato di log JSON strutturato con .NET”.

Se utilizzi i `Console` metodi.NET per scrivere messaggi CloudWatch nei registri, Lambda applica i livelli di registro ai tuoi messaggi come segue:

- Console. WriteLine metodo - Lambda applica un livello di registro di `INFO`
- Metodo Console.Error: Lambda applica il livello di log `ERROR`

Quando configuri la funzione per utilizzare il filtraggio a livello di log, devi selezionare una delle seguenti opzioni per il livello di log che Lambda invii a Logs. CloudWatch Nota la mappatura dei livelli di log utilizzati da Lambda con i livelli Microsoft standard utilizzati da .NET `ILambdaLogger`.

Livello di log Lambda	Livello Microsoft equivalente	Utilizzo standard
TRACE (dettaglio massimo)	Traccia	Le informazioni più dettaglia te utilizzate per tracciare il percorso di esecuzione del codice
DEBUG	Esegui il debug	Informazioni dettagliate per il debug del sistema
INFO	Informazioni	Messaggi che registrano il normale funzionamento della funzione
WARN	Attenzione	Messaggi relativi a potenzial i errori che possono portare a comportamenti imprevisti se non risolti
ERRORE	Errore	Messaggi relativi a problemi che impediscono al codice di funzionare come previsto
FATAL (dettaglio minimo)	Critica	Messaggi relativi a errori gravi che causano l'interruzione del funzionamento dell'appl icazione

Lambda invia i log del livello di dettaglio selezionato e inferiore a. CloudWatch Ad esempio, se configuri un livello di log WARN, Lambda invierà i log corrispondenti ai livelli WARN, ERROR e FATAL.

Strumenti e librerie di registrazione aggiuntivi

Powertools for AWS Lambda (.NET) è un toolkit per sviluppatori che implementa le migliori pratiche Serverless e aumenta la velocità degli sviluppatori. L'utilità di registrazione fornisce un logger ottimizzato per Lambda che include informazioni aggiuntive sul contesto delle funzioni in tutte le funzioni con output strutturato come JSON. Utilizza l'utility per eseguire le seguenti operazioni:

- Acquisizione di campi essenziali dal contesto Lambda, avvii a freddo e output di registrazione della struttura come JSON
- Registrazione degli eventi di chiamata Lambda quando richiesto (disabilitata per impostazione predefinita)
- Stampa di tutti i log solo per una percentuale di chiamate tramite campionamento dei log (disabilitata per impostazione predefinita)
- Aggiunta di chiavi supplementari al log strutturato in qualsiasi momento
- Utilizzo di un formattatore di log personalizzato (Bring Your Own Formatter) per generare i log in una struttura compatibile con Logging RFC dell'organizzazione

Utilizzo di Powertools per AWS Lambda (.NET) e per la registrazione strutturata AWS SAM

Segui i passaggi seguenti per scaricare, creare e distribuire un'applicazione Hello World C# di esempio con moduli Powertools for AWS Lambda (.NET) integrati utilizzando il. AWS SAM Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando invii una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e metriche utilizzando Embedded Metric Format e invia tracce a. CloudWatch AWS X-Ray La funzione restituisce un messaggio `hello world`.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- .NET 8
- AWS CLI versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della AWS SAM CLI, consulta Aggiornamento della CLI. AWS SAM

Implementa un'applicazione di esempio AWS SAM

1. Inizializza l'applicazione utilizzando il modello Hello World. TypeScript

```
sam init --app-template hello-world-powertools-dotnet --name sam-app --package-type
 Zip --runtime dotnet6 --no-tracing
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi `Enter`.

> **Note**
>
> Perché HelloWorldFunction potrebbe non avere un'autorizzazione definita. Va bene? , assicurati di entrarey.

5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Richiama l'endpoint dell'API:

```
curl -X GET <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere i log per la funzione, esegui sam logs. Per ulteriori informazioni, consulta l'argomento relativo all'utilizzo dei log nella Guida per sviluppatori AWS Serverless Application Model .

```
sam logs --stack-name sam-app
```

L'output del log ha la struttura seguente:

```
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. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

Gestione della conservazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, elimina il gruppo di log o configura un periodo di conservazione dopo il quale i log CloudWatch vengono eliminati automaticamente. Per configurare la conservazione dei log, aggiungi quanto segue al tuo modello: AWS SAM

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function
    Properties:
      # Omitting other properties

  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Sub "/aws/lambda/${HelloWorldFunction}"
      RetentionInDays: 7
```

Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor del codice incorporato, troverai i log nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l'output del log nella sezione Dettagli.

Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Apri la pagina Registra gruppi sulla CloudWatch console.

2. Scegli il gruppo di log per la tua funzione (***your-function-name***/aws/lambda/).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Viene visualizzato un flusso di log quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire le chiamate simultanee. Per trovare i log per una chiamata specifica, consigliamo di strumentare la funzione con. AWS X-Ray X-Ray registra i dettagli sulla richiesta e il flusso di log nella traccia.

Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della AWS CLI versione 2.

È possibile utilizzare AWS CLI per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'`base64` utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `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
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
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
```

L'utilità `base64` è disponibile su Linux, macOS e Ubuntu su Windows. Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

Example Script get-logs.sh

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza `sed` per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.

L'cli-binary-formatopzione è obbligatoria se utilizzi la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 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 (solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il seguente comando per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
    "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"
}
```


Eliminazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, eliminare il gruppo di log o configurare un periodo di conservazione trascorso il quale i log vengono eliminati automaticamente.

Strumentazione del codice C# in AWS Lambda

Lambda si integra con AWS X-Ray per aiutarti a tracciare, eseguire il debug e ottimizzare le applicazioni Lambda. Puoi utilizzare X-Ray per tracciare una richiesta mentre attraversa le risorse nell'applicazione, che possono includere funzioni Lambda e altri servizi AWS.

Per inviare dati di tracciamento a X-Ray, è possibile utilizzare una delle tre librerie SDK:

- AWS Distro for OpenTelemetry (ADOT): una distribuzione supportata da AWS sicura, pronta per la produzione, dell'SDK OpenTelemetry (OTel).
- SDK AWS X-Ray per .NET – un SDK per generare e inviare i dati di traccia su X-Ray.
- Powertools per AWS Lambda (.NET): un toolkit rivolto agli sviluppatori per implementare le best practice serverless e aumentare la velocità di sviluppo.

Ciascun SDK offre dei modi per inviare i dati di telemetria al servizio X-Ray. Puoi quindi utilizzare X-Ray per visualizzare, filtrare e analizzare le metriche delle prestazioni dell'applicazione per identificare i problemi e le opportunità di ottimizzazione.

> **Important**
>
> Gli SDK X-Ray e Powertools per AWS Lambda fanno parte di una soluzione di strumentazione strettamente integrata offerta da AWS. I livelli Lambda ADOT fanno parte di uno standard di settore per la strumentazione di tracciamento che in generale raccoglie più dati, ma potrebbero non essere adatti a tutti i casi d'uso. È possibile implementare il tracciamento end-to-end di X-Ray utilizzando entrambe le soluzioni. Per ulteriori informazioni sulla scelta più adatta, consulta Scegliere tra le AWS Distro per OpenTelemetry e SDK X-Ray.

Sezioni

- Utilizzo di Powertools per AWS Lambda (.NET) e di AWS SAM per il tracciamento
- Utilizzo dell'SDK X-Ray per strumentare le funzioni .NET
- Attivazione del tracciamento con la console Lambda
- Attivazione del tracciamento con l'API Lambda
- Attivazione del tracciamento con CloudFormation
- Interpretazione di una traccia X-Ray

Utilizzo di Powertools per AWS Lambda (.NET) e di AWS SAM per il tracciamento

Segui i passaggi riportati sotto per scaricare, creare e implementare un'applicazione Hello World in C# di esempio con i moduli Powertools per AWS Lambda (.NET) integrati utilizzando AWS SAM. Questa applicazione implementa un backend dell'API di base e utilizza Powertools per l'emissione di log, parametri e tracce. Consiste in un endpoint Gateway Amazon API e in una funzione Lambda. Quando si invia una richiesta GET all'endpoint API Gateway, la funzione Lambda richiama, invia log e parametri a CloudWatch utilizzando Embedded Metric Format e invia le tracce a AWS X-Ray. La funzione restituisce un messaggio hello world.

Prerequisiti

Per completare le fasi riportate in questa sezione, è necessario:

- .NET 8
- AWS CLI versione 2
- AWS SAM CLI versione 1.75 o successiva. Se disponi di una versione precedente della CLI AWS SAM, consulta Aggiornamento della CLI AWS SAM.

Implementare un'applicazione AWS SAM di esempio

1. Inizializzare l'applicazione utilizzando il modello Hello World TypeScript.

```
sam init --app-template hello-world-powertools-dotnet --name sam-app --package-type
 Zip --runtime dotnet6 --no-tracing
```

2. Costruisci l'app.

```
cd sam-app && sam build
```

3. Distribuire l'app.

```
sam deploy --guided
```

4. Seguire le istruzioni visualizzate sullo schermo. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi `Enter`.

> **Note**
>
> Per HelloWorldFunction potrebbe non avere l'autorizzazione definita, va bene?, assicurati di inserire y.

5. Ottieni l'URL dell'applicazione implementata:

```
aws cloudformation describe-stacks --stack-name sam-app --query
 'Stacks[0].Outputs[?OutputKey==`HelloWorldApi`].OutputValue' --output text
```

6. Richiama l'endpoint dell'API:

```
curl <URL_FROM_PREVIOUS_STEP>
```

In caso di esito positivo, vedrai questa risposta:

```
{"message":"hello world"}
```

7. Per ottenere le tracce per la funzione, esegui sam traces.

```
sam traces
```

L'output della traccia ha il seguente aspetto:

```
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. Questo è un endpoint API pubblico accessibile su Internet. È consigliabile eliminare l'endpoint dopo il test.

```
sam delete
```

X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non è possibile configurare la frequenza di campionamento di X-Ray per le funzioni.

Utilizzo dell'SDK X-Ray per strumentare le funzioni .NET

È possibile utilizzare il codice funzione per registrare i metadati e tracciare le chiamate a valle. Per registrare i dettagli delle chiamate effettuate dalla funzione ad altre risorse e servizi, utilizza SDK AWS X-Ray per .NET. Per ottenere l'SDK, aggiungere i pacchetti `AWSXRayRecorder` al file di progetto.

```
<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>
```

Esistono diversi pacchetti Nuget che forniscono strumentazione automatica per gli SDK AWS, Entity Framework e richieste HTTP. Per visualizzare il set completo di opzioni di configurazione, consulta SDK AWS X-Ray per .NET nella Guida per gli sviluppatori di AWS X-Ray.

Dopo aver aggiunto i pacchetti Nuget desiderati, configura la strumentazione automatica. Una best practice consiste nell'eseguire questa configurazione al di fuori della funzione di gestione della funzione. Ciò consente di sfruttare il riutilizzo dell'ambiente di esecuzione per migliorare le prestazioni della funzione. Nel seguente esempio di codice, il metodo `RegisterXRayForAllServices` viene chiamato nel costruttore della funzione per aggiungere la strumentazione per tutte le chiamate dell'SDK AWS.

```
[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)
        };
    }
}
```

Attivazione del tracciamento con la console Lambda

Per attivare il tracciamento attivo sulla funzione Lambda con la console, attenersi alla seguente procedura:

Per attivare il tracciamento attivo

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione) e quindi Monitoring and operations tools (Strumenti di monitoraggio e operazioni).
4. In Strumenti di monitoraggio aggiuntivi scegli Modifica.
5. In CloudWatch Application Signals and AWS X-Ray, scegli Abilita per Tracce di servizio Lambda.
6. Selezionare Salva.

Attivazione del tracciamento con l'API Lambda

Configurare il tracciamento sulla funzione Lambda con la AWS CLI o SDK AWS e utilizzare le seguenti operazioni API:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Il seguente comando AWS CLI di esempio consente il tracciamento attivo su una funzione denominata my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

La modalità di tracciamento fa parte della configurazione specifica della versione quando si pubblica una versione della funzione. Non è possibile modificare la modalità di tracciamento in una versione pubblicata.

Attivazione del tracciamento con CloudFormation

Per attivare il tracciamento su una risorsa `AWS::Lambda::Function` in un modello CloudFormation, utilizzare la proprietà `TracingConfig`.

Example function-inline.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Per una risorsa AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, utilizzare la proprietà `Tracing`.

Example template.yml – Configurazione del tracciamento

```
Resources:
```

```
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Interpretazione di una traccia X-Ray

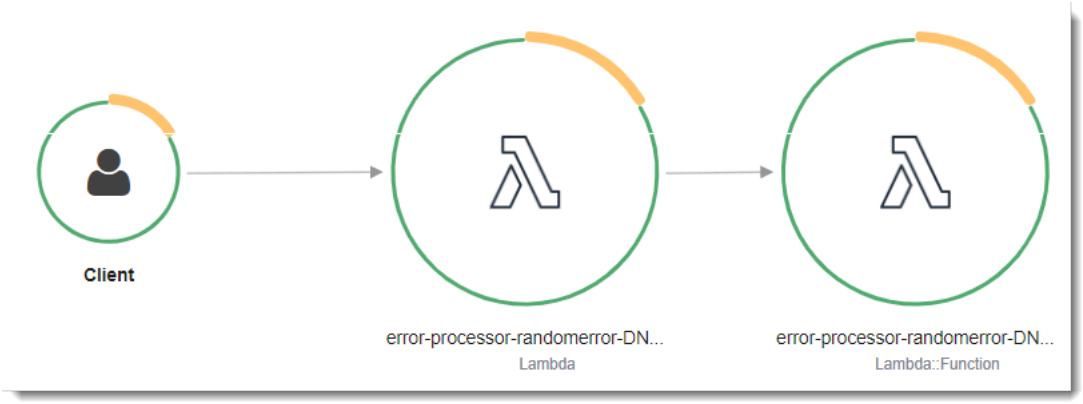
La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al ruolo di esecuzione della funzione. In caso contrario, aggiungere la policy AWSXRayDaemonWriteAccess al ruolo di esecuzione.

Dopo aver configurato il tracciamento attivo, è possibile osservare richieste specifiche tramite l'applicazione. Il grafico dei servizi X-Ray mostra informazioni sull'applicazione e tutti i suoi componenti. Il seguente esempio mostra un'applicazione con due funzioni. La funzione principale elabora gli eventi e talvolta restituisce errori. La seconda funzione elabora gli errori che appaiono nel gruppo di log della prima funzione e utilizza l'SDK AWS per chiamare X-Ray, Amazon Simple Storage Service (Amazon S3) e Amazon CloudWatch Logs.

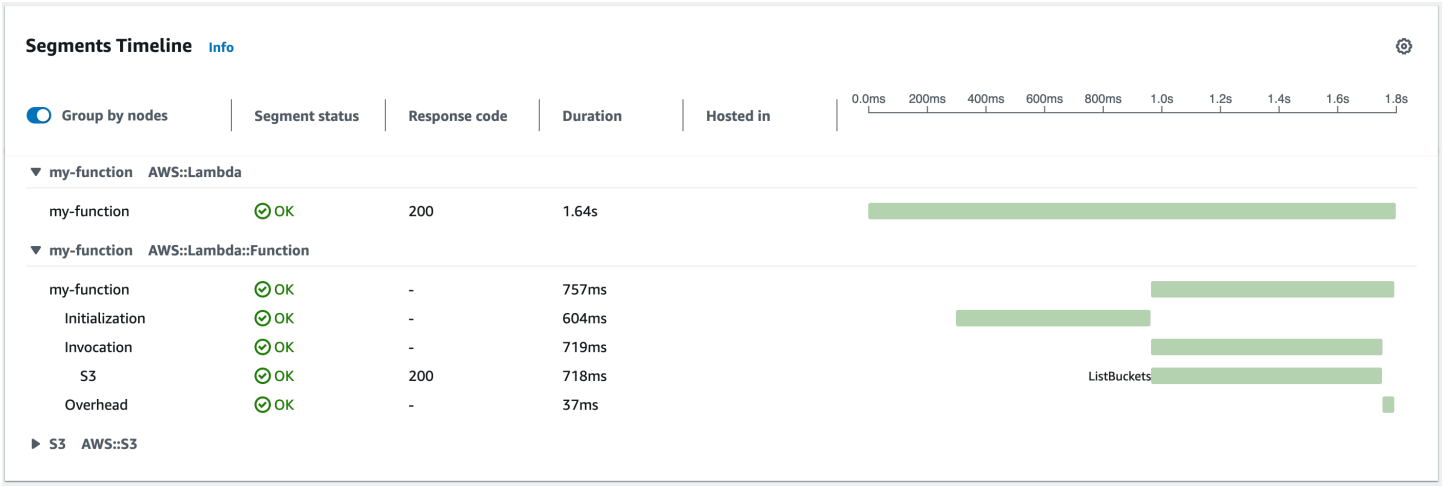


X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non è possibile configurare la frequenza di campionamento di X-Ray per le funzioni.

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda. L'esempio seguente mostra una traccia con questi 2 segmenti. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.



Questo esempio espande il segmento `AWS::Lambda::Function` per visualizzare i relativi tre sottosegmenti.

> **Note**
>
> AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. La traccia di esempio mostrata qui illustra il segmento di funzione vecchio stile. Le differenze tra i segmenti vecchio e nuovo stile sono descritte nei paragrafi seguenti.
> Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni in tutte le Regioni AWS, ad eccezione delle regioni Cina e GovCloud, passeranno all'utilizzo dei messaggi di log e dei segmenti di traccia di nuovo formato.

Il segmento di funzioni vecchio stile contiene i seguenti sottosegmenti:

- Inizializzazione – Rappresenta il tempo trascorso a caricare la funzione e ad eseguire il codice di inizializzazione. Questo sottosegmento viene visualizzato solo per il primo evento che viene elaborato da ogni istanza della funzione.
- Chiamata: rappresenta il tempo impiegato per eseguire il codice del gestore.
- Overhead: rappresenta il tempo impiegato dal runtime Lambda per prepararsi a gestire l'evento successivo.

Il segmento di funzione di nuovo stile non contiene un sottosegmento `Invocation`. I sottosegmenti dei clienti sono invece collegati direttamente al segmento di funzioni. Per ulteriori informazioni sulla struttura dei segmenti di funzioni vecchio e nuovo stile, consulta the section called “Informazioni sui monitoraggi di X-Ray”.

È inoltre possibile strumentare i client HTTP, registrare query SQL e creare segmenti secondari personalizzati con annotazioni e metadati. Per ulteriori informazioni, consulta SDK AWS X-Ray per .NET nella Guida per gli sviluppatori di AWS X-Ray.

> **Prezzi**
>
> È possibile utilizzare il tracciamento X-Ray gratuitamente ogni mese fino a un certo limite come parte del piano gratuito di AWS. Oltre la soglia, X-Ray addebita lo storage di traccia e il recupero. Per ulteriori informazioni, consulta Prezzi di AWS X-Ray.

Test della funzione AWS Lambda in C#

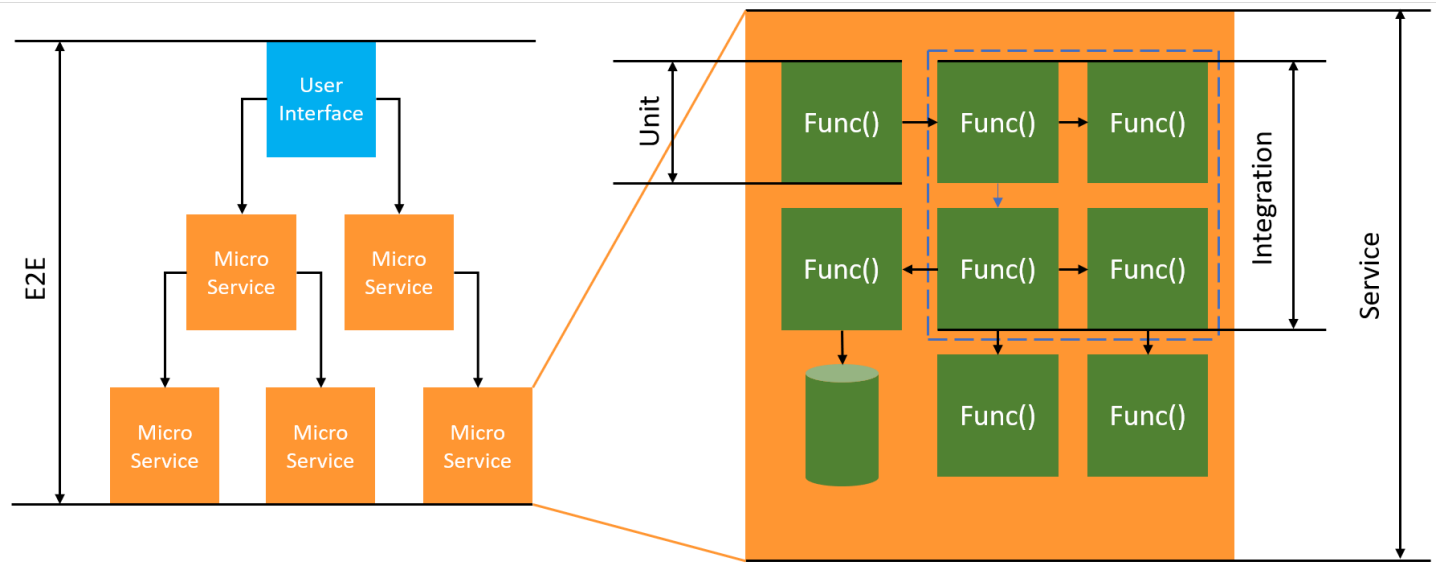
> **Note**
>
> Consulta il capitolo sul Test delle funzioni per un'introduzione completa alle tecniche e alle best practice per testare soluzioni serverless.

Sebbene il test delle funzioni serverless si basi su tipologie e tecniche di test consolidate, è importante tenere in considerazione la possibilità di effettuare test sulle applicazioni serverless nella loro interezza. I test basati sul cloud forniranno la misura più accurata della qualità sia delle funzioni sia delle applicazioni serverless.

Un'architettura applicativa serverless include servizi gestiti che forniscono funzionalità applicative critiche tramite chiamate API. Pertanto, è fondamentale che il ciclo di sviluppo preveda test automatici in grado di verificare il corretto funzionamento dell'interazione tra le funzioni e i servizi.

Se non crei test basati sul cloud, potresti riscontrare problemi dovuti alle differenze tra l'ambiente locale e quello implementato. Il processo di integrazione continua dovrebbe eseguire test su un insieme di risorse con provisioning nel cloud prima di promuovere il codice nell'ambiente di implementazione successivo, come quello di controllo qualità (QA), staging o produzione.

Continua a leggere questa breve guida per scoprire le strategie di test per le applicazioni serverless oppure visita il repository Serverless Test Samples per approfondire esempi pratici, specifici per il linguaggio e il runtime selezionati.



Per i test in ambito serverless, è necessario scrivere test unitari, test di integrazione e test end-to-end.

- Test unitari: vengono eseguiti su un blocco di codice isolato. Ad esempio, possono essere utilizzati per verificare la logica aziendale per calcolare le spese di spedizione in base a un articolo e a una destinazione specifici.
- Test di integrazione: coinvolgono due o più componenti o servizi che interagiscono, in genere in un ambiente cloud. Ad esempio, possono essere utilizzati per verificare che una funzione elabori gli eventi di una coda.
- Test end-to-end: verificano il comportamento di un'intera applicazione. Ad esempio, possono essere utilizzati per verificare che l'infrastruttura sia configurata correttamente e che gli eventi fluiscano tra i servizi come previsto per registrare l'ordine di un cliente.

Test delle applicazioni serverless

Generalmente, utilizzerai una combinazione di approcci per testare il codice dell'applicazione serverless, inclusi test nel cloud, test con mock e occasionalmente test con emulatori.

Test nel cloud

I test nel cloud sono utili per tutte le fasi di test, inclusi i test unitari, i test di integrazione e i test end-to-end. Esegui test su codice implementato nel cloud e interagisci con servizi basati su cloud. Questo approccio fornisce la misura più accurata della qualità del codice.

Un modo conveniente per eseguire il debug di una funzione Lambda nel cloud è con un evento di test nella console. Un evento di test è un input JSON per la funzione. Se la funzione non richiede input, l'evento può essere un documento JSON (`{}`) vuoto. La console fornisce eventi di esempio per una varietà di integrazioni di servizi. Dopo aver creato un evento nella console, puoi condividerlo con il tuo team per rendere i test più semplici e coerenti.

> **Note**
>
> Testare una funzione nella console è un modo rapido per iniziare, ma l'automazione dei cicli di test garantisce la qualità delle applicazioni e la velocità di sviluppo.

Strumenti di test

Per accelerare il ciclo di sviluppo, esistono diversi strumenti e tecniche che è possibile utilizzare durante il test delle funzioni. Ad esempio, AWS SAM Accelerate e AWS CDK CDK watch mode riducono entrambi il tempo necessario per aggiornare gli ambienti cloud.

Il modo in cui definisci il codice della funzione Lambda semplifica l'aggiunta di test unitari. Per inizializzare la classe, Lambda richiede un costruttore pubblico senza parametri. L'introduzione di un secondo costruttore interno consente di controllare le dipendenze utilizzate dall'applicazione.

```
[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)
        };
    }
}
```

Per scrivere un test per questa funzione, puoi inizializzare una nuova istanza della classe `Function` e passare un'implementazione simulata di `IDatabaseRepository`. Gli esempi seguenti utilizzano `XUnit`, `Moq` e `FluentAssertions` per scrivere un semplice test che assicuri che `FunctionHandler` restituisca un codice di stato 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);
    }
}
```

Per esempi più dettagliati, inclusi esempi di test asincroni, consulta il repository di esempi di test .NET su GitHub.

Creazione di funzioni Lambda con PowerShell

Le sezioni seguenti spiegano come si applicano i modelli di programmazione e i concetti fondamentali comuni quando si crea codice di funzione Lambda. PowerShell

Lambda fornisce le seguenti applicazioni di esempio per: PowerShell

- blank-powershell — Una PowerShell funzione che mostra l'uso della registrazione, delle variabili di ambiente e dell'SDK. AWS

Prima di iniziare, devi prima configurare un ambiente di sviluppo. PowerShell Per istruzioni su come eseguire questa operazione, consultare Configurazione di un ambiente di sviluppo PowerShell.

Per ulteriori informazioni su come utilizzare il AWSLambda PSCore modulo per scaricare PowerShell progetti di esempio dai modelli, creare pacchetti di PowerShell distribuzione e distribuire PowerShell funzioni nel AWS cloud, consultaDistribuzione delle funzioni Lambda di PowerShell con gli archivi di file .zip.

Lambda fornisce i runtime seguenti per i linguaggi .NET:

Name	Identificatore	Sistema operativo	Data di ritiro	Blocco creazione funzioni	Blocco aggiornamento funzioni
.NET 10	`dotnet10`	Amazon Linux 2023	14 novembre 2028	14 dicembre 2028	15 gennaio 2029
.NET 9 (solo contenitore)	`dotnet9`	Amazon Linux 2023	10 novembre 2026	Non programmato	Non programmato
.NET 8	`dotnet8`	Amazon Linux 2023	10 novembre 2026	10 dicembre 2026	11 gennaio 2027

Argomenti

- Configurazione di un ambiente di sviluppo PowerShell
- Distribuzione delle funzioni Lambda di PowerShell con gli archivi di file .zip

- Definire l'handler della funzione Lambda in PowerShell
- Utilizzo dell'oggetto contestuale Lambda per recuperare le informazioni sulla funzione PowerShell
- Registrare e monitorare le funzioni Lambda con Powershell

Configurazione di un ambiente di sviluppo PowerShell

Lambda fornisce un insieme di strumenti e librerie per il runtime PowerShell. Per le istruzioni di installazione, consulta Lambda Tools for Powershell su GitHub.

Il modulo AWSLambdaPSCore include i seguenti nuovi cmdlet per aiutare a creare e pubblicare funzioni PowerShell Lambda.

- Get-AWSPowerShellLambdaTemplate – Restituisce un elenco di modelli di base.
- New-AWSPowerShellLambda – Crea uno script PowerShell iniziale basato su un modello.
- Publish-AWSPowerShellLambda – Pubblica un dato script PowerShell su Lambda.
- New-AWSPowerShellLambdaPackage – Crea un pacchetto di implementazione Lambda che può essere utilizzato in un sistema CI/CD per l'implementazione.

Distribuzione delle funzioni Lambda di PowerShell con gli archivi di file .zip

Un pacchetto di distribuzione per il runtime PowerShell contiene lo script PowerShell, i moduli PowerShell necessari per lo script PowerShell e gli assembly necessari per ospitare PowerShell Core.

Creazione di una funzione Lambda

Per iniziare a scrivere e invocare uno script PowerShell con , è possibile utilizzare il cmdlet `New-AWSPowerShellLambda` per creare un script di inizio in base a un modello. È possibile utilizzare il cmdlet `Publish-AWSPowerShellLambda` per distribuire lo script in Lambda. Quindi è possibile testare lo script utilizzando la riga di comando o la console Lambda.

Per creare un nuovo script PowerShell, caricarlo e testarlo, effettuare le seguenti operazioni:

1. Per visualizzare l'elenco dei modelli disponibili, esegui il comando seguente:

   ```
   PS C:\> Get-AWSPowerShellLambdaTemplate

   Template               Description
   --------               -----------
   Basic                  Bare bones script
   CodeCommitTrigger      Script to process AWS CodeCommit Triggers
   ...
   ```

2. Per creare uno script di esempio in base al modello `Basic`, esegui il comando seguente:

   ```
   New-AWSPowerShellLambda -ScriptName MyFirstPSScript -Template Basic
   ```

   Un nuovo file con nome `MyFirstPSScript.ps1` viene creato in una nuova sottodirectory della directory corrente. Il nome della directory viene determinato in base al parametro `-ScriptName`. Puoi utilizzare il parametro `-Directory` per scegliere una directory alternativa.

   Puoi vedere che il nuovo file ha il seguente contenuto:

   ```
   # 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. Per vedere come i messaggi di log dello script PowerShell vengono inviati ad Amazon CloudWatch Logs, rimuovi il commento dalla riga `Write-Host` dello script di esempio.

   Per dimostrare come puoi restituire i dati dalle tue funzioni Lambda, aggiungi una nuova riga alla fine dello script con `$PSVersionTable`. In tal modo `$PSVersionTable` viene aggiunto alla pipeline PowerShell. Al completamento dello script di PowerShell, l'ultimo oggetto nella pipeline PowerShell sono i dati restituiti per la funzione Lambda. `$PSVersionTable` è una variabile globale PowerShell che fornisce inoltre informazioni sull'ambiente di esecuzione.

   Dopo aver apportato le modifiche, le ultime due righe dello script di esempio sono come riportato di seguito:

   ```
   Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)
   $PSVersionTable
   ```

4. Dopo la modifica del file `MyFirstPSScript.ps1`, modifica la directory sul percorso dello script. Quindi esegui il seguente comando per pubblicare lo script in Lambda:

   ```
   Publish-AWSPowerShellLambda -ScriptPath .\MyFirstPSScript.ps1 -Name
    MyFirstPSScript -Region us-east-2
   ```

   Nota che il parametro `-Name` specifica il nome della funzione Lambda che viene visualizzato nella console Lambda. Puoi utilizzare questa funzione per invocare manualmente lo script.

5. Invoca la funzione utilizzando il comando AWS Command Line Interface (AWS CLI) `invoke`.

```
> aws lambda invoke --function-name MyFirstPSScript out
```

Definire l'handler della funzione Lambda in PowerShell

Quando una funzione Lambda viene invocata, il gestore Lambda richiama lo script PowerShell.

Quando lo script PowerShell viene richiamato, le seguenti variabili sono predefinite:

- *$LambdaInput* – PSObject contenente l'input per il gestore. L'input può essere costituito da dati dell'evento, pubblicati da un'origine eventi, o un input personalizzato che fornisci, ad esempio una stringa o qualsiasi oggetto dati personalizzato.
- *$LambdaContext* – Un oggetto Amazon.Lambda.Core.ILambdaContext che è possibile utilizzare per accedere alle informazioni sulla chiamata corrente, come il nome della funzione corrente, il limite di memoria, il runtime rimanente e la registrazione.

A titolo illustrativo, considera l'esempio di codice PowerShell riportato di seguito.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
Write-Host 'Function Name:' $LambdaContext.FunctionName
```

Questo script restituisce la proprietà FunctionName ottenuta dalla variabile $LambdaContext.

> **Note**
>
> Sei tenuto a usare l'istruzione `#Requires` all'interno dei tuoi script PowerShell per indicare i moduli da cui dipendono gli script. Questa istruzione svolge due attività importanti. 1) Comunica ad altri sviluppatori quali moduli lo script utilizza e 2) identifica i moduli dipendenti richiesti dagli strumenti AWS PowerShell per creare un pacchetto con lo script, come parte della distribuzione. Per ulteriori informazioni sull'istruzione `#Requires` in PowerShell, consulta la pagina relativa alle informazioni sui requisiti. Per ulteriori dettagli sui pacchetti di distribuzione PowerShell, consulta Distribuzione delle funzioni Lambda di PowerShell con gli archivi di file .zip.
>
> Quando la funzione Lambda PowerShell usa i cmdlet PowerShell AWS, assicurati di impostare un'istruzione `#Requires` che fa riferimento al modulo `AWSPowerShell.NetCore` che supporta PowerShell Core e non al modulo `AWSPowerShell` che supporta solo Windows PowerShell. Inoltre, assicurati di utilizzare la versione 3.3.270.0 o più recente di `AWSPowerShell.NetCore` che ottimizza il processo di importazione dei cmdlet. Se usi una versione precedente, sperimenterai partenze a freddo più lunghe. Per ulteriori informazioni, consulta Tools for PowerShell AWS.

Restituzione dei dati

Alcune chiamate Lambda hanno lo scopo di restituire i dati al loro chiamante. Ad esempio, se una chiamata era in risposta a una richiesta web proveniente da API Gateway, la funzione Lambda deve restituire la risposta. Per PowerShell Lambda, l'ultimo oggetto aggiunto alla pipeline PowerShell costituisce i dati restituiti dalla chiamata Lambda. Se l'oggetto è una stringa, i dati vengono restituiti così come sono. In caso contrario, l'oggetto viene convertito in formato JSON utilizzando il cmdlet `ConvertTo-Json`.

Ad esempio, considera la seguente istruzione PowerShell che aggiunge `$PSVersionTable` alla pipeline PowerShell:

```
$PSVersionTable
```

Al termine dello script di PowerShell, l'ultimo oggetto nella pipeline PowerShell sono i dati restituiti per la funzione Lambda. `$PSVersionTable` è una variabile globale PowerShell che fornisce inoltre informazioni sull'ambiente di esecuzione.

Utilizzo dell'oggetto contestuale Lambda per recuperare le informazioni sulla funzione PowerShell

Quando Lambda esegue la tua funzione, passa le informazioni di contesto rendendo una variabile `$LambdaContext` disponibile al gestore. Questa variabile fornisce i metodi e le proprietà con informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

Proprietà del contesto

- `FunctionName`: il nome della funzione Lambda.
- `FunctionVersion`: la versione della funzione.
- `InvokedFunctionArn`: l'Amazon Resource Name (ARN) utilizzato per richiamare la funzione. Indica se l'invoker ha specificato un numero di versione o un alias.
- `MemoryLimitInMB`: la quantità di memoria allocata per la funzione.
- `AwsRequestId`: l'identificatore della richiesta di invocazione.
- `LogGroupName`: il gruppo di log per la funzione.
- `LogStreamName`: il flusso di log per l'istanza della funzione.
- `RemainingTime`: il numero di millisecondi rimasti prima del timeout dell'esecuzione.
- `Identity`: (app per dispositivi mobili) Informazioni relative all'identità Amazon Cognito che ha autorizzato la richiesta.
- `ClientContext`: (app per dispositivi mobili) Contesto client fornito a Lambda dall'applicazione client.
- `Logger`: l'oggetto logger per la funzione.

Lo snippet di codice PowerShell mostra una funzione semplice dell'handler che stampa alcune informazioni relative al contesto.

```
#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
```

Registrare e monitorare le funzioni Lambda con Powershell

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della funzione. L'ambiente del runtime Lambda invia i dettagli su ogni richiamo al flusso di log e inoltra i log e l'output del codice della funzione. Per ulteriori informazioni, consulta Lambda invia automaticamente i log delle funzioni a CloudWatch Logs..

Questa pagina descrive come produrre un output di registro dal codice della funzione Lambda e accedere ai log utilizzando AWS Command Line Interface la console Lambda o la console. CloudWatch

Sections

- Creazione di una funzione che restituisce i registri
- Visualizzazione dei log nella console Lambda
- Visualizzazione dei log nella console CloudWatch
- Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI
- Eliminazione dei log

Creazione di una funzione che restituisce i registri

Per generare log dal codice della funzione, è possibile utilizzare i cmdlet in Microsoft. PowerShell.Utility o qualsiasi modulo di registrazione che scrive su o. `stdout` `stderr` Nell'esempio seguente viene utilizzato `Write-Host`.

Example function/Handler.ps1 – Registrazione

```
#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 Formato dei 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
```

Il runtime di .NET registra `START`, `END` e `REPORT` per ogni chiamata. La riga del report fornisce i seguenti dettagli.

Campi dati della riga REPORT

- RequestId— L'ID univoco della richiesta per la chiamata.

- Durata – La quantità di tempo che il metodo del gestore della funzione impiega durante l'elaborazione dell'evento.
- Durata fatturata – La quantità di tempo fatturata per la chiamata.
- Dimensioni memoria – La quantità di memoria allocata per la funzione.
- Quantità max utilizzata – La quantità di memoria utilizzata dalla funzione. Quando le invocazioni condividono un ambiente di esecuzione, Lambda riporta la memoria massima utilizzata in tutte le invocazioni. Questo comportamento potrebbe comportare un valore riportato superiore al previsto.
- Durata Init – Per la prima richiesta servita, la quantità di tempo impiegato dal runtime per caricare la funzione ed eseguire il codice al di fuori del metodo del gestore.
- XRAY TraceId — Per le richieste tracciate, l'ID di traccia.AWS X-Ray
- SegmentId— Per le richieste tracciate, l'ID del segmento X-Ray.
- Campionato – Per le richieste tracciate, il risultato del campionamento.

Visualizzazione dei log nella console Lambda

È possibile utilizzare la console Lambda per visualizzare l'output del log dopo aver richiamato una funzione Lambda.

Se il codice può essere testato dall'editor del codice incorporato, troverai i log nei risultati dell'esecuzione. Quando utilizzi la funzionalità di test della console per richiamare una funzione, troverai l'output del log nella sezione Dettagli.

Visualizzazione dei log nella console CloudWatch

Puoi utilizzare la CloudWatch console Amazon per visualizzare i log di tutte le chiamate di funzioni Lambda.

Per visualizzare i log sulla console CloudWatch

1. Apri la pagina Registra gruppi sulla CloudWatch console.
2. Scegli il gruppo di log per la tua funzione (***your-function-name***/aws/lambda/).
3. Creare un flusso di log.

Ogni flusso di log corrisponde a un'istanza della funzione. Viene visualizzato un flusso di log quando aggiorni la funzione Lambda e quando vengono create istanze aggiuntive per gestire le chiamate

simultanee. Per trovare i log per una chiamata specifica, consigliamo di strumentare la funzione con. AWS X-Ray X-Ray registra i dettagli sulla richiesta e il flusso di log nella traccia.

Visualizzazione dei log utilizzando () AWS Command Line InterfaceAWS CLI

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della AWS CLI versione 2.

È possibile utilizzare AWS CLI per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'`base64` utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `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
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-`

`base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
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
```

L'utilità `base64` è disponibile su Linux, macOS e Ubuntu su Windows. Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

Example Script get-logs.sh

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza `sed` per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.

L'cli-binary-formatopzione è obbligatoria se utilizzi la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 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 (solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il seguente comando per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
    "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"
}
```


Eliminazione dei log

I gruppi di log non vengono eliminati automaticamente quando si elimina una funzione. Per evitare di archiviare i log a tempo indeterminato, eliminare il gruppo di log o configurare un periodo di conservazione trascorso il quale i log vengono eliminati automaticamente.

Compilazione di funzioni Lambda con Rust

Poiché Rust viene compilato in codice nativo, non è necessario un runtime dedicato per eseguire il codice Rust su Lambda. Utilizza invece il client di runtime Rust per creare il tuo progetto localmente, quindi distribuiscilo su Lambda utilizzando un runtime solo per il sistema operativo. Quando utilizzi un runtime solo per il sistema operativo, Lambda mantiene automaticamente il sistema operativo aggiornato con le patch più recenti.

Strumenti e librerie per Rust

- AWS SDK per Rust: L' AWS SDK per Rust consente APIs a Rust di interagire con i servizi di infrastruttura Amazon Web Services.
- Client di runtime Rust per Lambda: Il client di runtime Rust semplifica l'esecuzione di funzioni Lambda scritte in Rust.
- Cargo Lambda: si tratta di un'estensione open source di terze parti dello strumento da riga di comando Cargo che semplifica la creazione e l'implementazione di funzioni Rust Lambda.
- HTTP Lambda: questa libreria fornisce un wrapper per lavorare con gli eventi HTTP.
- Estensione Lambda: questa libreria fornisce supporto per scrivere estensioni Lambda con Rust.
- AWS Lambda Eventi: questa libreria fornisce definizioni dei tipi per le integrazioni di sorgenti di eventi comuni.

Esempi di applicazioni Lambda per Rust

- Funzione Lambda di base: una funzione Rust che mostra come elaborare gli eventi di base.
- Funzione Lambda con gestione degli errori: una funzione Rust che mostra come gestire gli errori Rust personalizzati in Lambda.
- Funzione Lambda con risorse condivise: un progetto Rust che inizializza le risorse condivise prima della creazione della funzione Lambda.
- Eventi HTTP Lambda: una funzione Rust che gestisce gli eventi HTTP.
- Eventi HTTP Lambda con intestazioni CORS: una funzione Rust che utilizza Tower per inserire le intestazioni CORS.
- REST API Lambda: una REST API che utilizza Axum e Diesel per connettersi a un database PostgreSQL.

- Demo di Rust senza server: un progetto Rust che mostra l'uso delle librerie Rust di Lambda, la registrazione, le variabili di ambiente e l'SDK. AWS
- Estensione Lambda di base: un'estensione Rust che mostra come elaborare gli eventi di estensione di base.
- Estensione Amazon Data Firehose per i log Lambda: un'estensione Rust che mostra come inviare log Lambda a Firehose.

Argomenti

- Definisci i gestori di funzioni Lambda in Rust
- Utilizzo dell'oggetto contestuale Lambda per recuperare le informazioni sulla funzione Rust
- Elaborazione di eventi HTTP con Rust
- Implementazione di funzioni Lambda per Go con gli archivi di file .zip
- Utilizzo dei livelli per le funzioni Lambda in Go
- Registrare e monitorare le funzioni Lambda con Rust

Definisci i gestori di funzioni Lambda in Rust

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Questa pagina descrive come lavorare con i gestori di funzioni Lambda in Rust, tra cui l'inizializzazione del progetto, le convenzioni di denominazione e le migliori pratiche. Questa pagina include anche un esempio di funzione Rust Lambda che raccoglie informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon Simple Storage Service (S3). Per ulteriori informazioni su come distribuire una funzione dopo averla scritta, consulta. the section called “Implementazione di archivi di file .zip”

Argomenti

- Configurazione del progetto Rust Handler
- Esempio di codice della funzione Rust Lambda
- Definizioni di classe valide per i gestori Rust
- Convenzioni di denominazione dei gestori
- Definizione e accesso all'oggetto evento di input
- Accesso e utilizzo dell'oggetto contestuale Lambda
- Usando il nel tuo gestore AWS SDK per Rust
- Accesso alle variabili d'ambiente
- Utilizzo dello stato condiviso
- Best practice di codice per funzioni Lambda in Rust

Configurazione del progetto Rust Handler

Quando si lavora con le funzioni Lambda in Rust, il processo prevede la scrittura del codice, la compilazione e la distribuzione degli artefatti compilati in Lambda. Il modo più semplice per configurare un progetto di gestione Lambda in Rust consiste nell'utilizzare il AWS Lambda Runtime for Rust. Nonostante il nome, AWS Lambda Runtime for Rust non è un runtime gestito nello stesso senso in cui lo è in Lambda for Python, Java o Node.js. Invece, il AWS Lambda Runtime for Rust è un crate (`lambda_runtime`) che supporta la scrittura di funzioni Lambda in Rust e l'interfacciamento AWS Lambda con l'ambiente di esecuzione.

Usa il seguente comando per installare Cargo Lambda, un'estensione open source di terze parti per lo strumento da riga di comando Cargo che semplifica la creazione e l'implementazione di funzioni Rust Lambda:

```
cargo install cargo-lambda
```

Dopo aver installato correttamente`cargo-lambda`, usa il seguente comando per inizializzare un nuovo progetto di gestore di funzioni Rust Lambda:

```
cargo lambda new example-rust
```

Quando esegui questo comando, l'interfaccia a riga di comando (CLI) ti pone un paio di domande sulla tua funzione Lambda:

- Funzione HTTP: se intendi richiamare la tua funzione tramite API Gateway o un URL di funzione, rispondi Sì. Altrimenti, rispondi No. Nel codice di esempio in questa pagina, invochiamo la nostra funzione con un evento JSON personalizzato, quindi rispondiamo No.
- Tipo di evento: se intendi utilizzare una forma di evento predefinita per richiamare la funzione, seleziona il tipo di evento previsto corretto. Altrimenti, lasciate vuota questa opzione. Nel codice di esempio in questa pagina, invochiamo la nostra funzione con un evento JSON personalizzato, quindi lasciamo vuota questa opzione.

Dopo che il comando è stato eseguito correttamente, accedi alla directory principale del tuo progetto:

```
cd example-rust
```

Questo comando genera un `generic_handler.rs` file e un `main.rs` file nella `src` directory. `generic_handler.rs`Può essere usato per personalizzare un gestore di eventi generico. Il `main.rs` file contiene la logica principale dell'applicazione. Il `Cargo.toml` file contiene metadati sul pacchetto ed elenca le sue dipendenze esterne.

Esempio di codice della funzione Rust Lambda

Il seguente esempio di codice della funzione Rust Lambda raccoglie informazioni su un ordine, produce una ricevuta in un file di testo e inserisce questo file in un bucket Amazon S3.

Example Funzione Lambda **`main.rs`**

```
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
}
```

Questo file `main.rs` contiene le sezioni seguenti:

- `use`istruzioni: usale per importare casse e metodi Rust richiesti dalla tua funzione Lambda.
- `#[derive(Deserialize, Serialize)]`: definisci la forma dell'evento di input previsto in questa struttura Rust.
- `async fn function_handler(event: LambdaEvent<Value>) -> Result<String, Error>`: questo è il metodo dell'handler principale, che contiene la logica principale dell'applicazione.
- `async fn upload_receipt_to_s3 (...)`: Questo è un metodo di supporto a cui fa riferimento il metodo principale`function_handler`.
- `#[tokio::main]`: Questa è una macro che segna il punto di ingresso di un programma Rust. Imposta anche un runtime Tokio, che consente al `main()` metodo di utilizzare`async`/`await`ed eseguire in modo asincrono.
- `async fn main() -> Result<(), Error>`: La `main()` funzione è il punto di ingresso del codice. Al suo interno, specifichiamo `function_handler` come metodo di gestione principale.

File Cargo.toml di esempio

Il seguente `Cargo.toml` file accompagna questa funzione.

```
[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"] }
```

Affinché questa funzione funzioni correttamente, il suo ruolo di esecuzione deve consentire l'`s3:PutObject`azione. Inoltre, assicuratevi di definire la variabile di `RECEIPT_BUCKET` ambiente. Dopo una chiamata riuscita, il bucket Amazon S3 dovrebbe contenere un file di ricevuta.

Definizioni di classe valide per i gestori Rust

Nella maggior parte dei casi, le firme dei gestori Lambda definite in Rust avranno il seguente formato:

```
async fn function_handler(event: LambdaEvent<T>) -> Result<U, Error>
```

Per questo gestore:

- Il nome di questo gestore è. `function_handler`
- L'input singolare al gestore è evento ed è di tipo. `LambdaEvent<T>`
  - `LambdaEvent`è un wrapper che proviene dalla cassa. `lambda_runtime` L'utilizzo di questo wrapper consente di accedere all'oggetto di contesto, che include metadati specifici di Lambda come l'ID della richiesta della chiamata.
  - Tè il tipo di evento deserializzato. Ad esempio, questo può essere`serde_json::Value`, che consente al gestore di accettare qualsiasi input JSON generico. In alternativa, questo può essere un tipo come `ApiGatewayProxyRequest` se la funzione si aspettasse un tipo di input specifico e predefinito.
- Il tipo restituito dal gestore è. `Result<U, Error>`
  - Uè il tipo di output deserializzato. Udeve implementare il `serde::Serialize` tratto in modo che Lambda possa convertire il valore restituito in JSON. Ad esempio, U può essere un tipo `String` semplice o una struttura personalizzata purché implementata. `serde_json::Value`

`Serialize` Quando il codice raggiunge un'istruzione Ok (U), ciò indica che l'esecuzione è riuscita e la funzione restituisce un valore di tipoU.

- Quando il codice rileva un errore (ad esempio`Err(Error)`), la funzione registra l'errore in Amazon CloudWatch e restituisce una risposta di errore di tipo. `Error`

Nel nostro esempio, la firma del gestore ha il seguente aspetto:

```
async fn function_handler(event: LambdaEvent<Value>) -> Result<String, Error>
```

Altre firme valide del gestore possono presentare quanto segue:

- Omissione del `LambdaEvent` wrapper: se si omette`LambdaEvent`, si perde l'accesso all'oggetto di contesto Lambda all'interno della funzione. Di seguito è riportato un esempio di questo tipo di firma:

  ```
  async fn handler(event: serde_json::Value) -> Result<String, Error>
  ```

- Utilizzo del tipo di unità come input — Per Rust, puoi usare il tipo di unità per rappresentare un input vuoto. Questo è comunemente usato per funzioni con invocazioni periodiche e programmate. Di seguito è riportato un esempio di questo tipo di firma:

  ```
  async fn handler(_: ()) -> Result<Value, Error>
  ```

Convenzioni di denominazione dei gestori

I gestori Lambda in Rust non hanno rigide restrizioni di denominazione. Sebbene sia possibile utilizzare qualsiasi nome per il gestore, i nomi delle funzioni in Rust sono generalmente in. `snake_case`

Per applicazioni più piccole, come in questo esempio, puoi usare un singolo `main.rs` file per contenere tutto il codice. Per progetti più grandi, `main.rs` deve contenere il punto di ingresso alla funzione, ma è possibile disporre di file aggiuntivi per separare il codice in moduli logici. Ad esempio, potresti avere la seguente struttura di file:

```
/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
```

Definizione e accesso all'oggetto evento di input

JSON è il formato di input più comune e standard per le funzioni Lambda. In questo esempio, la funzione prevede un input simile a quanto segue:

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

In Rust, puoi definire la forma dell'evento di input previsto in una struttura. In questo esempio, definiamo la seguente struttura per rappresentare un`Order`:

```
#[derive(Deserialize, Serialize)]
struct Order {
    order_id: String,
    amount: f64,
    item: String,
}
```

Questa struttura corrisponde alla forma di input prevista. In questo esempio, la `#[derive(Deserialize, Serialize)]` macro genera automaticamente codice per la serializzazione e la deserializzazione. Ciò significa che possiamo deserializzare il tipo JSON di input generico nella nostra struttura utilizzando il metodo. `serde_json::from_value()` Ciò è illustrato nelle prime righe dell'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)?;
    ...
}
```

È quindi possibile accedere ai campi dell'oggetto. Ad esempio, `order.order_id` recupera il valore di `order_id` dall'input originale.

Tipi di eventi di input predefiniti

Nella cassa sono disponibili molti tipi di eventi di input predefiniti. `aws_lambda_events` Ad esempio, se intendi richiamare la tua funzione con API Gateway, inclusa la seguente importazione:

```
use aws_lambda_events::event::apigw::ApiGatewayProxyRequest;
```

Quindi, assicurati che il tuo gestore principale utilizzi la seguente firma:

```
async fn handler(event: LambdaEvent<ApiGatewayProxyRequest>) -> Result<String, Error> {
    let body = event.payload.body.unwrap_or_default();
    ...
}
```

Fai riferimento alla cassa aws_lambda_events per ulteriori informazioni su altri tipi di eventi di input predefiniti.

Accesso e utilizzo dell'oggetto contestuale Lambda

L'oggetto contesto: contiene informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione. In Rust, il wrapper include l'oggetto context. `LambdaEvent` Ad esempio, è possibile utilizzare l'oggetto context per recuperare l'ID di richiesta della chiamata corrente con il seguente codice:

```
async fn function_handler(event: LambdaEvent<Value>) -> Result<String, Error> {
    let request_id = event.context.request_id;
    ...
}
```

Per ulteriori informazioni sulla copia di oggetti, consulta la sezione the section called “Contesto”.

Usando il nel tuo gestore AWS SDK per Rust

Spesso, utilizzerai le funzioni Lambda per interagire o aggiornare altre AWS risorse. Il modo più semplice per interfacciarsi con queste risorse consiste nell'utilizzare il AWS SDK per Rust.

Per aggiungere dipendenze SDK alla tua funzione, aggiungile nel tuo `Cargo.toml` file. Ti consigliamo di aggiungere solo le librerie necessarie per la tua funzione. Nel codice di esempio precedente, abbiamo usato il`aws_sdk_s3::Client`. Nel `Cargo.toml` file, puoi aggiungere questa dipendenza aggiungendo la seguente riga nella `[dependencies]` sezione:

```
aws-sdk-s3 = "1.78.0"
```

> **Note**
>
> Questa potrebbe non essere la versione più recente. Scegli la versione appropriata per la tua applicazione.

Quindi, importa le dipendenze direttamente nel tuo codice:

```
use aws_sdk_s3::{Client, primitives::ByteStream};
```

Il codice di esempio inizializza quindi un client Amazon S3 come segue:

```
let config = aws_config::load_defaults(aws_config::BehaviorVersion::latest()).await;
let s3_client = Client::new(&config);
```

Dopo aver inizializzato il client SDK, puoi utilizzarlo per interagire con altri servizi. AWS Il codice di esempio richiama l'`PutObject`API Amazon S3 nella funzione `upload_receipt_to_s3` helper.

Accesso alle variabili d'ambiente

Nel codice dell'handler, puoi fare riferimento a qualsiasi variabile di ambiente utilizzando il metodo `env::var`. In questo esempio, facciamo riferimento alla variabile di `RECEIPT_BUCKET` ambiente definita utilizzando la seguente riga di codice:

```
let bucket_name = env::var("RECEIPT_BUCKET")
    .map_err(|_| "RECEIPT_BUCKET environment variable is not set")?;
```

Utilizzo dello stato condiviso

È possibile dichiarare delle variabili condivise indipendenti dal codice del gestore della funzione Lambda. Queste variabili possono aiutarti a caricare le informazioni sullo stato durante i Fase di init, prima che la funzione riceva eventi. Ad esempio, puoi modificare il codice in questa pagina per utilizzare lo stato condiviso durante l'inizializzazione del client Amazon S3 aggiornando la funzione e la firma `main` del gestore:

```
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
```

Best practice di codice per funzioni Lambda in Rust

Segui le linee guida riportate nell'elenco seguente per utilizzare le best practice di codifica durante la creazione delle funzioni Lambda:

- Separare il gestore Lambda dalla logica principale. In questo modo è possibile creare una funzione di cui è più semplice eseguire l'unit test.
- Ridurre la complessità delle dipendenze. Preferire framework più semplici che si caricano velocemente all'avvio del contesto di esecuzione.
- Ridurre al minimo le dimensioni del pacchetto di implementazione al fine di soddisfare le esigenze di runtime. In questo modo viene ridotta la quantità di tempo necessaria per il download del pacchetto e per la relativa decompressione prima dell'invocazione.

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa

su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedere Riutilizzo delle connessioni con Keep-Alive in Node.js.

Utilizzare le variabili di ambiente per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su `0` per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare documenti non documentati e non pubblici APIs nel codice della funzione Lambda. Per i runtime AWS Lambda gestiti, Lambda applica periodicamente aggiornamenti di sicurezza e funzionalità all'interno di Lambda. APIs Questi aggiornamenti delle API interne possono essere incompatibili con le versioni precedenti e portare a conseguenze indesiderate, come errori di chiamata se la funzione dipende da questi elementi non pubblici. APIs Consulta il riferimento alle API per un elenco di quelle disponibili al pubblico. APIs

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda?.

Utilizzo dell'oggetto contestuale Lambda per recuperare le informazioni sulla funzione Rust

Quando Lambda esegue la funzione, aggiunge un oggetto di contesto a LambdaEvent quello ricevuto dal gestore. Questo oggetto fornisce proprietà con informazioni sulla chiamata, sulla funzione e sull'ambiente di esecuzione.

Proprietà del contesto

- `request_id`: l'ID della AWS richiesta generato dal servizio Lambda.
- `deadline`: la scadenza di esecuzione per la chiamata corrente in millisecondi.
- `invoked_function_arn`: il nome della risorsa Amazon (ARN) della funzione Lambda da richiamare.
- `xray_trace_id`: L'ID di AWS X-Ray traccia per la chiamata corrente.
- `client_content`: l'oggetto contestuale del client inviato dall'SDK AWS mobile. Questo campo è vuoto a meno che la funzione non venga richiamata utilizzando un SDK AWS mobile.
- `identity`: l'identità Amazon Cognito che ha richiamato la funzione. Questo campo è vuoto a meno che la richiesta di chiamata a Lambda non sia APIs stata effettuata utilizzando AWS credenziali emesse dai pool di identità di Amazon Cognito.
- `env_config`: la configurazione della funzione Lambda in base alle variabili di ambiente locali. Questa proprietà include informazioni come il nome della funzione, l'allocazione della memoria, la versione e i flussi di log.

Accesso alle informazioni relative al contesto di invocazione

Le funzioni Lambda hanno accesso ai metadata relativi al loro ambiente e alla richiesta di invocazione. L'oggetto `LambaEvent` ricevuto dal gestore delle funzioni include i metadati `context`:

```
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
}
```

Elaborazione di eventi HTTP con Rust

Amazon API Gateway APIs, Application Load Balancers e la funzione Lambda URLs possono inviare eventi HTTP a Lambda. Puoi utilizzare la cassa aws_lambda_events di crates.io per elaborare gli eventi da queste origini.

Example - Gestione della richiesta proxy di Gateway API

Tenere presente quanto segue:

- `use aws_lambda_events::apigw::{ApiGatewayProxyRequest, ApiGatewayProxyResponse}`: la cassa aws_lambda_events include molti eventi Lambda. Per ridurre i tempi di compilazione, utilizza i flag delle funzionalità per attivare gli eventi di cui hai bisogno. Esempio: `aws_lambda_events = { version = "0.8.3", default-features = false, features = ["apigw"] }`.
- `use http::HeaderMap`: questa importazione richiede l'aggiunta della cassa http alle dipendenze.

```
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
```

```
}
```

Il client di runtime Rust per Lambda fornisce anche un'astrazione su questi tipi di eventi che consente di lavorare con tipi HTTP nativi, indipendentemente dal servizio che invia gli eventi. Il codice seguente è equivalente all'esempio precedente e funziona immediatamente con la funzione Lambda URLs, Application Load Balancers e API Gateway.

> **Note**
>
> La cassa lambda_http utilizza la cassa lambda_runtime sottostante. Non è necessario eseguire l'importazione di `lambda_runtime` separatamente.

Example - Gestione delle richieste 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
}
```

Per un altro esempio di utilizzo`lambda_http`, consultate l'esempio di codice http-axum nel repository Labs. AWS GitHub

Eventi HTTP Lambda di esempio per Rust

- Eventi HTTP Lambda: una funzione Rust che gestisce gli eventi HTTP.
- Eventi HTTP Lambda con intestazioni CORS: una funzione Rust che utilizza Tower per inserire le intestazioni CORS.

- Eventi HTTP Lambda con risorse condivise: una funzione Rust che utilizza risorse condivise inizializzate prima della creazione del gestore della funzione.

Implementazione di funzioni Lambda per Go con gli archivi di file .zip

Questa pagina descrive come compilare la funzione Rust e quindi distribuire il binario compilato su Cargo Lambda AWS Lambda. Mostra anche come distribuire il binario compilato con AWS Command Line Interface e la AWS Serverless Application Model CLI.

Sections

- Prerequisiti
- Compilazione di funzioni Rust su macOS, Windows o Linux
- Implementazione del file binario di funzioni Rust con Cargo Lambda
- Richiamo della funzione Rust con Cargo Lambda

Prerequisiti

- Rust
- AWS CLI versione 2

Compilazione di funzioni Rust su macOS, Windows o Linux

I passaggi seguenti mostrano come creare il progetto per la prima funzione Lambda con Rust e compilarlo con Cargo Lambda, un'estensione open source di terze parti dello strumento da riga di comando Cargo che semplifica la creazione e l'implementazione di funzioni Rust Lambda.

1. Installa Cargo Lambda, un'estensione open source di terze parti per lo strumento da riga di comando Cargo che semplifica la creazione e l'implementazione di funzioni Rust Lambda:

   ```
   cargo install cargo-lambda
   ```

   Per altre opzioni di installazione, consulta Installazione nella documentazione di Cargo Lambda.

2. Crea la struttura del pacchetto. Questo comando crea un codice di funzione di base in `src/main.rs`. A fini di test, puoi utilizzare questo codice o sostituirlo con il tuo codice personalizzato.

   ```
   cargo lambda new my-function
   ```

3. All'interno della directory principale del pacchetto, esegui il sottocomando build per compilare il codice nella funzione.

```
cargo lambda build --release
```

(Facoltativo) Se vuoi usare AWS Graviton2 su Lambda, aggiungi il `--arm64` flag per compilare il codice per ARM. CPUs

```
cargo lambda build --release --arm64
```

4. Prima di distribuire la funzione Rust, configura le credenziali sulla tua macchina. AWS

```
aws configure
```

Implementazione del file binario di funzioni Rust con Cargo Lambda

Utilizza il sottocomando deploy per implementare il file binario compilato su Lambda. Questo comando crea un ruolo di esecuzione e quindi crea la funzione Lambda. Per specificare un ruolo di esecuzione esistente, utilizza il flag --iam-role.

```
cargo lambda deploy my-function
```

Distribuzione del file binario della funzione Rust con AWS CLI

Puoi anche distribuire il tuo file binario con. AWS CLI

1. Per compilare il pacchetto di implementazione .zip, utilizza il sottocomando build.

```
cargo lambda build --release --output-format zip
```

2. Per implementare il pacchetto .zip su Lambda, eseguire il comando create-function.

   - Per `--runtime`, specificare `provided.al2023`. Questo è un runtime solo per il sistema operativo. I runtime solo per il sistema operativo vengono utilizzati per distribuire file binari compilati e runtime personalizzati su Lambda.
   - Per `--role`, specifica l'ARN del ruolo di esecuzione.

```
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
```

Distribuzione del binario della funzione Rust con la CLI AWS SAM

Puoi anche distribuire il tuo file binario con la AWS SAM CLI.

1. Crea un AWS SAM modello con la definizione delle risorse e delle proprietà. Per `Runtime`, specificare `provided.al2023`. Questo è un runtime solo per il sistema operativo. I runtime solo per il sistema operativo vengono utilizzati per distribuire file binari compilati e runtime personalizzati su Lambda.

   Per ulteriori informazioni sulla distribuzione delle funzioni Lambda AWS SAM utilizzando, AWS::Serverless::Functionconsulta la Guida per AWS Serverless Application Model gli sviluppatori.

   Example Definizione di risorse e proprietà SAM per un file binario 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. Utilizza il sottocomando build per compilare la funzione.

```
cargo lambda build --release
```

3. Utilizza il comando sam deploy per implementare la funzione su Lambda.

```
sam deploy --guided
```

Per ulteriori informazioni sulla creazione di funzioni Rust con la AWS SAM CLI, consulta Creazione di funzioni Rust Lambda con Cargo Lambda nella Developer Guide.AWS Serverless Application Model

Richiamo della funzione Rust con Cargo Lambda

Utilizza il sottocomando invoke per testare la tua funzione con un payload.

```
cargo lambda invoke --remote --data-ascii '{"command": "Hello world"}' my-function
```

Invocare la funzione Rust con AWS CLI

Puoi anche usare il AWS CLI per richiamare la funzione.

```
aws lambda invoke --function-name my-function --cli-binary-format raw-in-base64-out --
payload '{"command": "Hello world"}' /tmp/out.txt
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Utilizzo dei livelli per le funzioni Lambda in Go

Consigliamo di non utilizzare i livelli per gestire le dipendenze per le funzioni Lambda scritte in Go. Questo perché le funzioni Lambda in Go vengono compilate in un unico eseguibile, che viene fornito a Lambda quando si distribuisce la funzione. Questo eseguibile contiene il codice di funzione compilato, insieme a tutte le sue dipendenze. L'uso dei livelli non solo complica questo processo, ma comporta anche un aumento dei tempi di avvio a freddo, poiché le funzioni devono caricare manualmente assiemi aggiuntivi in memoria durante la fase di inizializzazione.

Per utilizzare dipendenze esterne con i gestori Go, includile direttamente nel pacchetto di implementazione. In questo modo, semplifichi il processo di implementazione e sfrutti anche le ottimizzazioni integrate del compilatore Go. Per un esempio di come importare e utilizzare una dipendenza come l'SDK AWS per Go nella tua funzione, consulta the section called “Gestore”.

Registrare e monitorare le funzioni Lambda con Rust

AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto e invia i log ad Amazon. CloudWatch La funzione Lambda include un gruppo di log CloudWatch Logs e un flusso di log per ogni istanza della funzione. L'ambiente del runtime Lambda invia i dettagli su ogni richiamo al flusso di log e inoltra i log e l'output del codice della funzione. Per ulteriori informazioni, consulta Lambda invia automaticamente i log delle funzioni a CloudWatch Logs.. Per informazioni sulla configurazione dei formati di registro, consulta. Configurazione dei formati di log JSON e testo normale In questa pagina viene descritto come generare l'output del log dal codice della funzione Lambda.

Creazione di una funzione che scrive i log

Per generare i log dal codice della funzione, è possibile utilizzare qualsiasi funzione di registrazione che scriva in `stdout` o `stderr`, come la macro `println!`. L'esempio seguente utilizza `println!` per stampare un messaggio all'avvio del gestore della funzione e prima che finisca.

```
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
}
```

Registrazione avanzata con la cassa Tracing

Tracing è un framework che consente di strumentare i programmi Rust per raccogliere informazioni diagnostiche strutturate e basate sugli eventi. Questo framework fornisce utilità per personalizzare i livelli e i formati di output di registrazione, come la creazione di messaggi di log JSON strutturati. Per utilizzare questo framework, è necessario inizializzare un `subscriber` prima di implementare il gestore della funzione. Dopodiché, è possibile utilizzare macro di tracciamento come `debug`, `info` e `error` per specificare il livello di registrazione desiderato per ogni scenario.

Example - Utilizzo della cassa Tracing

Tenere presente quanto segue:

- `tracing_subscriber::fmt().json()`: quando questa opzione è inclusa, i log vengono formattati in JSON. Per utilizzare questa opzione, è necessario includere la funzionalità `json` nella dipendenza `tracing-subscriber` (ad esempio, `tracing-subscriber = { version = "0.3.11", features = ["json"] }`).
- `#[tracing::instrument(skip(event), fields(req_id = %event.context.request_id))]`: questa annotazione genera un intervallo ogni volta che viene richiamato il gestore. L'intervallo aggiunge l'ID della richiesta su ciascuna riga di log.
- `{ %first_name }`: questo costrutto aggiunge il campo `first_name` alla riga di log in cui viene utilizzato. Il valore di questo campo corrisponde alla variabile con lo stesso nome.

```
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
```

```
}
```

Quando viene richiamata questa funzione Rust, stampa due righe di log simili alle seguenti:

```
{"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"}]}
```

Procedure consigliate per l'utilizzo AWS Lambda delle funzioni

Di seguito sono indicate le best practice per l'utilizzo di AWS Lambda.

Argomenti

- Codice della funzione
- Configurazione delle funzioni
- Scalabilità delle funzioni
- Parametri e allarmi
- Utilizzo di flussi
- Best practice di sicurezza

Codice della funzione

Sfruttare il riutilizzo del contesto di esecuzione per migliorare le prestazioni della funzione. Inizializzare i client SDK e le connessioni al database all'esterno del gestore di funzioni e memorizzare localmente nella cache gli asset statici nella directory `/tmp`. Le chiamate successive elaborate dalla stessa istanza della funzione possono riutilizzare queste risorse. Ciò consente di risparmiare sui costi riducendo i tempi di esecuzione delle funzioni.

Per evitare potenziali perdite di dati tra le chiamate, non utilizzare il contesto di esecuzione per archiviare dati utente, eventi o altre informazioni con implicazioni di sicurezza. Se la funzione si basa su uno stato mutabile che non può essere archiviato in memoria all'interno del gestore, considerare la possibilità di creare una funzione separata o versioni separate di una funzione per ogni utente.

Utilizzare una direttiva keep-alive per mantenere le connessioni persistenti. Lambda elimina le connessioni inattive nel tempo. Se si tenta di riutilizzare una connessione inattiva quando si richiama una funzione, si verificherà un errore di connessione. Per mantenere la connessione persistente, utilizzare la direttiva keep-alive associata al runtime. Per un esempio, vedere Riutilizzo delle connessioni con Keep-Alive in Node.js.

Utilizzare le variabili di ambiente per passare i parametri operativi alla funzione. Se ad esempio si scrive in un bucket Amazon S3 anziché impostare come hard-coded il nome del bucket in cui si esegue la scrittura, configurare tale nome come una variabile di ambiente.

Evita di usare invocazioni ricorsive nella tua funzione Lambda, in cui la funzione si richiama da sola o avvia un processo che potrebbe richiamare nuovamente la funzione. Ciò potrebbe provocare un volume non desiderato di invocazioni della funzione e un aumento dei costi. Se noti un volume indesiderato di invocazioni, imposta immediatamente la simultaneità riservata della funzione su `0` per interrompere tutte le invocazioni della funzione mentre si aggiorna il codice.

Non utilizzare documenti non documentati e non pubblici APIs nel codice della funzione Lambda. Per i runtime AWS Lambda gestiti, Lambda applica periodicamente aggiornamenti di sicurezza e funzionalità all'interno di Lambda. APIs Questi aggiornamenti delle API interne possono essere incompatibili con le versioni precedenti e portare a conseguenze indesiderate, come errori di chiamata se la funzione dipende da questi elementi non pubblici. APIs Consulta il riferimento alle API per un elenco di quelle disponibili al pubblico. APIs

Scrivi un codice idempotente. La scrittura di un codice idempotente per le tue funzioni garantisce che gli eventi duplicati vengano gestiti allo stesso modo. Il tuo codice dovrebbe convalidare correttamente gli eventi e gestire con garbo gli eventi duplicati. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda?.

> **Note**
>
> Puoi usare Powertools per AWS Lambda rendere le funzioni idempotenti. Per ulteriori informazioni, consulta:
>
> - Python - Utilità di idempotenza
> - TypeScript - Utilità di idempotenza
> - Java - Utilità di idempotenza
> - .NET - Utilità di idempotenza

Per le best practice relative al codice specifico di un linguaggio, consulta le seguenti sezioni:

- Best practice di codice per funzioni Lambda in Node.js
- Procedure consigliate di codice per le TypeScript funzioni Lambda
- Best practice di codice per le funzioni Lambda con Python Lambda
- Best practice di codice per le funzioni Lambda con Ruby
- Best practice di codice per funzioni Lambda in Java
- Procedure consigliate di codice per le funzioni Go Lambda

- Best practice di codice per le funzioni Lambda con C#
- Best practice di codice per funzioni Lambda in Rust

Configurazione delle funzioni

La verifica delle prestazione della funzione Lambda è una fase fondamentale per garantire la scelta della configurazione delle dimensioni di memoria ottimali. Ogni aumento delle dimensioni di memoria determina un aumento equivalente della CPU disponibile per la funzione. L'utilizzo della memoria per la tua funzione è determinato per ogni richiamo e può essere visualizzato in Amazon. CloudWatch A ogni invocazione, verrà creata una voce `REPORT:`, come illustrato di seguito:

```
REPORT RequestId: 3604209a-e9a3-11e6-939a-754dd98c7be3 Duration: 12.34 ms Billed
 Duration: 100 ms Memory Size: 128 MB Max Memory Used: 18 MB
```

Se si analizza il campo `Max Memory Used:`, è possibile determinare se per la funzione è necessaria una maggiore quantità di memoria o se ne è stata riservata una quantità eccessiva.

Per trovare la configurazione di memoria giusta per le tue funzioni, ti consigliamo di utilizzare il progetto open source AWS Lambda Power Tuning. Per ulteriori informazioni, consulta AWS Lambda Power Tuning on. GitHub

Per ottimizzare le prestazioni delle funzioni, consigliamo inoltre di implementare librerie in grado di sfruttare Advanced Vector Extensions 2 (). AVX2 Ciò consente di elaborare carichi di lavoro complessi, tra cui l'inferenza di machine learning, l'elaborazione di elementi multimediali, l'elaborazione HPC (High Performance Computing), le simulazioni scientifiche e la modellazione finanziaria. Per ulteriori informazioni, consulta Creare funzioni più veloci AWS Lambda con. AVX2

Eseguire il test della funzione Lambda per determinare un valore di timeout ottimale. È importante analizzare il runtime della funzione in modo da individuare meglio gli eventuali problemi con un servizio di dipendenza che possa aumentare la simultaneità della funzione oltre le previsioni. Ciò risulta particolarmente utile quando la funzione effettua chiamate di rete a risorse che non sono in grado di gestire il dimensionamento di Lambda. Per ulteriori informazioni sul test di carico dell'applicazione, consulta Test del carico distribuito su AWS.

Utilizzare le autorizzazioni più restrittive per le impostazioni delle policy IAM. È importante comprendere le risorse e le operazioni necessarie alla funzione Lambda e limitare il ruolo di esecuzione in base a tali autorizzazioni. Per ulteriori informazioni, consulta Gestione delle autorizzazioni in AWS Lambda.

Acquisire familiarità con Quote di Lambda. Le dimensioni del payload, i descrittori di file e lo spazio / tmp sono aspetti spesso sottovalutati quando si determinano i limiti delle risorse del runtime.

Eliminare le funzioni Lambda non più utilizzate. In questo modo le funzioni non utilizzate non vengono conteggiate inutilmente ai fini del calcolo del limite delle dimensioni del pacchetto di distribuzione.

Se si sta usando Amazon Simple Queue Service come origine eventi, verificare che il valore del tempo di invocazione stimato della funzione non superi il valore Visibility Timeout (Timeout visibilità) della coda. Questo si applica a CreateFunction e UpdateFunctionConfiguration.

- Nel caso di CreateFunction, il processo di creazione della funzione AWS Lambda avrà esito negativo.
- Nel caso di UpdateFunctionConfiguration, potrebbe comportare invocazioni duplicate della funzione.

Scalabilità delle funzioni

Informati sui vincoli di throughput a monte e a valle. Sebbene le funzioni Lambda si dimensioni perfettamente in base al carico, le dipendenze a monte e a valle potrebbero non avere le stesse capacità di throughput. Se devi limitare il valore massimo di scalabilità della tua funzione, puoi configurare la simultaneità riservata sulla funzione.

Limitazione all'acceleratore incorporata. Se la tua funzione sincrona subisce una limitazione a causa del traffico che supera la velocità di scalabilità di Lambda, puoi utilizzare le seguenti strategie per migliorare la tolleranza alla limitazione:

- Utilizza timeout, nuovi tentativi e backoff con jitter. L'implementazione di queste strategie semplifica le chiamate ripetute e aiuta a garantire che Lambda possa scalare in pochi secondi per ridurre al minimo le limitazioni da parte dell'utente finale.
- Utilizza la simultaneità fornita. La simultaneità fornita è il numero di ambienti di esecuzione pre-inizializzati che Lambda assegna alla tua funzione. Lambda gestisce le richieste in arrivo utilizzando la simultaneità fornita, se disponibile. Lambda può anche scalare la funzione oltre l'impostazione di simultaneità fornita, se necessario. La configurazione della concorrenza fornita comporta costi aggiuntivi per l'account. AWS

Parametri e allarmi

Usa Utilizzo dei parametri di CloudWatch Logs con Lambda and CloudWatch Alarms invece di creare o aggiornare una metrica dal codice della funzione Lambda. In questo modo è molto più efficiente eseguire il controllo dello stato delle funzioni Lambda, perché è possibile rilevare tempestivamente i problemi nel processo di sviluppo. È possibile ad esempio configurare un allarme in base alla durata prevista dell'esecuzione della funzione Lambda per individuare colli di bottiglia o latenze attribuibili al codice della funzione.

Emetti metriche personalizzate in modo asincrono utilizzando Embedded Metric Format (EMF). Invece di effettuare chiamate API sincrone a CloudWatch, usa EMF per emettere metriche attraverso i log della funzione. Questo approccio riduce la latenza e migliora le prestazioni. L'utilità Metrics in Powertools for AWS Lambda gestisce automaticamente la formattazione EMF. Per ulteriori informazioni, vedete le utilità Python TypeScript, Java o.NET Metrics in Powertools per la documentazione. AWS Lambda Per informazioni sull'utilizzo di EMF per generare log in formato metrico, consulta Pubblicazione di log con il formato metrico incorporato nella Amazon User Guide. CloudWatch

Usa la registrazione JSON strutturata per una migliore osservabilità. La registrazione strutturata semplifica la ricerca, il filtraggio e l'analisi dei log delle funzioni. Prendi in considerazione l'utilizzo dell'utilità Logger di Powertools per AWS Lambda formattare automaticamente i log in JSON. Per ulteriori informazioni, vedere le utilità Python TypeScript, Java o.NET Logger in Powertools per la documentazione. AWS Lambda

Utilizzare la libreria di registrazione e parametri e dimensioni AWS Lambda per rilevare errori di applicazione (ad esempio, ERR, ERROR, WARNING e così via)

Utilizza AWS Cost Anomaly Detection per rilevare attività insolite sul tuo account. Cost Anomaly Detection utilizza il machine learning per monitorare continuamente i costi e l'utilizzo riducendo al minimo i falsi positivi. Cost Anomaly Detection utilizza i dati di AWS Cost Explorer, che hanno un ritardo massimo di 24 ore. Di conseguenza, possono essere necessarie fino a 24 ore per rilevare un'anomalia dopo l'utilizzo. Per iniziare a utilizzare Cost Anomaly Detection, è necessario registrarsi per Cost Explorer. Quindi, accedi a Cost Anomaly Detection.

Utilizzo di flussi

Eseguire il test con dimensioni di batch e record diverse in modo che la frequenza di polling di ogni origine eventi sia regolata in base alla velocità con cui la funzione è in grado di completare l'attività. Il

CreateEventSourceMapping BatchSize parametro controlla il numero massimo di record che possono essere inviati alla funzione con ogni chiamata. Una dimensione di batch maggiore può spesso assorbire in modo più efficiente il costo associato all'invocazione in un set di record più grande, aumentando in tal modo il throughput.

Per impostazione predefinita, Lambda richiama la funzione non appena i record sono disponibili. Se il batch che Lambda legge dall'origine eventi contiene un solo record, Lambda invia solo un record alla funzione. Per evitare di richiamare la funzione con pochi record è possibile, configurando un periodo di batch, chiedere all'origine eventi di memorizzare nel buffer i registri per un massimo di 5 minuti. Prima di richiamare la funzione, Lambda continua a leggere i registri dall'origine eventi fino a quando non ha raccolto un batch completo, fino alla scadenza del periodo di batch o fino a quando il batch non ha raggiunto il limite del payload di 6 MB. Per ulteriori informazioni, consulta Comportamento di batching.

> **Warning**
>
> Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center. AWS

Abilita la risposta batch parziale per l'elaborazione dei flussi. Durante l'elaborazione di batch di record da flussi come Kinesis o DynamoDB Streams, abilita la risposta batch parziale per consentire a Lambda di riprovare solo i record non riusciti anziché l'intero batch. Ciò migliora l'efficienza di elaborazione e riduce le rielaborazioni non necessarie. Facoltativamente, è possibile utilizzare l'utilità Batch di Powertools per AWS Lambda semplificare i modelli di elaborazione in batch.

> **Note**
>
> È possibile utilizzare Powertools AWS Lambda per l'elaborazione in batch. Per ulteriori informazioni, consulta:
>
> - Python - Elaborazione in batch
> - TypeScript - Elaborazione in Batch
> - Java - Elaborazione in batch

- .NET - Elaborazione in batch

Aumentare il throughput di elaborazione del flusso Kinesis tramite l'aggiunta di shard. Un flusso Kinesis è costituito da uno o più shard. La velocità con cui Lambda è in grado di leggere i dati da Kinesis si dimensiona linearmente con il numero di shard. L'aumento del numero di shard determina direttamente l'aumento del numero massimo di invocazioni di funzioni Lambda simultanee e può aumentare il throughput di elaborazione del flusso Kinesis. Per ulteriori informazioni sulle relazioni tra shard e invocazioni di funzioni, consulta the section called “Flussi di polling e batching”. Se si aumenta il numero di shard in un flusso Kinesis, verificare di avere scelto una chiave di partizione opportuna (consultare l'argomento relativo alle chiavi di partizione) per i dati, in modo che i record correlati appartengano agli stessi shard e che i dati siano distribuiti in modo uniforme.

Usa Amazon CloudWatch on IteratorAge per determinare se il tuo stream Kinesis è in fase di elaborazione. Ad esempio, configura un CloudWatch allarme con un'impostazione massima su 30000 (30 secondi).

Best practice di sicurezza

Monitora il tuo utilizzo AWS Lambda in relazione alle migliori pratiche di sicurezza utilizzando. AWS Security Hub CSPM Security Hub CSPM utilizza i controlli di sicurezza per valutare le configurazioni delle risorse e gli standard di sicurezza per aiutarti a rispettare vari framework di conformità. Per ulteriori informazioni sull'utilizzo di Security Hub CSPM per valutare le risorse Lambda, consulta AWS Lambda i controlli nella Guida per l'utente. AWS Security Hub CSPM

Monitora i log delle attività di rete Lambda utilizzando Amazon GuardDuty Lambda Protection. GuardDuty La protezione Lambda ti aiuta a identificare potenziali minacce alla sicurezza quando le funzioni Lambda vengono richiamate nel tuo. Account AWS Ad esempio, se una delle tue funzioni richiede un indirizzo IP associato ad attività legate alle criptovalute. GuardDuty monitora i registri delle attività di rete generati quando viene richiamata una funzione Lambda. Per ulteriori informazioni, consulta la protezione Lambda nella Amazon GuardDuty User Guide.

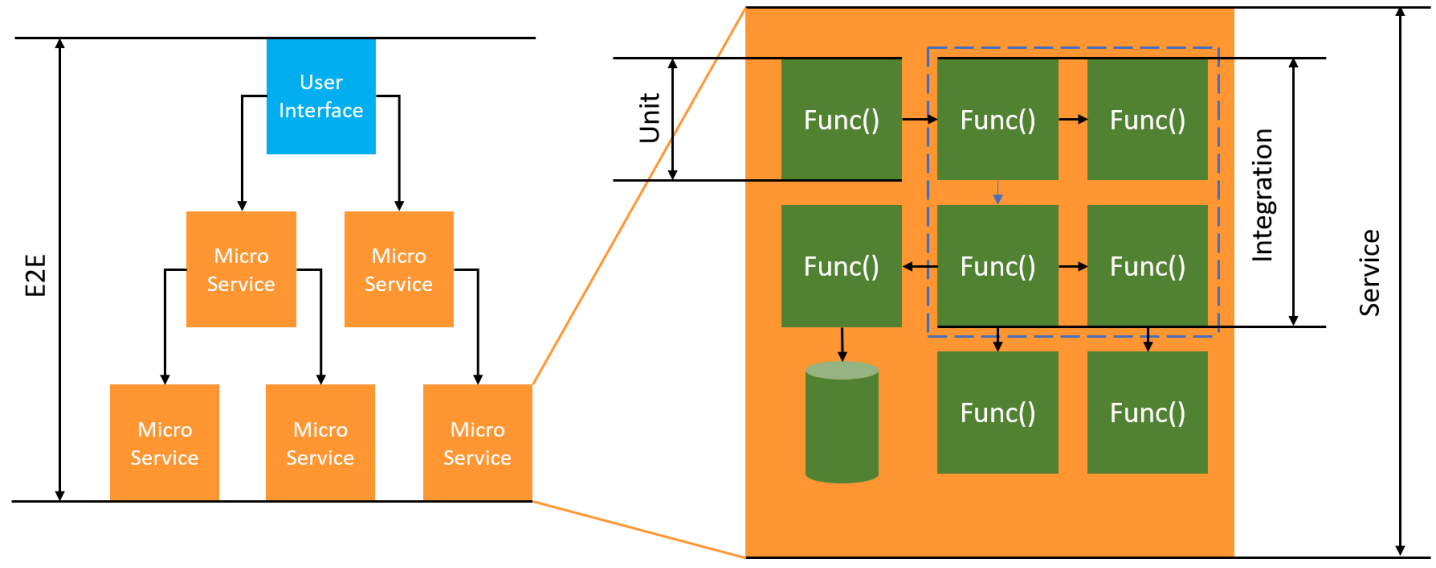
Come eseguire test di funzioni e applicazioni serverless

Sebbene il test delle funzioni serverless si basi su tipologie e tecniche di test consolidate, è importante tenere in considerazione la possibilità di effettuare test sulle applicazioni serverless nella loro interezza. I test basati sul cloud forniranno la misura più accurata della qualità sia delle funzioni sia delle applicazioni serverless.

Un'architettura applicativa serverless include servizi gestiti che forniscono funzionalità applicative critiche tramite chiamate API. Pertanto, è fondamentale che il ciclo di sviluppo preveda test automatici in grado di verificare il corretto funzionamento dell'interazione tra le funzioni e i servizi.

Se non crei test basati sul cloud, potresti riscontrare problemi dovuti alle differenze tra l'ambiente locale e quello implementato. Il processo di integrazione continua dovrebbe eseguire test su un insieme di risorse con provisioning nel cloud prima di promuovere il codice nell'ambiente di implementazione successivo, come quello di controllo qualità (QA), staging o produzione.

Continua a leggere questa breve guida per scoprire le strategie di test per le applicazioni serverless oppure visita il repository Serverless Test Samples per approfondire esempi pratici, specifici per il linguaggio e il runtime selezionati.



Per i test in ambito serverless, è necessario scrivere test unitari, test di integrazione e test end-to-end.

- Test unitari: vengono eseguiti su un blocco di codice isolato. Ad esempio, possono essere utilizzati per verificare la logica aziendale per calcolare le spese di spedizione in base a un articolo e a una destinazione specifici.
- Test di integrazione: coinvolgono due o più componenti o servizi che interagiscono, in genere in un ambiente cloud. Ad esempio, possono essere utilizzati per verificare che una funzione elabori gli eventi di una coda.
- Test end-to-end: verificano il comportamento di un'intera applicazione. Ad esempio, possono essere utilizzati per verificare che l'infrastruttura sia configurata correttamente e che gli eventi fluiscano tra i servizi come previsto per registrare l'ordine di un cliente.

Obiettivi aziendali specifici

Le verifiche delle soluzioni serverless possono richiedere un po' più di tempo per configurare test in grado di analizzare le interazioni basate sugli eventi tra i servizi. Durante la lettura della presente guida, tieni a mente i seguenti aspetti pratici per l'azienda:

- Aumenta la qualità della tua applicazione
- Riduci il tempo necessario per creare funzionalità e correggere bug

La qualità di un'applicazione dipende dai test condotti per verificarne la funzionalità in vari scenari. Per migliorare la qualità della tua applicazione, è consigliabile valutare attentamente gli scenari aziendali e automatizzare i relativi test, eseguendoli sui servizi cloud.

Rilevare i bug del software e i problemi di configurazione durante un ciclo di sviluppo iterativo può avere un impatto minimo sui costi e sulla pianificazione. Se i problemi non vengono rilevati durante lo sviluppo, la ricerca e la risoluzione in fase di produzione richiede un maggiore impegno da parte di più persone.

Una strategia di test in ambito serverless ben pianificata aumenterà la qualità del software e migliorerà i tempi di iterazione, verificando che le applicazioni e le funzioni Lambda funzionino come previsto in un ambiente cloud.

Cosa testare

Ti consigliamo di adottare una strategia di test che verifichi i comportamenti dei servizi gestiti, la configurazione del cloud, le policy di sicurezza e l'integrazione con il codice per migliorare la qualità del software. I test comportamentali, noti anche come test della scatola nera, sono progettati per

verificare il corretto funzionamento di un sistema senza la necessità di conoscere tutti i dettagli interni del software.

- Esegui test unitari per verificare la logica aziendale all'interno delle funzioni Lambda.
- Verifica che i servizi integrati siano effettivamente richiamati e che i parametri di input siano corretti.
- Controlla che un evento utilizzi tutti i servizi previsti in un flusso di lavoro dall'inizio alla fine.

Nell'architettura tradizionale basata su server, i team spesso definiscono l'ambito di test andando a includere solo il codice che viene eseguito sul server applicativo. Altri componenti, servizi o dipendenze spesso sono considerati esterni e non possono essere testati.

Le applicazioni serverless abitualmente sono costituite da piccole unità di lavoro, come le funzioni Lambda che recuperano prodotti da un database, elaborano elementi da una coda o ridimensionano un'immagine nell'archiviazione. Ogni componente viene eseguito nel proprio ambiente. Probabilmente, i team saranno responsabili di molte di queste piccole unità all'interno di una singola applicazione.

Alcune funzionalità delle applicazioni possono essere delegate interamente a servizi gestiti come Amazon S3 o create senza utilizzare alcun codice sviluppato internamente. Non è necessario testare questi servizi gestiti, ma è necessario testare l'integrazione con questi servizi.

Come testare le soluzioni serverless

Probabilmente hai familiarità con i test delle applicazioni implementate in locale: si scrivono test che vengono eseguiti su codice completamente in esecuzione sul proprio sistema operativo desktop o all'interno di container. Ad esempio, è possibile richiamare un componente del servizio Web locale con una richiesta e quindi formulare affermazioni sulla risposta.

Le soluzioni serverless vengono create a partire dal codice funzionale e dai servizi gestiti basati sul cloud, come code, database, bus di eventi e sistemi di messaggistica. Questi componenti sono tutti collegati tramite un'architettura basata sugli eventi, in cui i messaggi, chiamati eventi, fluiscono da una risorsa all'altra. Queste interazioni possono essere sincrone, ad esempio quando un servizio Web restituisce immediatamente i risultati, o un'operazione asincrona che viene completata in un secondo momento, come l'inserimento di elementi in una coda o l'avvio di una fase del flusso di lavoro. La tua strategia di test deve includere entrambi gli scenari e testare le interazioni tra i servizi. Per le interazioni asincrone, potrebbe essere necessario rilevare effetti collaterali nei componenti a valle che potrebbero non essere immediatamente osservabili.

La replica di un intero ambiente cloud, tra cui code, tabelle di database, bus di eventi, policy di sicurezza e altro ancora, non è un metodo pratico. Incontrerai inevitabilmente problemi dovuti alle differenze tra il tuo ambiente locale e gli ambienti implementati nel cloud. Le discrepanze tra i tuoi ambienti aumenteranno il tempo necessario per riprodurre e correggere i bug.

Nelle applicazioni serverless, i componenti dell'architettura solitamente esistono interamente nel cloud, quindi per sviluppare funzionalità e correggere bug è necessario eseguire test su codice e servizi nel cloud.

Tecniche di test

Nella realtà, per aumentare la qualità delle tue soluzioni, la tua strategia di test includerà probabilmente un mix di tecniche diverse. Utilizzerai test interattivi rapidi per eseguire il debug delle funzioni nella console, test unitari automatici per verificare la logica aziendale isolata, verifiche delle chiamate a servizi esterni con mock e test occasionali su emulatori che imitano un servizio.

- Test nel cloud: consente di implementare l'infrastruttura e il codice per eseguire test utilizzando effettivamente servizi, policy di sicurezza, configurazioni e parametri specifici dell'infrastruttura. I test basati sul cloud forniscono la misura più accurata della qualità del codice.

  Il debug di una funzione nella console è un modo rapido per eseguire test nel cloud. È possibile scegliere da una raccolta di esempi di eventi di test o creare un evento personalizzato per testare una funzione in modo isolato. Tramite la console puoi anche condividere gli eventi di test con il tuo team.

  Per automatizzare i test nel ciclo di vita dello sviluppo e della realizzazione, dovrai eseguire i test all'esterno della console. Per le strategie e le risorse di automazione, consulta le sezioni relative ai test per linguaggi specifici nella presente guida.
- Test con mock (chiamati anche fake): i mock sono oggetti all'interno del codice che simulano e sostituiscono un servizio esterno. I mock forniscono un comportamento predefinito per verificare le chiamate e i parametri del servizio. Un fake (o "falso") è un'implementazione fittizia che utilizza scorciatoie per semplificare o migliorare le prestazioni. Ad esempio, un oggetto di accesso ai dati falso potrebbe restituire dati da un datastore in memoria. I mock possono simulare e semplificare dipendenze complesse, ma possono anche portare a più mock per sostituire le dipendenze nidificate.
- the section called “Test localmente utilizzando AWS SAMCLI”: utilizza la AWS SAM CLI per richiamare localmente le funzioni Lambda nei contenitori Docker che utilizzano lo stesso ambiente

di runtime di Lambda. AWS Puoi testare la logica delle funzioni e l'elaborazione degli eventi senza eseguire la distribuzione nel cloud.
- the section called “Test con emulazione”: Utilizza l'integrazione LocalStack in VS Code per emulare più AWS servizi localmente per testare le integrazioni dei servizi.

Test nel cloud

I test nel cloud sono utili per tutte le fasi di test, inclusi i test unitari, i test di integrazione e i test end-to-end. Quando esegui test su codice basato sul cloud che interagisce anche con i servizi basati sul cloud, ottieni la misura più accurata della qualità del tuo codice.

Un modo conveniente per eseguire una funzione Lambda nel cloud è con un evento di test nella Console di gestione AWS. Un evento di test è un input JSON per la funzione. Se la funzione non richiede input, l'evento può essere un documento JSON vuoto (`{}`). La console fornisce eventi di esempio per una varietà di integrazioni di servizi. Dopo aver creato un evento nella console, puoi anche condividerlo con il tuo team per rendere i test più semplici e coerenti.

Scopri come eseguire il debug di una funzione di esempio nella console.

> **Note**
>
> Sebbene l'esecuzione delle funzioni nella console sia un modo rapido per eseguire il debug, l'automazione dei cicli di test è essenziale per aumentare la qualità delle applicazioni e la velocità di sviluppo.

Gli esempi di automazione dei test sono disponibili nel repository Serverless Test Samples. La seguente linea di comando esegue un esempio di test di integrazione Python automatizzato:

```
python -m pytest -s tests/integration -v
```

Sebbene il test venga eseguito localmente, interagisce con le risorse basate sul cloud. Queste risorse sono state implementate utilizzando lo strumento della linea di comando AWS Serverless Application Model e AWS SAM. Il codice di test recupera innanzitutto gli output dello stack implementato, che includono l'endpoint API, la funzione ARN e il ruolo di sicurezza. Successivamente, il test invia una richiesta all'endpoint API, che risponde con un elenco di bucket Amazon S3. Questo test viene eseguito interamente su risorse basate sul cloud per verificare che tali risorse siano implementate e protette e che funzionino come previsto.

```
========================= 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 ==========================
```

I test nel cloud offrono i seguenti vantaggi per lo sviluppo di applicazioni native del cloud:

- Puoi testare ogni servizio disponibile.
- Utilizzi sempre le API del servizio e i valori restituiti più recenti.

- Un ambiente di test cloud somiglia molto al tuo ambiente di produzione.
- I test possono verificare policy di sicurezza, Service Quotas, configurazioni e parametri specifici dell'infrastruttura.
- Ogni sviluppatore può creare rapidamente uno o più ambienti di test nel cloud.
- I test nel cloud aumentano le probabilità che il codice venga eseguito correttamente in produzione.

I test nel cloud presentano alcuni svantaggi. L'aspetto negativo più evidente dei test nel cloud è che le implementazioni in ambienti cloud richiedono in genere più tempo rispetto alle implementazioni in ambienti desktop locali.

Fortunatamente, strumenti come AWS Serverless Application Model (AWS SAM) Accelerate, AWS Cloud Development Kit (AWS CDK) watch mode e SST (terze parti) riducono la latenza associata alle iterazioni di implementazione del cloud. Questi strumenti possono monitorare l'infrastruttura e il codice e implementare automaticamente aggiornamenti incrementali nell'ambiente cloud.

> **Note**
>
> Consulta la sezione Creazione di un'infrastruttura come codice nella Guida per gli sviluppatori serverless per ottenere ulteriori informazioni su AWS Serverless Application Model, CloudFormation e AWS Cloud Development Kit (AWS CDK).

A differenza dei test locali, i test nel cloud richiedono risorse aggiuntive che possono comportare costi di servizio. La creazione di ambienti di test isolati può aumentare il carico di lavoro dei team DevOps, in particolare nelle organizzazioni con controlli rigorosi su account e infrastruttura. Tuttavia, quando si lavora con scenari infrastrutturali complessi, il costo in termini di tempo degli sviluppatori per configurare e gestire un ambiente locale complesso potrebbe essere simile (o più alto) rispetto all'utilizzo di ambienti di test usa e getta creati con gli strumenti di automazione Infrastructure as Code.

I test nel cloud, anche alla luce di queste considerazioni, restano il modo migliore per garantire la qualità delle soluzioni serverless.

Test con mock

Il test con mock è una tecnica in cui crei oggetti sostitutivi nel tuo codice per simulare il comportamento di un servizio cloud.

Ad esempio, puoi scrivere un test che utilizza un mock del servizio Amazon S3 che restituisce una risposta specifica ogni volta che viene richiamato il metodo CreateObject. Quando viene eseguito un test, il mock restituisce quella risposta programmata senza chiamare Amazon S3 o altri endpoint del servizio.

Gli oggetti fittizi (mock) sono spesso generati da un framework fittizio per ridurre gli sforzi necessari per lo sviluppo. Alcuni framework fittizi sono generici e altri sono progettati specificamente per gli SDK AWS, come Moto, una libreria Python per simulare servizi e risorse AWS.

Ricorda che gli oggetti fittizi (mock) differiscono dagli emulatori in quanto i mock vengono generalmente creati o configurati da uno sviluppatore come parte del codice di test, mentre gli emulatori sono applicazioni autonome che espongono le funzionalità allo stesso modo dei sistemi che emulano.

Di seguito puoi trovare alcuni vantaggi legati all'utilizzo dei mock:

- I mock possono simulare servizi di terze parti che sfuggono al controllo dell'applicazione, come API e provider di software as a service (SaaS), senza bisogno di accedere direttamente a tali servizi.
- I mock sono utili per testare le condizioni di errore, soprattutto quando tali condizioni sono difficili da simulare, come un'interruzione del servizio.
- Una volta configurato, il mock consente di eseguire rapidamente test locali.
- I mock possono fornire un comportamento sostitutivo praticamente per qualsiasi tipo di oggetto, quindi le strategie di simulazione possono riguardare una più ampia varietà di servizi rispetto agli emulatori.
- Quando diventano disponibili nuove funzionalità o comportamenti, i test con mock possono reagire più rapidamente. Utilizzando un framework fittizio generico, puoi simulare nuove funzionalità non appena diventa disponibile l'SDK AWS aggiornato.

I test con mock presentano i seguenti svantaggi:

- Generalmente, i mock richiedono un notevole impegno per l'impostazione e la configurazione, in particolare quando si cerca di determinare i valori restituiti da servizi diversi al fine di simulare correttamente le risposte.
- I mock devono essere scritti, configurati e gestiti dagli sviluppatori, aumentando le loro responsabilità.
- Potrebbe essere necessario avere accesso al cloud per comprendere le API e restituire i valori dei servizi.

- I mock possono essere difficili da mantenere. Quando le firme delle API cloud fittizie cambiano o gli schemi dei valori restituiti evolvono, è necessario aggiornare i mock. I mock richiedono aggiornamenti anche se si estende la logica dell'applicazione per effettuare chiamate a nuove API.
- I test che utilizzano mock potrebbero avere un esito positivo negli ambienti desktop ma fallire nel cloud. I risultati potrebbero non corrispondere all'API corrente. La configurazione e le quote del servizio non possono essere testate.
- I framework fittizi sono limitati nel testare o rilevare le limitazioni delle policy o delle quote di AWS Identity and Access Management (IAM). Sebbene i mock siano più efficaci per simulare quando l'autorizzazione viene negata o quando viene superata una quota, i test non possono determinare quale risultato si verificherà effettivamente in un ambiente di produzione.

Test localmente utilizzando AWS SAMCLI

AWS SAMCLIUsalo per testare le tue funzioni nei contenitori Docker utilizzando lo stesso ambiente di runtime di. AWS Lambda Puoi testare la logica delle funzioni e l'elaborazione degli eventi localmente senza distribuirla nel cloud. Se la tua funzione effettua chiamate API verso altriServizi AWS, tali chiamate raggiungeranno AWS risorse reali.

Di seguito puoi trovare alcuni vantaggi legati ai test con contenitori locali:

- Utilizza ambienti AWS Lambda di runtime per test accurati.
- Consente iterazioni rapide di sviluppo locale senza implementazione nel cloud.
- Supporta il debug con strumenti di sviluppo locale familiari.

I test con contenitori locali presentano le seguenti limitazioni:

- Servizio AWSIe chiamate provenienti dalla funzione interagiranno con AWS risorse reali, il che potrebbe comportare costi e influire sui dati di produzione.
- Richiede che Docker sia installato e funzionante localmente.

Test con emulazione

Gli emulatori sono applicazioni eseguite localmente che imitano AWS i servizi fornendo API e valori restituiti simili. LocalStack è uno strumento di emulazione popolare che fornisce un ambiente di sviluppo locale completo per testare le integrazioni di servizi.

LocalStack è un emulatore AWS cloud che puoi utilizzare per testare localmente le applicazioni serverless. Puoi testare le funzioni Lambda che si integrano con servizi come DynamoDB, Amazon S3 e Amazon SQS senza collegarti a servizi effettivi. AWS Puoi usare LocalStack nel AWS Toolkit for VS Code.

I vantaggi dei test con gli emulatori sono molteplici:

- Gli emulatori possono facilitare iterazioni e test di sviluppo locale rapidi.
- Gli emulatori offrono un ambiente familiare per gli sviluppatori abituati a sviluppare codice in un ambiente locale. Ad esempio, se hai familiarità con lo sviluppo di un'applicazione n-tier, potresti disporre di un motore di database e un server Web (simili a quelli in esecuzione in produzione) in esecuzione sul tuo computer locale per fornire una capacità di test rapida, locale e isolata.
- Gli emulatori non richiedono alcuna modifica all'infrastruttura cloud (come gli account cloud per sviluppatori), quindi sono facili da implementare con i modelli di test esistenti.
- Poiché gli emulatori non utilizzano AWS risorse effettive, non si verificheranno addebiti imprevisti quando si avviano più servizi o se si lasciano funzionare alcune risorse per lunghi periodi di tempo.

I test con emulatori presentano i seguenti svantaggi:

- Gli emulatori possono essere difficili da configurare e replicare, soprattutto se utilizzati nelle pipeline CI/CD. Ciò può contribuire ad aumentare il carico di lavoro del personale IT o degli sviluppatori che gestiscono il proprio software.
- Le funzionalità e le API emulate sono in genere in ritardo rispetto agli aggiornamenti del servizio. Ciò può causare errori perché il codice testato non corrisponde all'API effettiva e impedisce l'adozione di nuove funzionalità.
- Gli emulatori richiedono miglioramenti in termini di supporto, aggiornamenti, correzioni di bug e parità delle funzionalità. Questi sono di responsabilità dell'autore dell'emulatore, che potrebbe essere una società terza.
- I test che si basano sugli emulatori possono fornire risultati positivi a livello locale, ma restituire un esito negativo nel cloud a causa delle policy di sicurezza in fase di produzione, delle configurazioni tra servizi o del superamento delle quote Lambda.

Best practice

Le sezioni seguenti forniscono consigli per testare correttamente le applicazioni serverless.

Puoi trovare esempi pratici di test e automazione dei test nel repository Serverless Test Samples.

Dai priorità ai test nel cloud

I test nel cloud sono un'opzione molto valida per garantire la copertura dei test più affidabile, accurata e completa. L'esecuzione di test nel contesto del cloud controllerà in modo completo non solo la logica aziendale, ma anche le policy di sicurezza, le configurazioni dei servizi, le quote, i valori restituiti e le firme delle API più aggiornati.

Struttura il tuo codice per la testabilità

Semplifica i test e le funzioni Lambda separando il codice specifico di Lambda dalla logica aziendale principale.

Il gestore delle funzioni Lambda deve essere un adattatore agile in grado di acquisire i dati degli eventi e trasmettere solo i dettagli importanti ai metodi della logica aziendale. Con questa strategia, puoi eseguire test completi sulla tua logica aziendale senza preoccuparti dei dettagli specifici di Lambda. Le funzioni AWS Lambda non devono richiedere la configurazione di un ambiente complesso o una grande quantità di dipendenze per creare e inizializzare il componente in esame.

In generale, è consigliabile scrivere un gestore che estrae e convalida i dati dagli eventi e dagli oggetti di contesto in entrata, per poi inviare tale input ai metodi che eseguono la logica aziendale.

Accelera i cicli di feedback sullo sviluppo

Esistono strumenti e tecniche per accelerare i cicli di feedback sullo sviluppo. Ad esempio, AWS SAM Accelerate e AWS CDK watch mode riducono entrambi il tempo necessario per aggiornare gli ambienti cloud.

Gli esempi nel GitHub repository Serverless Test Sample esplorano alcune di queste tecniche.

Inoltre, ti consigliamo di creare e testare le risorse cloud il prima possibile durante lo sviluppo, anziché solo dopo aver controllato il codice sorgente. Questa pratica consente di accelerare l'esplorazione e la sperimentazione durante lo sviluppo di soluzioni. Inoltre, l'automazione dell'implementazione da una macchina di sviluppo ti aiuta a scoprire i problemi di configurazione del cloud più rapidamente e riduce gli sprechi di tempo per gli aggiornamenti e i processi di revisione del codice.

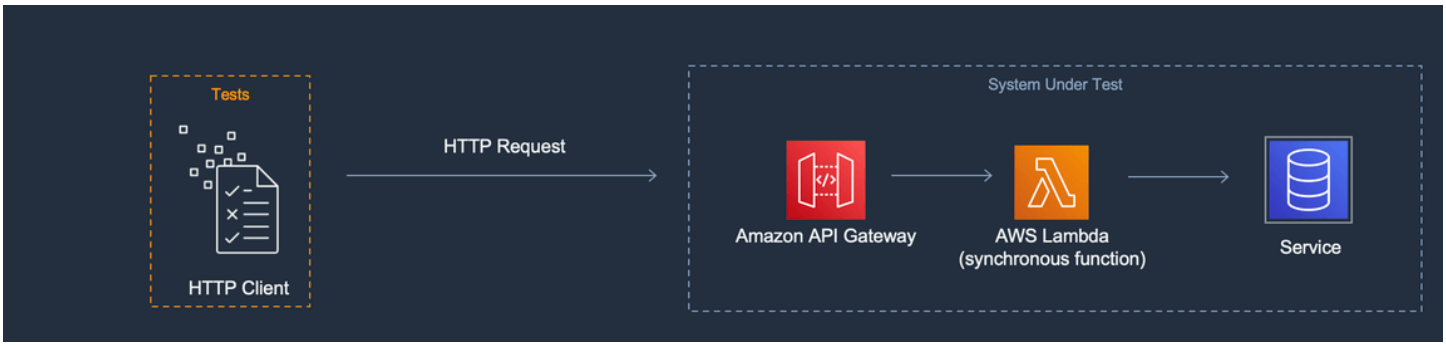
Concentrati sui test di integrazione

Quando si sviluppano applicazioni con Lambda, una buona pratica è quella di testare i componenti insieme.

I test eseguiti su due o più componenti architettonici sono chiamati test di integrazione. L'obiettivo dei test di integrazione è comprendere non solo come verrà eseguito il codice tra i componenti, ma anche come si comporterà l'ambiente che ospita il codice. I test end-to-end sono test di integrazione particolari che verificano i comportamenti in un'intera applicazione.

Per creare test di integrazione, implementa la tua applicazione in un ambiente cloud. Questa operazione può essere eseguita da un ambiente locale o tramite una pipeline CI/CD. Quindi, scrivi dei test per esercitare il sistema sottoposto a test (SUT) e convalidare il comportamento previsto.

Ad esempio, il sistema sottoposto a test potrebbe essere un'applicazione che utilizza Gateway API, Lambda e DynamoDB. Un test potrebbe effettuare una chiamata HTTP sintetica a un endpoint API Gateway e verificare che la risposta includa il payload previsto. Questo test verifica che il codice AWS Lambda sia corretto e che ogni servizio sia configurato correttamente per gestire la richiesta, comprese le autorizzazioni IAM tra di essi. Inoltre, è possibile progettare un test in cui vengano scritti record di varie dimensioni per verificare che le Service Quotas, come la dimensione massima dei record in DynamoDB, siano impostate correttamente.



Crea ambienti di test isolati

In genere, i test nel cloud richiedono ambienti di sviluppo isolati, in modo che test, dati ed eventi non si sovrappongano.

Un approccio consiste nel fornire a ogni sviluppatore un account AWS dedicato. Ciò eviterà i conflitti relativi alla denominazione delle risorse che possono verificarsi quando più sviluppatori lavorano a una base di codice condivisa, tentano di implementare risorse o richiamano un'API.

I processi di test automatizzati dovrebbero creare risorse con un nome univoco per ogni stack. Ad esempio, è possibile configurare script o file di configurazione TOML in modo che i comandi sam deploy o sam sync della CLI di AWS SAM specifichino automaticamente uno stack con un prefisso univoco.

In alcuni casi, gli sviluppatori condividono un account AWS. Ciò può essere dovuto alla presenza di risorse nello stack che sono dispendiose in termini di gestione o di provisioning e configurazione. Ad esempio, un database può essere condiviso per semplificare la configurazione e il seeding dei dati in modo corretto.

Se gli sviluppatori condividono un account, è necessario stabilire dei limiti per identificare la proprietà ed eliminare le sovrapposizioni. Un modo per farlo è anteporre ai nomi degli stack gli ID utente degli sviluppatori. Un altro approccio comune è quello di configurare stack basati su rami di codice. Grazie ai confini dei rami, gli ambienti sono isolati, ma gli sviluppatori possono comunque condividere risorse, come un database relazionale. Questo approccio è una procedura consigliata quando gli sviluppatori lavorano su più di un ramo alla volta.

I test nel cloud sono utili per tutte le fasi di test, inclusi i test unitari, i test di integrazione e i test end-to-end. Mantenere un isolamento adeguato è essenziale, ma è comunque necessario che l'ambiente di controllo qualità (QA) assomigli il più possibile all'ambiente di produzione. Per questo motivo, i team aggiungono processi di controllo delle modifiche per gli ambienti QA.

Generalmente, per gli ambienti di preproduzione e produzione vengono fissati dei limiti a livello di account per isolare i carichi di lavoro dai problemi di tipo noisy neighbor e implementare controlli di sicurezza con privilegi minimi per proteggere i dati sensibili. I carichi di lavoro hanno delle quote assegnate. Non è auspicabile che i test consumino le quote allocate per la produzione (rischio di effetto noisy neighbor) o che abbiano accesso ai dati dei clienti. I test di carico sono un'altra attività da isolare dallo stack di produzione.

In tutti i casi, gli ambienti devono essere configurati con avvisi e controlli per evitare spese inutili. Ad esempio, è possibile limitare il tipo, il livello o la dimensione delle risorse che possono essere create e impostare avvisi e-mail quando i costi stimati superano una determinata soglia.

Usa i mock per una logica aziendale isolata

I framework fittizi sono uno strumento prezioso per scrivere test unitari rapidi. Questi sono particolarmente utili quando i test riguardano logiche aziendali interne complesse, come calcoli o simulazioni matematiche o finanziarie. Cerca test unitari che prevedono un gran numero di casi di test o varianti di input, in cui tali input non modificano lo schema o il contenuto delle chiamate ad altri servizi cloud.

Il codice verificato dai test unitari con mock dovrebbe essere verificato anche dai test nel cloud. Questa operazione è consigliata perché l'ambiente su un laptop per sviluppatori o una macchina di compilazione potrebbe essere configurato in modo diverso rispetto a un ambiente di produzione

nel cloud. Ad esempio, le funzioni Lambda potrebbero utilizzare più memoria o impiegare più tempo rispetto a quanto allocato quando vengono eseguite con determinati parametri di input. Oppure il codice potrebbe includere variabili di ambiente non configurate o che non sono state configurate allo stesso modo, e le differenze potrebbero causare un comportamento diverso o un errore del codice.

Il vantaggio dei mock è minore per i test di integrazione, poiché il livello di impegno richiesto per implementare i mock necessari aumenta in proporzione al numero di punti di connessione. I test end-to-end non dovrebbero utilizzare mock, poiché questi test generalmente riguardano stati e logiche complesse che non possono essere facilmente simulati con framework fittizi.

Infine, evita di utilizzare servizi cloud fittizi per verificare la corretta implementazione delle chiamate del servizio. Al contrario, per convalidare l'implementazione a livello di comportamento, configurazione e funzionalità, è consigliabile effettuare le chiamate ai servizi cloud direttamente nell'ambiente cloud.

Utilizza gli emulatori con parsimonia

Gli emulatori possono essere utili in alcuni casi d'uso, ad esempio per un team di sviluppo con accesso a Internet limitato, inaffidabile o lento. Tuttavia, nella maggior parte dei casi, assicurati di usare gli emulatori con parsimonia.

Evitando gli emulatori, sarai in grado di creare e innovare con le funzionalità di servizio più recenti e le API aggiornate. Non rischierai di dover aspettare che i fornitori rilascino le versioni necessarie per raggiungere la parità di funzionalità. Ridurrai le spese iniziali e correnti per l'acquisto e la configurazione di più sistemi di sviluppo e macchine di compilazione. Inoltre, eviterai il problema che molti servizi cloud semplicemente non dispongono di emulatori. Pertanto, se la strategia di test dipende dall'emulazione, non si potranno utilizzare tali servizi (e, di conseguenza, occorreranno soluzioni alternative potenzialmente più costose), oppure si produrranno codice e configurazioni non testati adeguatamente.

Quando si utilizza l'emulazione per i test, è comunque necessario eseguire il test nel cloud per verificare la configurazione e testare le interazioni con i servizi cloud che possono essere simulate o ricorrere all'utilizzo di mock solo in un ambiente emulato.

Problematiche legate ai test locali

Quando si utilizzano emulatori e chiamate fittizie per eseguire i test sul desktop locale, è possibile che si verifichino incongruenze nei test man mano che il codice passa da un ambiente all'altro nella

pipeline CI/CD. I test unitari impiegati per convalidare la logica aziendale dell'applicazione sul desktop potrebbero non testare con precisione gli aspetti critici dei servizi cloud.

Gli esempi seguenti forniscono alcuni casi a cui prestare attenzione quando si eseguono test localmente con mock ed emulatori:

Esempio: la funzione Lambda crea un bucket S3

Se la logica di una funzione Lambda dipende dalla creazione di un bucket S3, un test completo dovrebbe confermare che Amazon S3 è stato chiamato e che il bucket è stato creato correttamente.

- In una configurazione di test con mock, potresti simulare una risposta con esito positivo e potenzialmente aggiungere un caso di test per gestire una risposta non esito negativo.
- In uno scenario di test di emulazione, è possibile che venga chiamata l'API CreateBucket, ma è necessario tenere presente che l'identità che effettua la chiamata locale non proverrà dal servizio Lambda. L'identità che effettua la chiamata non assumerà un ruolo di sicurezza come nel cloud, quindi verrà utilizzata invece un'autenticazione segnaposto, possibilmente con un ruolo o un'identità utente più permissivi che saranno diversi quando vengono eseguiti nel cloud.

Le configurazioni di mock ed emulazione verificheranno cosa farà la funzione Lambda se chiama Amazon S3; tuttavia, tali test non verificheranno che la funzione Lambda, così come configurata, sia in grado di creare correttamente il bucket Amazon S3. È necessario assicurarsi che al ruolo assegnato alla funzione sia associata una policy di sicurezza che consenta alla funzione di eseguire l'operazione `s3:CreateBucket`. In caso contrario, la funzione probabilmente genererà un errore qualora implementata in un ambiente cloud.

Esempio: una funzione Lambda elabora i messaggi da una coda Amazon SQS

Se una coda Amazon SQS è l'origine di una funzione Lambda, un test completo deve verificare che quando un messaggio viene inserito in una coda la funzione Lambda venga richiamata correttamente.

I test di emulazione e i test con mock sono generalmente impostati per eseguire direttamente il codice della funzione Lambda e per simulare l'integrazione con Amazon SQS passando un payload di eventi JSON (o un oggetto deserializzato) come input del gestore della funzione.

I test locali che simulano l'integrazione con Amazon SQS verificheranno cosa farà la funzione Lambda quando viene chiamata da Amazon SQS con un determinato payload, ma non

verificheranno che Amazon SQS richiami correttamente la funzione Lambda quando viene implementata in un ambiente cloud.

Di seguito troverai alcuni esempi di problemi di configurazione che potresti riscontrare con Amazon SQS e Lambda:

- Il timeout di visibilità di Amazon SQS è troppo basso e comporta più chiamate quando ne era prevista una sola.
- Il ruolo di esecuzione della funzione Lambda non consente la lettura di messaggi dalla coda (tramite `sqs:ReceiveMessage`, `sqs:DeleteMessage` o `sqs:GetQueueAttributes`).
- L'evento di esempio passato alla funzione Lambda supera la quota relativa alla dimensione dei messaggi di Amazon SQS. Pertanto, il test non è valido perché Amazon SQS non sarebbe mai in grado di inviare un messaggio di tali dimensioni.

Come mostrano questi esempi, con molta probabilità si otterranno risultati inattendibili dai test che riguardano la logica aziendale ma non le configurazioni tra i servizi cloud.

Domande frequenti

Ho una funzione Lambda che esegue calcoli e restituisce un risultato senza chiamare altri servizi. Devo davvero testarla nel cloud?

Sì. Le funzioni Lambda hanno parametri di configurazione che possono modificare l'esito del test. Tutto il codice della funzione Lambda dipende dalle impostazioni di timeout e memoria, e se tali aspetti non sono impostati correttamente potrebbero causare errori della funzione. Le policy Lambda consentono anche la registrazione degli output standard su Amazon CloudWatch. Anche se il codice non richiama direttamente CloudWatch, è necessaria l'autorizzazione per abilitare la registrazione. Questa autorizzazione richiesta non può essere simulata o emulata con precisione.

In che modo i test nel cloud possono contribuire ai test unitari? Se sono nel cloud e si connettono ad altre risorse, non si tratta di test di integrazione?

Definiamo test unitari quei test che operano su componenti architettonici isolati, ma ciò non impedisce di includere componenti che possono richiamare altri servizi o di utilizzare alcune comunicazioni di rete.

Molte applicazioni serverless dispongono di componenti architettonici che possono essere testati in modo isolato, anche nel cloud. Un esempio può essere quello di una funzione Lambda che accetta

l'input, elabora i dati e invia un messaggio a una coda Amazon SQS. Un test unitario di questa funzione verificherebbe probabilmente se i valori di input determinano la presenza di determinati valori nel messaggio in coda.

Prendi in considerazione un test scritto utilizzando lo schema Arrange, Act, Assert (organizzazione, azione, affermazione):

- Arrange: alloca le risorse (una coda per ricevere messaggi e la funzione sottoposta a test).
- Act: chiama la funzione sottoposta a test.
- Assert: recupera il messaggio inviato dalla funzione e convalida l'output.

Un approccio di test con mock implicherebbe la simulazione della coda con un oggetto fittizio in corso di elaborazione e la creazione di un'istanza durante il processo della classe o del modulo che contiene il codice della funzione Lambda. Durante la fase Assert, il messaggio in coda verrebbe recuperato dall'oggetto fittizio.

In un approccio basato sul cloud, il test creerebbe una coda Amazon SQS ai fini del test e implementerebbe la funzione Lambda con variabili di ambiente configurate per utilizzare la coda Amazon SQS isolata come destinazione di output. Dopo aver eseguito la funzione Lambda, il test recupererebbe il messaggio dalla coda di Amazon SQS.

Il test basato sul cloud eseguirebbe lo stesso codice, affermerebbe lo stesso comportamento e convaliderebbe la correttezza funzionale dell'applicazione. Tuttavia, avrebbe l'ulteriore vantaggio di poter convalidare le impostazioni della funzione Lambda: il ruolo IAM, le policy IAM e le impostazioni di timeout e di memoria della funzione.

Risorse e passaggi successivi

Utilizza le seguenti risorse per ottenere ulteriori informazioni ed esplorare esempi pratici di test.

Implementazioni di esempio

Il repository Serverless Test Samples su GitHub contiene esempi concreti di test che seguono gli schemi e le best practice descritti nella presente guida. Il repository contiene codice di esempio e procedure dettagliate dei processi di test con mock, emulazione e cloud descritti nelle sezioni precedenti. Usa questo repository per rimanere aggiornato da AWS sulle ultime linee guida per i test in ambito serverless.

Approfondimenti

Visita Serverless Land per accedere ai blog, ai video e ai corsi di formazione più recenti sulle tecnologie AWS serverless.

Inoltre, ti consigliamo di leggere i seguenti post sul blog AWS:

- Accelerazione dello sviluppo serverless con AWS SAM Accelerate (post del blog AWS)
- Aumento della velocità di sviluppo con CDK Watch (post del blog AWS)
- Simulazione delle integrazioni di servizi con AWS Step Functions Local (post del blog AWS)
- Guida introduttiva al test delle applicazioni serverless (post del blog AWS)

Strumenti

- AWS SAM – esecuzione di test e debug di applicazioni serverless
- AWS SAM – integrazione con test automatici
- Lambda: Test delle funzioni Lambda nella console Lambda

Migliorare le prestazioni di avvio con Lambda SnapStart

Lambda SnapStart può fornire prestazioni di avvio anche inferiori al secondo, in genere senza modifiche al codice della funzione. SnapStart semplifica la creazione di applicazioni altamente reattive e scalabili senza fornire risorse o implementare complesse ottimizzazioni delle prestazioni.

Il fattore che contribuisce maggiormente alla latenza di avvio (spesso definita tempo di avvio a freddo) è il tempo impiegato da Lambda per inizializzare la funzione, che include il caricamento del codice della funzione, l'avvio del runtime e l'inizializzazione del codice della funzione. Con SnapStart, Lambda inizializza la funzione quando si pubblica una versione della funzione. Lambda utilizza lo snapshot di una microVM Firecracker della memoria e dello stato del disco dell'ambiente di esecuzione inizializzato, crittografa lo snapshot e lo memorizza nella cache in maniera intelligente per ottimizzare la latenza del recupero.

Per garantire la resilienza, Lambda conserva diverse copie di ogni snapshot. Lambda aggiorna automaticamente gli snapshot e le relative copie con gli aggiornamenti di runtime e sicurezza più recenti. Quando richiami la versione della funzione per la prima volta e man mano che le chiamate aumentano, Lambda riprende i nuovi ambienti di esecuzione dallo snapshot memorizzato nella cache invece di inizializzarli da zero, migliorando così la latenza di avvio.

> **Important**
>
> Se le applicazioni dipendono dall'univocità dello stato, è necessario valutare il codice della funzione e verificare che sia resiliente alle operazioni di snapshot. Per ulteriori informazioni, consulta Gestione dell'unicità con Lambda SnapStart.

Argomenti

- Quando usare SnapStart
- Funzionalità e limitazioni supportate
- Regioni supportate
- Considerazioni sulla compatibilità
- SnapStart prezzi
- Attivazione e gestione di Lambda SnapStart
- Gestione dell'unicità con Lambda SnapStart
- Implementare il codice prima o dopo gli snapshot della funzione Lambda

- Monitoraggio per Lambda SnapStart
- Modello di sicurezza per Lambda SnapStart
- Massimizzare le prestazioni di Lambda SnapStart
- Risoluzione degli errori di SnapStart per le funzioni Lambda

Quando usare SnapStart

Lambda SnapStart è progettato per affrontare la variabilità della latenza introdotta dal codice di inizializzazione monouso, come il caricamento di dipendenze o framework dei moduli. Il completamento di queste operazioni a volte può richiedere diversi secondi durante l'invocazione iniziale. Utilizzala SnapStart per ridurre questa latenza da alcuni secondi a meno di un secondo, in scenari ottimali. SnapStart funziona meglio se utilizzato con invocazioni di funzioni su larga scala. Le funzioni richiamate di rado potrebbero non presentare gli stessi miglioramenti delle prestazioni.

SnapStart è particolarmente utile per due tipi principali di applicazioni:

- Flussi sensibili alla latenza APIs e agli utenti: le funzioni che fanno parte degli endpoint API critici o dei flussi rivolti agli utenti possono trarre vantaggio dalla latenza ridotta e dai tempi SnapStart di risposta migliorati.
- Flussi di lavoro di elaborazione dati sensibili alla latenza: i flussi di lavoro di elaborazione dati con vincoli temporali che utilizzano le funzioni Lambda possono ottenere un throughput migliore riducendo la latenza di inizializzazione delle funzioni anomale.

La simultaneità fornita mantiene le funzioni inizializzate e pronte a rispondere entro i 100 millisecondi. Utilizza la concorrenza fornita se la tua applicazione ha requisiti di latenza rigorosi per l'avvio a freddo che non possono essere soddisfatti in modo adeguato da. SnapStart

Funzionalità e limitazioni supportate

SnapStart è disponibile per i seguenti runtime gestiti da Lambda:

- Java 11 e versioni successive
- Python 3.12 e versioni successive
- .NET 8 e versioni successive. Se utilizzi il framework Lambda Annotations per.NET, esegui l'upgrade alla versione Amazon.Lambda.Annotations 1.6.0 o successiva per garantire la compatibilità con. SnapStart

Altri runtime gestiti (come `nodejs24.x` e `ruby3.4`), Runtime solo per il sistema operativo e le immagini di container non sono supportati.

SnapStart non supporta la concorrenza fornita, Amazon Elastic File System (Amazon EFS) o lo storage temporaneo superiore a 512 MB.

> **Note**
>
> È possibile utilizzare SnapStart solo versioni di funzioni pubblicate e alias che rimandano a versioni. Non è possibile utilizzare SnapStart sulla versione non pubblicata di una funzione ($LATEST).

Regioni supportate

Lambda SnapStart è disponibile in tutte le regioni commerciali ad eccezione di Asia Pacifico (Nuova Zelanda) e Asia Pacifico (Taipei).

Considerazioni sulla compatibilità

Con SnapStart, Lambda utilizza una singola istantanea come stato iniziale per più ambienti di esecuzione. Se la funzione utilizza uno dei seguenti elementi durante la fase di inizializzazione, potrebbe essere necessario apportare alcune modifiche prima di utilizzarla: SnapStart

Unicità

Se il codice di inizializzazione genera contenuti unici inclusi nello snapshot, il contenuto potrebbe non essere più unico quando viene riutilizzato in più ambienti di esecuzione. Per mantenere l'unicità durante l'utilizzo SnapStart, è necessario generare contenuti unici dopo l'inizializzazione. Ciò include segreti unici IDs e unici e l'entropia utilizzata per generare pseudocasualità. Per informazioni su come ripristinare l'unicità, consulta Gestione dell'unicità con Lambda SnapStart.

Connessioni di rete

Lo stato delle connessioni che la funzione stabilisce durante la fase di inizializzazione non è garantito quando Lambda riprende la funzione da uno snapshot. Convalida lo stato delle connessioni di rete e ristabiliscile se necessario. Nella maggior parte dei casi, le connessioni di rete stabilite da un SDK riprendono automaticamente. AWS Per altre connessioni, consulta le best practice.

Dati temporanei

Durante la fase di inizializzazione, alcune funzioni scaricano o inizializzano dati effimeri, come credenziali temporanee o timestamp memorizzati nella cache. Aggiorna i dati temporanei nel gestore delle funzioni prima di utilizzarli, anche quando non li usi. SnapStart

SnapStart prezzi

> **Note**
>
> Per i runtime gestiti da Java, non sono previsti costi aggiuntivi per SnapStart. L'addebito viene effettuato in base al numero di richieste per le funzioni, al tempo impiegato dal codice per l'esecuzione e alla memoria configurata per la funzione.

Il costo di utilizzo SnapStart include quanto segue:

- Memorizzazione nella cache: per ogni versione della funzione pubblicata con SnapStart abilitata, si pagano i costi della memorizzazione nella cache e della manutenzione dell'istantanea. Il prezzo dipende dalla quantità di memoria allocata alla funzione. L'addebito sarà per un minimo di 3 ore. L'addebito continuerà a essere effettuato finché la funzione rimarrà attiva. Utilizza l'azione ListVersionsByFunctionAPI per identificare le versioni delle funzioni, quindi utilizza DeleteFunctionper eliminare le versioni non utilizzate. Per eliminare automaticamente le versioni delle funzioni non utilizzate, consulta il modello Pulizia della versione Lambda su Serverless Land.
- Ripristino: ogni volta che un'istanza di funzione viene ripristinata da uno snapshot, si paga una tariffa di ripristino. Il prezzo dipende dalla quantità di memoria allocata alla funzione.

Come per tutte le funzioni Lambda, i costi di durata si applicano al codice eseguito nell'handler delle funzioni. Per quanto riguarda SnapStart le funzioni, i costi di durata si applicano anche al codice di inizializzazione dichiarato all'esterno del gestore, al tempo impiegato per il caricamento del runtime e a qualsiasi codice eseguito in un hook di runtime. La durata viene calcolata dal momento in cui il codice inizia a funzionare fino a quando ritorna o termina in altro modo, arrotondato al valore di 1 ms più vicino. Lambda conserva copie memorizzate nella cache dello snapshot per garantire la resilienza e applica automaticamente gli aggiornamenti software, come gli upgrade di runtime e le patch di sicurezza. Gli addebiti si applicano ogni volta che Lambda esegue nuovamente il codice di inizializzazione per applicare gli aggiornamenti software.

Per ulteriori informazioni sui costi di utilizzo SnapStart, consulta AWS Lambda la sezione Prezzi.

Attivazione e gestione di Lambda SnapStart

Per utilizzarlo SnapStart, attivalo SnapStart su una funzione Lambda nuova o esistente. Quindi, pubblica e richiama una versione della funzione.

Argomenti

- Attivazione SnapStart (console)
- Attivazione SnapStart ()AWS CLI
- Attivazione SnapStart (API)
- Lambda SnapStart e stati funzionali
- Aggiornamento di uno snapshot
- Utilizzo con SnapStart AWS SDKs
- Utilizzando SnapStart with CloudFormation, e AWS SAMAWS CDK
- Eliminazione di snapshot

Attivazione SnapStart (console)

Attivare SnapStart per una funzione

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Scegli Configuration (Configurazione), quindi scegli Configurazione generale.
4. Nel pannello General configuration (Configurazione generale), scegli Edit (Modifica).
5. Nella pagina Modifica impostazioni di base, per SnapStart, scegli Versioni pubblicate.
6. Scegli Save (Salva).
7. Pubblica una versione della funzione. Lambda inizializza il codice, crea uno snapshot dell'ambiente di esecuzione inizializzato e quindi memorizza nella cache lo snapshot per l'accesso a bassa latenza.
8. Invoca la versione della funzione.

Attivazione SnapStart ()AWS CLI

Da attivare SnapStart per una funzione esistente

1. Aggiorna la configurazione della funzione eseguendo il update-function-configurationcomando con l'--snap-startopzione.

```
aws lambda update-function-configuration \
  --function-name my-function \
  --snap-start ApplyOn=PublishedVersions
```

2. Pubblica una versione della funzione con il comando publish-version.

```
aws lambda publish-version \
  --function-name my-function
```

3. Confermate che SnapStart sia attivata per la versione della funzione eseguendo il get-function-configurationcomando e specificando il numero di versione. Il seguente esempio specifica la versione 1.

```
aws lambda get-function-configuration \
  --function-name my-function:1
```

   Se la risposta mostra che OptimizationStatusis 0n e State è`Active`, allora SnapStart viene attivata e un'istantanea è disponibile per la versione della funzione specificata.

```
"SnapStart": {
    "ApplyOn": "PublishedVersions",
    "OptimizationStatus": "On"
 },
 "State": "Active",
```

4. Richiama la versione della funzione eseguendo il comando invoke e specificando la versione. L'esempio seguente richiama la versione 1.

```
aws lambda invoke \
  --cli-binary-format raw-in-base64-out \
  --function-name my-function:1 \
  --payload '{ "name": "Bob" }' \
  response.json
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Da attivare SnapStart quando si crea una nuova funzione

1. Crea una funzione eseguendo il comando create-function con l'opzione --snap-start. Per --role, specificare il nome della risorsa Amazon (ARN) del tuo ruolo di esecuzione.

```
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. Crea una versione con il comando publish-version.

```
aws lambda publish-version \
  --function-name my-function
```

3. Confermate che SnapStart sia attivata per la versione della funzione eseguendo il get-function-configurationcomando e specificando il numero di versione. Il seguente esempio specifica la versione 1.

```
aws lambda get-function-configuration \
  --function-name my-function:1
```

Se la risposta mostra che OptimizationStatusis 0n e State è`Active`, allora SnapStart viene attivata e un'istantanea è disponibile per la versione della funzione specificata.

```
"SnapStart": {
     "ApplyOn": "PublishedVersions",
     "OptimizationStatus": "On"
  },
  "State": "Active",
```

4. Richiama la versione della funzione eseguendo il comando invoke e specificando la versione. L'esempio seguente richiama la versione 1.

```
aws lambda invoke \
  --cli-binary-format raw-in-base64-out \
  --function-name my-function:1 \
  --payload '{ "name": "Bob" }' \
  response.json
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Attivazione SnapStart (API)

Attivare SnapStart

1. Esegui una delle seguenti operazioni:
   - Crea una nuova funzione con SnapStart activated utilizzando l'azione CreateFunctionAPI con il SnapStartparametro.
   - Attiva SnapStart per una funzione esistente utilizzando l'UpdateFunctionConfigurationazione con il SnapStartparametro.
2. Pubblica una versione della funzione con l'PublishVersionazione. Lambda inizializza il codice, crea uno snapshot dell'ambiente di esecuzione inizializzato e quindi memorizza nella cache lo snapshot per l'accesso a bassa latenza.
3. Verifica che SnapStart sia attivata per la versione della funzione utilizzando l'GetFunctionConfigurationazione. Specificate un numero di versione per confermare che SnapStart è attivata per quella versione. Se la risposta mostra che OptimizationStatusè 0n e State è`Active`, allora SnapStart viene attivata ed è disponibile un'istantanea per la versione della funzione specificata.

```
"SnapStart": {
        "ApplyOn": "PublishedVersions",
        "OptimizationStatus": "On"
    },
    "State": "Active",
```

4. Richiama la versione della funzione con l'operazione Invoke.

Lambda SnapStart e stati funzionali

I seguenti stati di funzione possono verificarsi quando si utilizza SnapStart.

Pending (In attesa)

Lambda sta inizializzando il codice e acquisendo uno snapshot dell'ambiente di esecuzione inizializzato. Qualsiasi chiamata o altre operazioni API che operano sulla versione della funzione avranno esito negativo.

Attivo

La creazione dello snapshot è completa ed è possibile richiamare la funzione. Per utilizzarla SnapStart, è necessario richiamare la versione della funzione pubblicata, non la versione non pubblicata ($LATEST).

Inattivo

Lo `Inactive` stato può verificarsi quando Lambda rigenera periodicamente le istantanee delle funzioni per applicare gli aggiornamenti software. In questo caso, se la funzione non riesce a inizializzarsi, può entrare in uno stato `Inactive`.

Per le funzioni che utilizzano un runtime Java, Lambda elimina gli snapshot dopo 14 giorni senza alcuna invocazione. Se si richiama la versione della funzione dopo 14 giorni, Lambda restituisce una risposta `SnapStartNotReadyException` e inizia a inizializzare un nuovo snapshot. Attendi che la versione della funzione raggiunga lo stato `Active`, quindi richiamala di nuovo.

Non riuscito

Lambda ha riscontrato un errore durante l'esecuzione del codice di inizializzazione o la creazione dello snapshot.

Aggiornamento di uno snapshot

Lambda crea uno snapshot per ogni versione della funzione pubblicata. Per aggiornare uno snapshot, pubblica una nuova versione della funzione.

Utilizzo con SnapStart AWS SDKs

Per effettuare chiamate AWS SDK dalla tua funzione, Lambda genera un set temporaneo di credenziali assumendo il ruolo di esecuzione della funzione. Queste credenziali sono disponibili come variabili d'ambiente durante l'invocazione della funzione. Non è necessario fornire le credenziali per l'SDK direttamente nel codice. Per impostazione predefinita, la catena di fornitori di credenziali controlla in sequenza ogni punto in cui è possibile impostare le credenziali e seleziona il primo disponibile, in genere le variabili d'ambiente (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` e `AWS_SESSION_TOKEN`).

> **Note**
>
> Quando SnapStart è attivato, il runtime Lambda utilizza automaticamente le credenziali del contenitore (`AWS_CONTAINER_CREDENTIALS_FULL_URI`and`AWS_CONTAINER_AUTHORIZATION_TOKEN`) anziché le variabili di ambiente della chiave di accesso. Ciò impedisce la scadenza delle credenziali prima che la funzione venga ripristinata.

Utilizzando SnapStart with CloudFormation, e AWS SAMAWS CDK

- AWS CloudFormation: dichiara l'SnapStartentità nel modello.
- AWS Serverless Application Model (AWS SAM): dichiara la SnapStartproprietà nel modello.
- AWS Cloud Development Kit (AWS CDK): Usa il SnapStartPropertytipo.

Eliminazione di snapshot

Lambda elimina gli snapshot quando:

- Si elimina la funzione o la versione della funzione.
- Solo runtime Java: non si richiama la versione della funzione per 14 giorni. Dopo 14 giorni senza una chiamata, la versione della funzione passa allo stato Inactive (Inattivo). Se si richiama la versione della funzione dopo 14 giorni, Lambda restituisce una risposta `SnapStartNotReadyException` e inizia a inizializzare un nuovo snapshot. Attendi che la versione della funzione raggiunga lo stato Active (Attivo), quindi richiamala di nuovo.

Lambda rimuove tutte le risorse associate agli snapshot eliminati in conformità con il Regolamento generale sulla protezione dei dati (GDPR).

Gestione dell'unicità con Lambda SnapStart

Quando le chiamate aumentano su una funzione SnapStart, Lambda utilizza un singolo snapshot inizializzato per riprendere più ambienti di esecuzione. Se il codice di inizializzazione genera contenuti unici inclusi nello snapshot, il contenuto potrebbe non essere più unico quando viene riutilizzato in più ambienti di esecuzione. Per mantenere l'unicità quando si utilizza SnapStart, è necessario generare contenuti univoci dopo l'inizializzazione. Ciò include ID univoci, segreti univoci ed entropia utilizzata per generare pseudocasualità.

Di seguito sono riportate le best practice che consentono di mantenere l'unicità nel codice. Per le funzioni Java, Lambda fornisce anche uno strumento di scansione SnapStart open source che aiuta a verificare la presenza di codice che presuppone l'unicità. Se durante la fase di inizializzazione vengono generati dati univoci, è possibile utilizzare un hook di runtime per ripristinare l'unicità. Con gli hook di runtime, puoi eseguire codice specifico immediatamente prima che Lambda esegua uno snapshot o subito dopo che Lambda riprende una funzione da uno snapshot.

Evitare lo stato di salvataggio che dipende dall'unicità durante l'inizializzazione

Durante la fase di inizializzazione della funzione, evita di memorizzare nella cache dati destinati a essere univoci, ad esempio la generazione di un ID univoco per la registrazione o l'impostazione di seed per funzioni casuali. Ti consigliamo invece di generare dati univoci o impostare seed per le funzioni casuali all'interno dell'handler delle funzioni o di utilizzare un hook di runtime.

I seguenti esempi mostrano come generare un UUID nell'handler delle funzioni.

Java

Example : generazione di un ID univoco nel gestore delle funzioni

```
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 : generazione di un ID univoco nel gestore delle funzioni

```
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 : generazione di un ID univoco nel gestore delle funzioni

```
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}";
    }
}
```

Utilizza generatori di numeri pseudocasuali crittograficamente sicuri (CSPRNG)

Se l'applicazione dipende dalla casualità, si consiglia di utilizzare generatori di numeri casuali crittograficamente sicuri (CSPRNG). Oltre a OpenSSL 1.0.2, i runtime gestiti da Lambda includono anche i seguenti CSPRNG integrati:

- Java: `java.security.SecureRandom`
- Python: `random.SystemRandom`
- .NET: `System.Security.Cryptography.RandomNumberGenerator`

Il software che ottiene sempre numeri casuali da `/dev/random` o `/dev/urandom` mantiene la casualità anche con SnapStart.

Le librerie di crittografia AWS mantengono automaticamente la casualità con SnapStart a partire dalle versioni minime specificate nella tabella seguente. Se usi queste librerie con le tue funzioni Lambda, assicurati di utilizzare le seguenti versioni minime o versioni successive:

Libreria	Versione minima supportata (x86)	Versione minima supportata (ARM)
AWS libcrypto (AWS-LC)	1.16.0	1.30.0

Libreria	Versione minima supportata (x86)	Versione minima supportata (ARM)
AWS libcrypto FIPS	2.0.13	2.0.13

Se impacchetti le librerie di crittografia precedenti con le funzioni Lambda come dipendenze transitive tramite le seguenti librerie, assicurati di utilizzare le seguenti versioni minime o versioni successive:

Libreria	Versione minima supportata (x86)	Versione minima supportata (ARM)
AWS SDK for Java 2.x	2,23,20	2,26,12
Runtime comune AWS per Java	0,229,8	0,29,25
Fornitore di servizi di crittografia Amazon Corretto	2.4.1	2.4.1
FIPS Fornitore di servizi di crittografia Amazon Corretto	2.4.1	2.4.1

Gli esempi seguenti mostrano come utilizzare i CSPRNG per garantire sequenze numeriche univoche anche quando la funzione viene ripristinata da uno 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 : random.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!";
    }
}
```

Strumento di scansione SnapStart (solo Java)

Lambda fornisce uno strumento di scansione che aiuta a verificare la presenza di codice che presuppone l'unicità. Lo strumento di scansione SnapStart è un plug-in SpotBugs open source che esegue un'analisi statica in base a una serie di regole. Lo strumento di scansione consente di identificare potenziali implementazioni di codice che potrebbero infrangere i presupposti sull'unicità. Per le istruzioni di installazione e un elenco dei controlli eseguiti dallo strumento di scansione, consulta il repository aws-lambda-snapstart-java-rules su GitHub.

Per ulteriori informazioni sulla gestione dell'unicità con SnapStart, consulta Avvio più rapido con AWS Lambda SnapStart sul blog AWS Compute.

Implementare il codice prima o dopo gli snapshot della funzione Lambda

Gli hook di runtime possono essere utilizzati per implementare il codice prima che Lambda crei uno snapshot o dopo che ha ripristinato una funzione da uno snapshot. Gli hook di runtime sono utili per diversi scopi, ad esempio:

- Pulizia e inizializzazione: prima di creare uno snapshot, è possibile utilizzare un hook di runtime per eseguire operazioni di pulizia o rilascio delle risorse. Dopo il ripristino di uno snapshot, è possibile utilizzare un runtime hook per reinizializzare tutte le risorse o lo stato che non sono stati acquisiti nello snapshot.
- Configurazione dinamica: è possibile utilizzare gli hook di runtime per aggiornare dinamicamente la configurazione o altri metadati prima della creazione di uno snapshot o dopo il suo ripristino. Ciò può essere utile se la funzione deve adattarsi ai cambiamenti nell'ambiente di runtime.
- Integrazioni esterne: è possibile utilizzare gli hook di runtime per l'integrazione con servizi o sistemi esterni, ad esempio l'invio di notifiche o l'aggiornamento dello stato esterno, come parte del processo di checkpoint e ripristino.
- Ottimizzazione delle prestazioni: è possibile utilizzare gli hook di runtime per ottimizzare la sequenza di avvio della funzione, ad esempio precaricando le dipendenze. Per ulteriori informazioni, consulta Ottimizzazione prestazioni.

Le pagine seguenti spiegano come implementare gli hook di runtime per il runtime preferito.

Argomenti

- Hook di runtime SnapStart per Lambda per Java
- Hook di runtime SnapStart per Lambda per Python
- Hook di runtime SnapStart per Lambda per .NET

Hook di runtime SnapStart per Lambda per Java

Gli hook di runtime possono essere utilizzati per implementare il codice prima che Lambda crei uno snapshot o dopo che ha ripristinato una funzione da uno snapshot. Gli hook di runtime sono disponibili come parte del progetto open source Coordinated Restore at Checkpoint (CRaC). CRaC è in fase sviluppo per l'Open Java Development Kit (OpenJDK). Per un esempio di come utilizzare

CRaC con un'applicazione di riferimento, consulta il repository CRaC su GitHub. Il CRaC utilizza tre elementi principali:

- `Resource`: un'interfaccia con due metodi, `beforeCheckpoint()` e `afterRestore()`. Utilizza questi metodi per implementare il codice che desideri eseguire prima di uno snapshot e dopo un ripristino.
- `Context <R extends Resource>`: per ricevere notifiche relative ai checkpoint e ai ripristini, è necessario registrare una `Resource` con un `Context`.
- `Core`: il servizio di coordinamento, che fornisce il `Context` globale predefinito tramite il metodo statico `Core.getGlobalContext()`.

Per ulteriori informazioni su `Context` e `Resource`, consulta Pacchetto org.crac nella documentazione di CRaC.

Completa le seguenti operazioni per implementare gli hook di runtime con il pacchetto org.crac. Il runtime Lambda contiene un'implementazione del contesto CRaC personalizzata che chiama gli hook di runtime prima del checkpoint e dopo il ripristino.

Registrazione ed esecuzione di hook di runtime

L'ordine in cui Lambda esegue gli hook di runtime è determinato dall'ordine di registrazione. L'ordine di registrazione segue l'ordine di importazione, definizione o esecuzione nel codice.

- `beforeCheckpoint()`: eseguito nell'ordine di registrazione inverso
- `afterRestore()`: eseguito nell'ordine di registrazione

Assicurati che tutti gli hook registrati siano stati importati e inclusi correttamente nel codice della tua funzione. Se registri gli hook di runtime in un file o modulo separato, dovrai assicurarti che il modulo venga importato, direttamente o come parte di un pacchetto più grande, nel file handler della funzione. Se il file o il modulo non viene importato nell'handler della funzione, Lambda ignorerà gli hook di runtime.

> **Note**
>
> Quando Lambda crea uno snapshot, il codice di inizializzazione può essere eseguito per un massimo di 15 minuti. Il limite di tempo è 130 secondi o il timeout della funzione configurato (massimo 900 secondi), a seconda di quale dei due valori sia più elevato. I tuoi

hook di runtime `beforeCheckpoint()` contano ai fini del limite di tempo del codice di inizializzazione. Quando Lambda ripristina uno snapshot, il runtime deve essere caricato e gli hook di runtime `afterRestore()` devono essere completati entro il limite di timeout (10 secondi). Altrimenti, otterrai un'eccezione SnapStartTimeoutException.

Fase 1: aggiornamento della configurazione di build

Aggiungi la dipendenza `org.crac` alla configurazione della build. Nell'esempio seguente viene utilizzato Gradle. Per esempi di altri sistemi di compilazione, consulta la documentazione di 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'
}
```

Fase 2: aggiornamento del gestore Lambda

Il gestore di funzioni Lambda è il metodo nel codice della funzione che elabora gli eventi. Quando viene richiamata la funzione, Lambda esegue il metodo del gestore. La funzione viene eseguita fino a quando il gestore non restituisce una risposta, termina o scade.

Per ulteriori informazioni, consulta Definire l'handler della funzione Lambda in Java.

Il seguente gestore di esempio mostra come eseguire il codice prima del checkpoint (`beforeCheckpoint()`) e dopo il ripristino (`afterRestore()`). Questo gestore registra anche la `Resource` nel `Context` gestito dal runtime.

> **Note**
>
> Quando Lambda crea uno snapshot, il codice di inizializzazione può essere eseguito per un massimo di 15 minuti. Il limite di tempo è 130 secondi o il timeout della funzione configurato (massimo 900 secondi), a seconda di quale dei due valori sia più elevato. I tuoi hook di runtime `beforeCheckpoint()` contano ai fini del limite di tempo del codice di inizializzazione. Quando Lambda ripristina uno snapshot, il runtime (JVM) deve essere

caricato e gli hook di runtime `afterRestore()` devono essere completati entro il limite di timeout (10 secondi). Altrimenti, otterrai un'eccezione 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` mantiene solo un `WeakReference` all'oggetto registrato. Se per una `Resource` viene eseguita la rimozione di oggetti inutili (garbage collection), gli hook di runtime non vengono eseguiti. Il codice deve mantenere un forte riferimento alla `Resource` per garantire l'esecuzione dell'hook di runtime.

Ecco due esempi di modelli da evitare:

Example : oggetto senza un forte riferimento

```
Core.getGlobalContext().register( new MyResource() );
```

Example : oggetti di classi anonime

```
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 {
     // ...
    }

} );
```

Invece, mantieni un riferimento forte. Nell'esempio seguente, per la risorsa registrata non viene eseguita la rimozione di oggetti inutili (garbage collection) e gli hook di runtime vengono eseguiti in modo coerente.

Example : oggetto con un forte riferimento

```
Resource myResource = new MyResource(); // This reference must be maintained to prevent
 the registered resource from being garbage collected
Core.getGlobalContext().register( myResource );
```

Hook di runtime SnapStart per Lambda per Python

Gli hook di runtime possono essere utilizzati per implementare il codice prima che Lambda crei uno snapshot o dopo che ha ripristinato una funzione da uno snapshot. Gli hook di runtime Python sono disponibili come parte della libreria open source Snapshot Restore for Python, che è inclusa nei runtime gestiti da Python. Questa libreria fornisce due decoratori che possono essere utilizzati per definire i tuoi hook di runtime:

- `@register_before_snapshot`: per le funzioni che si desidera eseguire prima che Lambda crei uno snapshot.
- `@register_after_restore`: per le funzioni che si desidera eseguire quando Lambda riprende una funzione da uno snapshot.

In alternativa, è possibile utilizzare i seguenti metodi per registrare gli elementi chiamabili per gli hook di runtime:

- `register_before_snapshot(func, *args, **kwargs)`

- `register_after_restore(func, *args, **kwargs)`

Registrazione ed esecuzione di hook di runtime

L'ordine in cui Lambda esegue gli hook di runtime è determinato dall'ordine di registrazione:

- Prima dello snapshot: eseguito nell'ordine di registrazione inverso
- Dopo lo snapshot: eseguito nell'ordine di registrazione

L'ordine di registrazione degli hook di runtime dipende da come vengono definiti gli hook. Quando si utilizzano i decoratori (`@register_before_snapshot` e `@register_after_restore`), l'ordine di registrazione segue l'ordine di importazione, definizione o esecuzione nel codice. Se hai bisogno di un maggiore controllo sull'ordine di registrazione, usa i metodi `register_before_snapshot()` e `register_after_restore()` anziché i decoratori.

Assicurati che tutti gli hook registrati siano stati importati e inclusi correttamente nel codice della tua funzione. Se registri gli hook di runtime in un file o modulo separato, dovrai assicurarti che il modulo venga importato, direttamente o come parte di un pacchetto più grande, nel file handler della funzione. Se il file o il modulo non viene importato nell'handler della funzione, Lambda ignorerà gli hook di runtime.

> **Note**
>
> Quando Lambda crea uno snapshot, il codice di inizializzazione può essere eseguito per un massimo di 15 minuti. Il limite di tempo è 130 secondi o il timeout della funzione configurato (massimo 900 secondi), a seconda di quale dei due valori sia più elevato. I tuoi hook di runtime `@register_before_snapshot` contano ai fini del limite di tempo del codice di inizializzazione. Quando Lambda ripristina uno snapshot, il runtime deve essere caricato e gli hook di runtime `@register_after_restore` devono essere completati entro il limite di timeout (10 secondi). Altrimenti, otterrai un'eccezione SnapStartTimeoutException.

Esempio

Il seguente gestore di esempio mostra come eseguire il codice prima del checkpoint (`@register_before_snapshot`) e dopo il ripristino (`@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
```

Per altri esempi, consulta Ripristino di snapshot per Python nel repository AWS GitHub.

Hook di runtime SnapStart per Lambda per .NET

Gli hook di runtime possono essere utilizzati per implementare il codice prima che Lambda crei uno snapshot o dopo che ha ripristinato una funzione da uno snapshot. Gli hook di runtime .NET sono disponibili come parte del pacchetto Amazon.Lambda.Core (versione 2.5.0 o successiva). Questa libreria fornisce due metodi che possono essere utilizzati per definire i tuoi hook di runtime:

- `RegisterBeforeSnapshot()`: codice da eseguire prima della creazione dello snapshot
- `RegisterAfterSnapshot()`: codice da eseguire dopo la ripresa di una funzione da uno snapshot

> **Note**
>
> Se utilizzi il framework Lambda Annotations per .NET, esegui l'upgrade alla versione 1.6.0 o successiva di Amazon.Lambda.Annotations per garantire la compatibilità con SnapStart.

Registrazione ed esecuzione di hook di runtime

Registra i tuoi hook nel codice di inizializzazione. Prendi in considerazione le seguenti linee guida basate sul modello di esecuzione della tua funzione Lambda:

- Per l'approccio all'assembly eseguibile, registra i tuoi hook prima di avviare il bootstrap Lambda con `RunAsync`.

- Per l'approccio basato sulle librerie di classi, registra i tuoi hook nel costruttore della classe handler.
- Per le applicazioni ASP.NET Core, registra gli hook prima di chiamare il metodo `WebApplications.Run`.

Per registrare gli hook di runtime per SnapStart in .NET, utilizza i seguenti metodi:

```
Amazon.Lambda.Core.SnapshotRestore.RegisterBeforeSnapshot(BeforeCheckpoint);
Amazon.Lambda.Core.SnapshotRestore.RegisterAfterRestore(AfterCheckpoint);
```

Quando vengono registrati più tipi di hook, l'ordine in cui Lambda esegue gli hook di runtime è determinato dall'ordine di registrazione:

- `RegisterBeforeSnapshot()`: eseguito nell'ordine di registrazione inverso
- `RegisterAfterSnapshot()`: eseguito nell'ordine di registrazione

> **Note**
>
> Quando Lambda crea uno snapshot, il codice di inizializzazione può essere eseguito per un massimo di 15 minuti. Il limite di tempo è 130 secondi o il timeout della funzione configurato (massimo 900 secondi), a seconda di quale dei due valori sia più elevato. I tuoi hook di runtime `RegisterBeforeSnapshot()` contano ai fini del limite di tempo del codice di inizializzazione. Quando Lambda ripristina uno snapshot, il runtime deve essere caricato e gli hook di runtime `RegisterAfterSnapshot()` devono essere completati entro il limite di timeout (10 secondi). Altrimenti, otterrai un'eccezione SnapStartTimeoutException.

Esempio

La seguente funzione di esempio mostra come eseguire il codice prima del checkpoint (`RegisterBeforeSnapshot`) e dopo il ripristino (`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
        };
    }
}
```

Monitoraggio per Lambda SnapStart

È possibile monitorare le funzioni Lambda SnapStart utilizzando Amazon CloudWatch, AWS X-Ray e l'Accesso ai dati di telemetria in tempo reale per le estensioni tramite l'API Telemetry.

> **Note**
>
> Le variabili di ambiente `AWS_LAMBDA_LOG_GROUP_NAME` e `AWS_LAMBDA_LOG_STREAM_NAME` non sono disponibili nelle funzioni Lambda SnapStart.

Informazioni sul comportamento di registrazione e fatturazione con SnapStart

Esistono alcune differenze con il formato del flusso di log di CloudWatch per le funzioni SnapStart:

- Log di inizializzazione: quando viene creato un nuovo ambiente di esecuzione, il `REPORT` non include il campo `Init Duration`. Questo perché Lambda inizializza le funzioni SnapStart quando si crea una versione anziché durante l'invocazione della funzione. Per le funzioni SnapStart, il campo `Init Duration` è nel record `INIT_REPORT`. Questo record mostra i dettagli della durata di Fase di init, inclusa la durata di eventuali hook di runtime `beforeCheckpoint`.
- Log di invocazione: quando viene creato un nuovo ambiente di esecuzione, il `REPORT` include i campi `Restore Duration` e `Billed Restore Duration`:
  - `Restore Duration`: il tempo impiegato da Lambda per ripristinare uno snapshot, caricare il runtime (JVM) ed eseguire qualsiasi hook di runtime . Il processo di ripristino degli snapshot può includere il tempo dedicato ad attività esterne alla MicroVM. Questo tempo non è riportato in `Restore Duration`.
  - `Billed Restore Duration`: il tempo impiegato da Lambda per ripristinare uno snapshot, caricare il runtime (JVM) ed eseguire qualsiasi hook di runtime .

> **Note**
>
> Come per tutte le funzioni Lambda, i costi di durata si applicano al codice eseguito nell'handler delle funzioni. Per le funzioni SnapStart, i costi di durata si applicano anche

al codice di inizializzazione dichiarato all'esterno dell'handler, al tempo necessario per il caricamento del runtime e a qualsiasi codice eseguito in un hook di runtime.

La durata dell'avviamento a freddo è la somma di `Restore Duration` e `Duration`.

L'esempio seguente è una query Lambda Insights che restituisce i percentili di latenza per le funzioni SnapStart. Per ulteriori informazioni sulle query Lambda Insights, consulta Esempio di flusso di lavoro utilizzando query per risolvere i problemi di una funzione.

```
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
```

Tracciamento attivo X-Ray per SnapStart

È possibile utilizzare X-Ray per tracciare le richieste alle funzioni Lambda SnapStart. I sottosegmenti X-Ray per le funzioni SnapStart presentano alcune differenze:

- Non esiste un sottosegmento `Initialization` per le funzioni SnapStart.
- Il sottosegmento mostra il tempo impiegato da Lambda per ripristinare uno snapshot, caricare il runtime (JVM) ed eseguire qualsiasi hook di runtime . Il processo di ripristino degli snapshot può includere il tempo dedicato ad attività esterne alla MicroVM. Questa volta è riportato nel segmento secondario `Restore`. Non ti viene addebitato il tempo trascorso fuori dalla microVM per il ripristino di una snapshot.

Eventi dell'API di telemetria per SnapStart

Lambda invia i seguenti eventi SnapStart all'API di telemetria:

- `platform.restoreStart`: mostra l'ora in cui è iniziata la fase `Restore`.
- `platform.restoreRuntimeDone`: indica se la fase `Restore` è riuscita correttamente. Lambda genera questo messaggio quando il runtime invia una richiesta API di runtime `restore/next`. Ci sono tre stati possibili: riuscito, errore e timeout.
- `platform.restoreReport`: mostra quanto è durata la fase `Restore` e quanti millisecondi sono stati fatturati durante questa fase.

Parametri di Gateway Amazon API e della funzione URL

Se crei un'API Web utilizzando Gateway API, puoi utilizzare il parametro IntegrationLatency per misurare la latenza end-to-end (il tempo che intercorre tra il momento in cui Gateway API inoltra una richiesta al backend e il momento in cui riceve una risposta dal backend).

Se utilizzi l'URL di una funzione Lambda, puoi utilizzare il parametro URLRequestLatency per misurare la latenza end-to-end (il tempo che intercorre tra il momento in cui la funzione URL riceve una richiesta e il momento in cui restituisce una risposta).

Modello di sicurezza per Lambda SnapStart

Lambda SnapStart supporta la crittografia dei dati inattivi. Lambda crittografa gli snapshot con una AWS KMS key. Per impostazione predefinita, Lambda utilizza un Chiave gestita da AWS. Se questo comportamento predefinito si adatta al flusso di lavoro, non è necessario impostare altro. Altrimenti, puoi utilizzare l'opzione `--kms-key-arn` nel comando create-function o update-function-configuration per fornire una chiave gestita dal cliente AWS KMS. È possibile eseguire questa operazione per controllare la rotazione della chiave KMS o per soddisfare i requisiti dell'organizzazione per la gestione delle chiavi KMS. Le chiavi gestite dal cliente sono soggette a costi AWS KMS standard. Per ulteriori informazioni, consultare Prezzi di AWS Key Management Service.

Quando si elimina una funzione o la versione di una funzione SnapStart, tutte le richieste `Invoke` relative a tale funzione o versione della funzione avranno esito negativo. Lambda rimuove tutte le risorse associate agli snapshot eliminati in conformità con il Regolamento generale sulla protezione dei dati (GDPR).

Massimizzare le prestazioni di Lambda SnapStart

Argomenti

- Ottimizzazione prestazioni
- Best practice per la rete

Ottimizzazione prestazioni

Per massimizzare i vantaggi di SnapStart, prendi in considerazione i seguenti consigli di ottimizzazione del codice per il tuo runtime.

> **Note**
>
> SnapStart funziona al meglio se utilizzato con richiami di funzioni su larga scala. Le funzioni richiamate di rado potrebbero non presentare gli stessi miglioramenti delle prestazioni.

Java

Per aumentare al massimo i vantaggi di SnapStart, consigliamo di precaricare le dipendenze e inizializzare le risorse che contribuiscono alla latenza di avvio nel codice di inizializzazione anziché nell'handler della funzione. Ciò sposta la latenza associata al caricamento intensivo delle classi dal percorso di chiamata, ottimizzando le prestazioni di avvio con SnapStart.

Se non riesci a precaricare le dipendenze o le risorse durante l'inizializzazione, ti consigliamo di precaricarle con invocazioni fittizie. Per far ciò, aggiorna il codice del gestore delle funzioni come mostrato nell'esempio seguente dalla funzione pet store nel repository GitHub di 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

Per massimizzare i vantaggi di SnapStart, concentrati sull'organizzazione efficiente del codice e sulla gestione delle risorse all'interno delle tue funzioni Python. Come linea guida generale, esegui attività di calcolo complesse durante la fase di inizializzazione. Questo approccio elimina dal percorso di invocazione le operazioni che richiedono molto tempo, migliorando le prestazioni complessive delle funzioni. Per implementare questa strategia in modo efficace, consigliamo le seguenti best practice:

- Importa le dipendenze al di fuori dell'handler della funzione.

- Crea istanze `boto3` al di fuori dell'handler.
- Inizializza le risorse o le configurazioni statiche prima che l'handler venga richiamato.
- Prendi in considerazione l'utilizzo di un hook di runtime prima dello snapshot per attività che richiedono molte risorse come il download di file esterni, il precaricamento di framework come Django o il caricamento di modelli di machine learning.

Example : ottimizzazione della funzione Python per 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

Per ridurre i tempi di compilazione just-in-time (JIT) e di caricamento degli assembly, considera la possibilità di richiamare l'handler della funzione da un hook di runtime `RegisterBeforeCheckpoint`. Grazie al funzionamento della compilazione su più livelli .NET, otterrai risultati ottimali richiamando l'handler più volte, come illustrato nell'esempio seguente.

> **Important**
>
> Assicurati che l'invocazione della funzione fittizia non produca effetti collaterali indesiderati, come l'avvio di transazioni commerciali.

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);
    }
}
```

Best practice per la rete

Lo stato delle connessioni che la funzione stabilisce durante la fase di inizializzazione non è garantito quando Lambda riprende la funzione da uno snapshot. Nella maggior parte dei casi, le connessioni di rete stabilite da un SDK AWS riprendono automaticamente. Per altre connessioni, consigliamo le seguenti best practice.

Ristabilire le connessioni di rete

Ristabilisci sempre le connessioni di rete quando la funzione riprende da uno snapshot. Si consiglia di ristabilire le connessioni di rete nel gestore delle funzioni. In alternativa, puoi usare un hook di runtime dopo il ripristino.

Non utilizzare hostname come identificatore univoco dell'ambiente di esecuzione

Si consiglia di non utilizzare `hostname` per identificare l'ambiente di esecuzione come nodo o container univoco nelle applicazioni. Con SnapStart, viene utilizzato un singolo snapshot come stato iniziale per più ambienti di esecuzione. Tutti gli ambienti di esecuzione restituiscono lo stesso valore `hostname` per `InetAddress.getLocalHost()` (Java), `socket.gethostname()` (Python) e `Dns.GetHostName()` (.NET). Per le applicazioni che richiedono un'identità dell'ambiente di esecuzione o un valore `hostname` univoco, si consiglia di generare un ID univoco nel gestore della funzione. Oppure, utilizza un hook di runtime dopo il ripristino per generare un ID univoco, quindi utilizza l'ID univoco come identificatore per l'ambiente di esecuzione.

Evitare di collegare connessioni a porte di origine fisse

Si consiglia di evitare di associare le connessioni di rete a porte di origine fisse. Le connessioni vengono ristabilite quando una funzione riprende da uno snapshot e le connessioni di rete legate a una porta di origine fissa potrebbero non riuscire.

Evitare di usare la cache DNS Java

Le funzioni Lambda memorizzano già nella cache le risposte DNS. Se con SnapStart si utilizza un'altra cache DNS, è possibile che si verifichino dei timeout di connessione quando la funzione riprende da uno snapshot.

La classe `java.util.logging.Logger` può abilitare indirettamente la cache DNS JVM. Per sovrascrivere le impostazioni predefinite, imposta networkaddress.cache.ttl su 0 prima dell'inizializzazione di `logger`. Esempio:

```
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);
}
```

Per evitare errori `UnknownHostException` nel runtime di Java 11, si consiglia di impostare `networkaddress.cache.negative.ttl` su 0. Nei runtime di Java 17 e successivi, questa operazione non è necessaria. È possibile impostare questa proprietà per una funzione Lambda con la variabile di ambiente `AWS_LAMBDA_JAVA_NETWORKADDRESS_CACHE_NEGATIVE_TTL=0`.

La disabilitazione della cache DNS JVM non disabilita la cache DNS gestita da Lambda.

Risoluzione degli errori di SnapStart per le funzioni Lambda

Questa pagina affronta i problemi più comuni che si verificano durante l'utilizzo di Lambda SnapStart, inclusi errori di creazione di snapshot, errori di timeout ed errori interni del servizio.

SnapStartNotReadyException

Errore: si è verificato un errore (SnapStartNotReadyException) durante la chiamata dell'operazione Invoke20150331: Lambda sta inizializzando la funzione. Sarà pronta per essere richiamata una volta che lo stato della funzione diventerà ATTIVO.

Cause comuni

Questo errore si verifica quando si prova a richiamare una versione della funzione che si trova nello stato `Inactive`. La versione della funzione diventa `Inactive` quando non viene richiamata per 14 giorni o quando Lambda riavvia periodicamente l'ambiente di esecuzione

Risoluzione

Attendi che la versione della funzione raggiunga lo stato `Active`, quindi richiamala di nuovo.

SnapStartTimeoutException

Problema: si riceve una `SnapStartTimeoutException` quando si prova richiamare una versione della funzione SnapStart.

Cause comuni

Durante la fase di ripristino, Lambda ripristina il runtime Java ed esegue qualsiasi hook di runtime successivo al ripristino. Se un hook di runtime post-ripristino viene eseguito per più di 10 secondi, la fase `Restore` scade e viene visualizzato un errore quando si prova a richiamare la funzione. I problemi relativi alla connessione di rete e alle credenziali possono anche causare un timeout per la fase `Restore`.

Risoluzione

Controlla i log di CloudWatch della funzione per eventuali errori di timeout verificatisi durante la fase di ripristino. Assicurati che tutti gli hook post-ripristino vengano completati in meno di 10 secondi.

Example Log CloudWatch

```
{ "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"}
```

Errore interno del servizio 500

Errore: Lambda non è riuscito a creare un nuovo snapshot perché è stato raggiunto il limite di creazione simultanea di snapshot.

Cause comuni

Un errore 500 è un errore interno al servizio Lambda stesso e non un problema relativo alla funzione o al codice. Questi errori sono spesso intermittenti.

Risoluzione

Prova a pubblicare nuovamente la versione della funzione.

401 - Autorizzazione negata

Errore: token di sessione o chiave di intestazione errati

Cause comuni

Questo errore si verifica quando si utilizzano l'archivio dei parametri AWS Systems Manager e l'estensione Gestione dei segreti AWS con Lambda SnapStart.

Risoluzione

L'archivio di parametri AWS Systems Manager e l'estensione Gestione dei segreti AWS non sono compatibili con SnapStart. L'estensione genera credenziali con cui comunicare con Gestione dei segreti AWS durante l'inizializzazione della funzione, il che causa errori di credenziali scadute se utilizzate con SnapStart.

UnknownHostException (Java)

Errore: impossibile eseguire la richiesta HTTP: il certificato per `abc.us-east-1.amazonaws.com` non corrisponde a nessuno dei nomi alternativi del soggetto.

Cause comuni

Le funzioni Lambda memorizzano già nella cache le risposte DNS. Se con SnapStart si utilizza un'altra cache DNS, è possibile che si verifichino dei timeout di connessione quando la funzione riprende da uno snapshot.

Risoluzione

Per evitare errori `UnknownHostException` nel runtime di Java 11, si consiglia di impostare `networkaddress.cache.negative.ttl` su 0. Nei runtime di Java 17 e successivi, questa operazione non è necessaria. È possibile impostare questa proprietà per una funzione Lambda con la variabile di ambiente `AWS_LAMBDA_JAVA_NETWORKADDRESS_CACHE_NEGATIVE_TTL=0`.

Errori di creazione snapshot

Errore: AWS Lambda non ha potuto richiamare la funzione SnapStart. Se l'errore persiste, controlla i log CloudWatch della tua funzione per individuare eventuali errori di inizializzazione.

Risoluzione

Controlla i log di Amazon CloudWatch della tua funzione per verificare i timeout degli hook di runtime prima del checkpoint. Puoi anche provare a pubblicare una nuova versione della funzione, che a volte può risolvere il problema.

Frequenza di creazione degli snapshot

Problema: quando si pubblica una nuova versione della funzione, la funzione rimane nello stato `Pending` per un lungo periodo.

Cause comuni

Quando Lambda crea uno snapshot, il codice di inizializzazione può essere eseguito per un massimo di 15 minuti. Il limite di tempo è 130 secondi o il timeout della funzione configurato (massimo 900 secondi), a seconda di quale dei due valori sia più elevato.

Se la funzione è collegata a un VPC, Lambda potrebbe anche dover creare interfacce di rete prima che la funzione diventi `Active`. Se provi a richiamare la versione della funzione mentre la funzione è `Pending`, potresti ottenere un `ResourceConflictException` 409. Se la funzione viene richiamata utilizzando un endpoint Gateway Amazon API, potresti ricevere un errore 500 in API Gateway.

Risoluzione

Attendi almeno 15 minuti per l'inizializzazione della versione della funzione prima di richiamarla.

Isolamento degli inquilini

Utilizza la modalità di isolamento dei tenant quando è necessaria l'elaborazione isolata delle richieste per singoli utenti finali o tenant che richiamano una funzione Lambda. Questa funzionalità semplifica la creazione di applicazioni multi-tenant che elaborano codice o dati specifici del tenant, come le piattaforme SaaS per l'automazione del flusso di lavoro o l'esecuzione del codice, eliminando la necessità di gestire le risorse funzionali specifiche del tenant e la logica di routing delle richieste.

Le applicazioni multi-tenant hanno requisiti di isolamento rigorosi durante l'esecuzione di codice o l'elaborazione di dati per singoli tenant o utenti finali. Con la modalità di isolamento dei tenant, Lambda utilizza un identificatore tenant specificato dal cliente per indirizzare le richieste agli ambienti di esecuzione delle funzioni sottostanti, garantendo che gli ambienti di esecuzione di una funzione vengano utilizzati solo per servire le chiamate dall'utente finale o dal tenant specificato. Gli ambienti di esecuzione delle funzioni di Lambda sfruttano la virtualizzazione Firecracker per fornire l'isolamento dei carichi di lavoro.

Quando una funzione che utilizza la modalità di isolamento del tenant riceve una chiamata con un identificatore tenant, Lambda tenta innanzitutto di individuare un ambiente di esecuzione disponibile associato a quell'identificatore tenant. Se non esistono ambienti di esecuzione, Lambda crea e assegna un nuovo ambiente di esecuzione a quel tenant. Man mano che le chiamate di funzione con l'identificatore tenant specificato aumentano, Lambda individua o crea nuovi ambienti di esecuzione, se necessario.

Argomenti

- Quando utilizzare la modalità di isolamento dei tenant
- Funzionalità e limitazioni supportate
- Supportato Regioni AWS
- Considerazioni
- Prezzi
- Modalità di isolamento
- Abilitazione dell'isolamento dei tenant per le funzioni Lambda
- Richiamo di funzioni Lambda con isolamento del tenant
- Accesso all'identificatore del tenant nel codice della funzione Lambda
- Monitoraggio delle funzioni Lambda con isolamento dei tenant
- Risoluzione dei problemi di isolamento dei tenant per le funzioni Lambda

Quando utilizzare la modalità di isolamento dei tenant

Utilizza la modalità di isolamento dei tenant quando devi servire più utenti finali o tenant utilizzando una singola funzione Lambda, assicurando al contempo che gli ambienti di esecuzione utilizzati per elaborare le chiamate per i singoli tenant rimangano isolati l'uno dall'altro. Questo rigoroso isolamento degli ambienti di esecuzione è necessario per le applicazioni multi-tenant che:

- Esegui codice fornito dall'utente finale: il mantenimento di ambienti di esecuzione separati per i singoli tenant può limitare l'impatto dell'esecuzione di codice fornito dall'utente che potrebbe essere errato o dannoso.
- Elabora dati specifici del tenant: il mantenimento di ambienti di esecuzione separati per i singoli tenant può impedire l'esposizione di dati sensibili specifici del tenant ad altri tenant.

Richieste di invocazione multiple provenienti dallo stesso tenant possono riutilizzare lo stesso ambiente di esecuzione delle funzioni, riducendo gli avviamenti a freddo e migliorando i tempi di risposta per le applicazioni sensibili alla latenza.

Funzionalità e limitazioni supportate

La modalità di isolamento dei tenant non è supportata con le funzioni che utilizzano function, provisioned concurrency o. URLs SnapStart Puoi inviare richieste a una funzione isolata dal tenant utilizzando chiamate sincrone, chiamate asincroneo utilizzando Amazon API Gateway come trigger di eventi.

Supportato Regioni AWS

La modalità di isolamento dei tenant è supportata in tutte le regioni commerciali ad eccezione dell'Asia Pacifico (Nuova Zelanda).

Considerazioni

Quando usi l'isolamento dei tenant con le tue funzioni Lambda, tieni presente quanto segue:

- Configurazione immutabile: l'isolamento dei tenant è una proprietà immutabile della funzione. Può essere abilitato solo durante la creazione di una funzione.

- Parametro tenant-id richiesto: le funzioni che utilizzano la modalità di isolamento del tenant devono essere richiamate con un parametro. `tenant-id` L'omissione di questo parametro causerà l'esito negativo delle invocazioni delle funzioni.
- Il ruolo di esecuzione si applica a tutti i tenant: le chiamate da tutti i tenant utilizzano le autorizzazioni definite nel ruolo di esecuzione della funzione Lambda.
- Concorrenza: non ci sono modifiche al comportamento di concorrenza o di scalabilità della funzione quando si utilizza l'isolamento dei tenant. Lambda impone un limite di 2.500 ambienti di esecuzione isolati dal tenant (attivi o inattivi) per ogni 1.000 esecuzioni simultanee configurate per la funzione Lambda.

Prezzi

Ti viene addebitato un costo quando Lambda crea un nuovo ambiente di esecuzione isolato dal tenant. Il prezzo dipende dalla quantità di memoria allocata alla funzione e dall'architettura della CPU utilizzata. Per ulteriori informazioni, consultare Prezzi di AWS Lambda.

Modalità di isolamento

La tabella seguente illustra le differenze tra le funzioni Lambda con e senza isolamento dei tenant.

Funzionalità	Con isolamento dei tenant	Senza isolamento degli inquilini
Tipo di isolamento	Isolamento a livello di inquilino	Isolamento a livello di funzione
Riutilizzo dell'ambiente	Gli ambienti di esecuzione non vengono mai riutilizzati tra diversi tenant	Gli ambienti di esecuzione potrebbero essere riutilizzati tra invocazioni della stessa funzione
Isolamento dei dati	I dati degli altri inquilini non sono accessibili	I dati delle precedenti chiamate della stessa versione della funzione potrebbero essere accessibili
Partenze a freddo	Più partenze a freddo dovute agli ambienti specifici degli inquilini	Meno avviamenti a freddo grazie al riutilizzo dell'ambiente

Funzionalità	Con isolamento dei tenant	Senza isolamento degli inquilini
Prezzi	Costo aggiuntivo oltre al prezzo standard Lambda	Prezzi Lambda standard

Abilitazione dell'isolamento dei tenant per le funzioni Lambda

Per attivare la modalità di isolamento del tenant, crea una nuova funzione Lambda. Non è possibile abilitare l'isolamento dei tenant sulle funzioni esistenti.

Argomenti

- Abilitazione dell'isolamento dei tenant (console)
- Abilitazione dell'isolamento dei tenant ()AWS CLI
- Abilitazione dell'isolamento dei tenant (API)
- Abilitare l'isolamento dei tenant ()CloudFormation

Abilitazione dell'isolamento dei tenant (console)

Per creare una funzione Lambda utilizzando la console

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli Crea funzione.
3. Scegli Crea da zero.
4. Nel riquadro Informazioni base, per Nome funzione inserisci **`image-analysis`**.
5. Per Runtime, scegli uno dei runtime Lambda supportati.
6. In configurazioni aggiuntive, modalità di isolamento Tenant, seleziona Abilita.
7. Controlla le impostazioni e scegli Crea funzione.

Abilitazione dell'isolamento dei tenant ()AWS CLI

Crea una funzione con l'isolamento dei tenant

Quando crei una nuova funzione utilizzando la CLI, aggiungi l'`--tenancy-config '{"TenantIsolationMode": "PER_TENANT"}'` opzione alla tua richiesta create-function. Esempio:

```
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"}'
```

Abilitazione dell'isolamento dei tenant (API)

Per abilitare l'isolamento dei tenant utilizzando l'API Lambda

1. Crea una nuova funzione con l'isolamento dei tenant abilitato utilizzando l'azione CreateFunctionAPI con il parametro. `TenancyConfig`
2. Verifica che l'isolamento del tenant sia abilitato per la funzione utilizzando l'azione GetFunctionConfiguration. Se la risposta indica che `TenantIsolationMode` è così`PER_TENANT`, l'isolamento del tenant è abilitato per la funzione:

   ```
   "TenancyConfig": {
           "TenantIsolationMode": "PER_TENANT"
       }
   ```

Richiama la versione della funzione con l'operazione Invoke. Per ulteriori informazioni, consulta Richiamo di funzioni Lambda con isolamento del tenant.

Abilitare l'isolamento dei tenant ()CloudFormation

Il CloudFormation modello seguente crea una nuova funzione Lambda con l'isolamento dei tenant abilitato:

```
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
```

Richiamo di funzioni Lambda con isolamento del tenant

Quando si richiama una funzione con l'isolamento del tenant abilitato, è necessario fornire un parametro. `tenant-id` Questo parametro garantisce che la chiamata della funzione venga elaborata in un ambiente di esecuzione dedicato a quel tenant specifico.

Richiamo di funzioni con tenant isolation ()AWS CLI

Invocazione sincrona

Per le chiamate sincrone, aggiungi il parametro al tuo comando Invoke: `--tenant-id`

```
aws lambda invoke \
    --function-name image-analysis \
    --tenant-id blue \
    response.json
```

Invocazione asincrona

Per le chiamate asincrone, includi entrambi i parametri e: `--tenant-id --invocation-type Event`

```
aws lambda invoke \
    --function-name image-analysis \
    --tenant-id blue \
    --invocation-type Event \
    response.json
```

Richiamo di funzioni con isolamento dei tenant (API)

Quando utilizzi direttamente l'azione Invoke API, includi l'identificatore del tenant utilizzando il `X-Amzn-Tenant-Id` parametro nella richiesta.

Esempio di richiesta 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"
}
```

Richiamo di funzioni con isolamento del tenant (API Gateway)

Quando si utilizza API Gateway REST APIs per attivare funzioni Lambda isolate dal tenant, è necessario configurare API Gateway per mappare le proprietà della richiesta del client all'`X-Amz-Tenant-Id`intestazione prevista da Lambda. API Gateway utilizza l'azione Invoke API di Lambda, che richiede il passaggio dell'ID tenant tramite l'`X-Amz-Tenant-Id`intestazione HTTP. Puoi configurare API Gateway per iniettare questa intestazione HTTP nella richiesta di chiamata Lambda con un valore ottenuto dalle proprietà della richiesta del client come intestazioni HTTP, parametri di query o parametri di percorso. È necessario mappare la proprietà della richiesta del client prima di poter sovrascrivere l'intestazione. `X-Amz-Tenant-Id`

> **Note**
>
> Non è possibile utilizzare HTTP APIs per richiamare funzioni Lambda isolate dal tenant perché non è possibile sovrascrivere l'intestazione. `X-Amz-Tenant-Id`

Utilizzo delle intestazioni di richiesta

Configura l'integrazione con API Gateway per mappare un'intestazione personalizzata dalla richiesta del client all'`X-Amz-Tenant-Id`intestazione. L'esempio seguente mostra una richiesta client con un'`x-tenant-id`intestazione:

```
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"
}
```

Nella configurazione del metodo API Gateway, è necessario:

1. Abilitare il parametro di intestazione della richiesta del client (ad esempio,`method.request.header.x-tenant-id`)
2. Mappare l'intestazione del client all'intestazione di integrazione Lambda utilizzando `integration.request.header.X-Amz-Tenant-Id`

Utilizzo dei parametri di interrogazione

Allo stesso modo, puoi mappare i parametri della query all'`X-Amz-Tenant-Id`intestazione:

```
GET /api/process?tenant-id=blue&data=sample HTTP/1.1
Host: your-api-id.execute-api.us-east-1.amazonaws.com
```

Configura il metodo per abilitare il parametro di query e mapparlo all'intestazione di integrazione.

Richiamo di funzioni con isolamento dei tenant (SDK)

Quando si utilizza AWS SDKs per richiamare funzioni isolate dal tenant, includi l'identificatore del tenant nella richiesta di invocazione.

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);
    }
}
```

Accesso all'identificatore del tenant nel codice della funzione Lambda

Quando la funzione Lambda ha l'isolamento del tenant abilitato, l'identificatore tenant utilizzato per richiamare la funzione viene reso disponibile all'interno dell'oggetto context passato al gestore della funzione. È possibile utilizzare questo identificatore per implementare funzionalità di logica, monitoraggio e debug specifiche del tenant.

Argomenti

- Accesso all'identificatore del tenant
- Modelli di utilizzo comuni
- Monitoraggio e debug

Accesso all'identificatore del tenant

L'identificatore del tenant è disponibile tramite la `tenantId` proprietà dell'oggetto context. Si noti che questa proprietà è disponibile durante la fase di invocazione, non durante la fase di inizializzazione.

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);
        }
    }
}
```

Modelli di utilizzo comuni

Ecco alcuni modi comuni per utilizzare l'identificatore tenant nel codice della funzione:

Configurazione specifica per il tenant

Usa l'ID tenant per caricare configurazioni o impostazioni specifiche del tenant:

```
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)
```

Accesso ai dati specifico del tenant

Utilizza l'ID del tenant per garantire l'isolamento dei dati e il controllo degli accessi:

```
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)
```

Monitoraggio e debug

L'identificatore del tenant viene incluso automaticamente nei log Lambda quando è abilitata la registrazione JSON, semplificando il monitoraggio e il debug dei problemi specifici del tenant. Puoi anche utilizzare l'ID tenant per metriche e tracciamento personalizzati.

Example Metriche personalizzate con ID tenant

L'esempio seguente mostra come utilizzare l'ID tenant per creare CloudWatch metriche specifiche per il tenant per monitorare i modelli di utilizzo e le prestazioni per tenant:

```
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)
```

Monitoraggio delle funzioni Lambda con isolamento dei tenant

Puoi monitorare le funzioni Lambda isolate dai tenant utilizzando CloudWatch AWS X-Ray Amazon e accedendo ai dati di telemetria in tempo reale per le estensioni utilizzando l'API di telemetria.

Comprendere la registrazione per la modalità isolata dei tenant

Per le funzioni che utilizzano l'isolamento dei tenant, Lambda include automaticamente l'identificatore del tenant nei registri delle funzioni quando è abilitata la registrazione JSON, semplificando il monitoraggio e il debug di problemi specifici del tenant. Lambda crea un flusso di CloudWatch log separato per ogni ambiente di esecuzione. Puoi utilizzare CloudWatch Logs Insights per trovare i flussi di log che appartengono a un particolare tenant filtrandoli per identificatore del tenant:

```
fields @logStream, @message
| filter tenantId=='BlueTenant' or record.tenantId=='BlueTenant'
| stats count() as logCount by @logStream
| sort @timestamp desc
```

È inoltre possibile utilizzare questo parametro per recuperare i log specifici del tenant in tutti i flussi di log:

```
fields @message
| filter tenantId=='BlueTenant' or record.tenantId=='BlueTenant'
| limit 1000
```

La `tenantId` proprietà è inclusa per gli eventi delle piattaforme (come `platform.start` e`platform.report`) e i log personalizzati stampati nel codice della funzione, come illustrato di seguito:

```
{
    "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"
}
```

Risoluzione dei problemi di isolamento dei tenant per le funzioni Lambda

Questa pagina affronta i problemi più comuni che si verificano quando si utilizza l'isolamento dei tenant per. AWS Lambda

InvalidParameterValueException

Errore: configurazione dell'ID tenant non specificata o passata a una funzione quando l'isolamento del tenant non è abilitato.

Cause comuni

Questo errore si verifica quando si richiama una funzione isolata dal tenant senza un ID tenant o si richiama una funzione con un ID tenant. non-tenant-isolated

Risoluzione

Aggiungi un ID tenant se la funzione ha l'isolamento del tenant abilitato o rimuovi l'ID tenant se la funzione non ha l'isolamento del tenant abilitato.

TooManyRequestsException

Errore: tariffa superata

Cause comuni

Oltre alla limitazione della velocità basata sul numero massimo di esecuzioni simultanee e sulla velocità di scalabilità delle funzioni, Lambda limita il numero massimo di ambienti di esecuzione con riconoscimento del tenant (attivi o inattivi) che possono esistere contemporaneamente a 2.500 per ogni 1.000 esecuzioni simultanee della funzione.

Risoluzione

Per risolvere questo problema, puoi ridurre la frequenza con cui vengono effettuate le richieste di invocazione con identificatori di tenant univoci, implementare nuovi tentativi con backoff e jitter o richiedere un aumento del limite di concomitanza delle funzioni.

Richiamare Lambda con eventi di altri servizi AWS

Alcuni Servizi AWS possono richiamare direttamente le funzioni Lambda utilizzando i trigger. Questi servizi inviano eventi a Lambda e la funzione viene richiamata immediatamente quando si verifica l'evento specificato. I trigger sono adatti per eventi discreti ed elaborazione in tempo reale. Quando crei un trigger utilizzando la console Lambda, la console interagisce con il AWS servizio corrispondente per configurare la notifica degli eventi su quel servizio. Il trigger viene effettivamente archiviato e gestito dal servizio che genera gli eventi, non da Lambda.

Gli eventi sono dati strutturati nel formato JSON. La struttura di JSON varia a seconda del servizio che la genera e del tipo di evento, ma contengono tutte i dati necessari alla funzione per elaborare l'evento.

Una funzione può avere più trigger. Ogni trigger agisce come un client che invoca la funzione in modo indipendente e ogni evento che Lambda invia alla funzione contiene dati solo da un trigger. Lambda converte il documento di evento in un oggetto e lo passa al gestore funzione.

A seconda del servizio, l'invocazione basata sugli eventi può essere sincrona o asincrona.

- Per la chiamata sincrona, il servizio che genera l'evento attende la risposta della funzione. Tale servizio definisce i dati che la funzione deve restituire nella risposta. Il servizio controlla la strategia di errore, ad esempio se riprovare in caso di errori.
- Per la chiamata asincrona, Lambda inserisce l'evento in una coda prima di passarlo alla funzione. Quando Lambda accoda l'evento, invia immediatamente una risposta riuscita al servizio che ha generato l'evento. Dopo che la funzione elabora l'evento, Lambda non restituisce una risposta al servizio generatore di eventi.

Creazione di un trigger

Il modo più semplice per creare un trigger consiste nell'utilizzare la console Lambda. Quando crei un trigger utilizzando la console, Lambda aggiunge automaticamente le autorizzazioni richieste alla policy basata sulle risorse della funzione.

Per creare un trigger utilizzando la console Lambda

1. Aprire la pagina Funzioni della console Lambda.
2. Seleziona la funzione per cui desideri creare un trigger.

3. Nel riquadro Panoramica della funzione, scegli Aggiungi trigger.
4. Seleziona il AWS servizio a cui desideri richiamare la tua funzione.
5. Compila le opzioni nel riquadro Configurazione trigger e scegli Aggiungi. A seconda della Servizio AWS funzione scelta per richiamare la funzione, le opzioni di configurazione del trigger saranno diverse.

Servizi che possono richiamare le funzioni Lambda

La tabella seguente elenca i servizi che possono richiamare le funzioni Lambda.

Servizio	Metodo di chiamata
Amazon Managed Streaming per Apache Kafka	Strumento di mappatura dell'origine degli eventi
Apache Kafka gestito dal cliente	Strumento di mappatura dell'origine degli eventi
Gateway Amazon API	Chiamata sincrona basata su eventi
AWS CloudFormation	Chiamata asincrona basata su eventi
CloudWatch Registri Amazon	Chiamata asincrona basata su eventi
AWS CodeCommit	Chiamata asincrona basata su eventi
AWS CodePipeline	Chiamata asincrona basata su eventi
Amazon Cognito	Chiamata sincrona basata su eventi
AWS Config	Chiamata asincrona basata su eventi
Amazon Connect	Chiamata sincrona basata su eventi
Amazon DocumentDB	Strumento di mappatura dell'origine degli eventi
Amazon DynamoDB	Strumento di mappatura dell'origine degli eventi

Servizio	Metodo di chiamata
Elastic Load Balancer (Application Load Balancer)	Chiamata sincrona basata su eventi
Amazon EventBridge (CloudWatch Eventi)	Basata sugli eventi; invocazione asincrona (router di eventi), invocazione sincrona o asincrona (pipe e pianificazioni)
AWS IoT	Chiamata asincrona basata su eventi
Amazon Kinesis	Strumento di mappatura dell'origine degli eventi
Amazon Data Firehose	Chiamata sincrona basata su eventi
Amazon Lex	Chiamata sincrona basata su eventi
Amazon MQ	Strumento di mappatura dell'origine degli eventi
Amazon Simple Email Service	Chiamata asincrona basata su eventi
Amazon Simple Notification Service	Chiamata asincrona basata su eventi
Amazon Simple Queue Service	Strumento di mappatura dell'origine degli eventi
Amazon Simple Storage Service (Amazon S3)	Chiamata asincrona basata su eventi
Batch di Amazon Simple Storage Service	Chiamata sincrona basata su eventi
Secrets Manager	Rotazione del segreto
AWS Step Functions	Basata sugli eventi; invocazione sincrona o asincrona
Amazon VPC Lattice	Chiamata sincrona basata su eventi

Usare Lambda con Apache Kafka

Lambda supporta Apache Kafka come origine eventi. Apache Kafka è una piattaforma di streaming di eventi open source progettata per gestire pipeline di dati e applicazioni di streaming ad alto rendimento e in tempo reale. Esistono due modi principali per utilizzare Lambda con Apache Kafka:

- the section called “MSK”— Amazon Managed Streaming for Apache Kafka (Amazon MSK) è un servizio completamente gestito da. AWS Amazon MSK aiuta ad automatizzare la gestione dell'infrastruttura Kafka, inclusi il provisioning, l'applicazione di patch e la scalabilità.
- the section called “Apache Kafka gestito dal cliente”— In AWS terminologia, un cluster autogestito include cluster Kafka non ospitati.AWS Ad esempio, puoi ancora utilizzare Lambda con un cluster Kafka ospitato presso un provider non cloud come Confluent AWS Cloud o Redpanda.

Quando decidi tra Amazon MSK e Apache Kafka autogestito, considera le tue esigenze operative e i requisiti di controllo. Amazon MSK è la scelta migliore se desideri aiutarti AWS a gestire rapidamente una configurazione Kafka scalabile e pronta per la produzione con un sovraccarico operativo minimo. Semplifica la sicurezza, il monitoraggio e l'elevata disponibilità, aiutandoti a concentrarti sullo sviluppo di applicazioni piuttosto che sulla gestione dell'infrastruttura. D'altra parte, Apache Kafka autogestito è più adatto per casi d'uso eseguiti su ambienti non AWS ospitati, inclusi i cluster locali.

Argomenti

- Uso di Lambda con Amazon MSK
- Utilizzo di Lambda con Apache Kafka gestito dal cliente
- Modalità di ridimensionamento del poller di eventi di Apache Kafka in Lambda
- Sondaggi di Apache Kafka e posizioni iniziali dello streaming in Lambda
- ID gruppo di consumer personalizzabile
- Filtraggio eventi da origini eventi Amazon MSK e Apache Kafka autogestito
- Utilizzo dei registri degli schemi con le sorgenti di eventi Kafka in Lambda
- Elaborazione a bassa latenza per sorgenti di eventi Kafka
- Configurazione dei controlli di gestione degli errori per le sorgenti di eventi Kafka
- Acquisizione di batch scartati per un'origine eventi di Amazon MSK e Apache Kafka autogestito
- Usare un argomento di Kafka come destinazione in caso di errore
- Risoluzione degli errori relativi allo strumento di mappatura dell'origine degli eventi di Apache Kafka autogestito

Uso di Lambda con Amazon MSK

Amazon Managed Streaming for Apache Kafka (Amazon MSK) è un servizio completamente gestito che puoi utilizzare per creare ed eseguire applicazioni che utilizzano Apache Kafka per elaborare dati di streaming. Amazon MSK semplifica la configurazione, la scalabilità e la gestione dei cluster Kafka. Amazon MSK semplifica inoltre la configurazione dell'applicazione per più zone di disponibilità e per la sicurezza con AWS Identity and Access Management (IAM).

Questo capitolo spiega come utilizzare un cluster Amazon MSK come origine di eventi per la funzione Lambda. Il processo generale per l'integrazione di Amazon MSK con Lambda prevede i seguenti passaggi:

1. Configurazione del cluster e della rete: per prima cosa, configura il tuo cluster Amazon MSK. Ciò include la configurazione di rete corretta per consentire a Lambda di accedere al cluster.
2. Configurazione della mappatura delle sorgenti degli eventi: quindi, crea la risorsa di mappatura delle sorgenti degli eventi di cui Lambda ha bisogno per connettere in modo sicuro il tuo cluster Amazon MSK alla tua funzione.
3. Configurazione delle funzioni e delle autorizzazioni: infine, assicurati che la funzione sia configurata correttamente e disponga delle autorizzazioni necessarie per il suo ruolo di esecuzione.

> **Note**
>
> Ora puoi creare e gestire le mappature delle sorgenti degli eventi Amazon MSK direttamente dalla console Lambda o Amazon MSK. Entrambe le console offrono la possibilità di gestire automaticamente la configurazione delle autorizzazioni necessarie per il ruolo di esecuzione Lambda per un processo di configurazione più semplificato.

Per esempi su come configurare un'integrazione Lambda con un cluster Amazon MSK, consulta the section called “Tutorial” Utilizzo di Amazon MSK come origine di eventi per AWS Lambda sul AWS Compute Blog e Integrazione di Amazon MSK Lambda negli Amazon MSK Labs.

Argomenti

- Esempio di evento
- Configurazione del cluster Amazon MSK e della rete Amazon VPC per Lambda

- Configurazione delle autorizzazioni Lambda per le mappature delle sorgenti degli eventi Amazon MSK
- Configurazione delle origini eventi di Amazon MSK per Lambda
- Tutorial: Utilizzo di uno strumento di mappatura dell'origine degli eventi Amazon MSK per richiamare una funzione Lambda

Esempio di evento

Lambda invia il batch di messaggi nel parametro evento quando richiama la funzione. Il payload evento contiene un array di messaggi. Ogni elemento dell'array contiene i dettagli dell'argomento e dell'identificatore dello shard Amazon MSK, insieme a una data/ora e a un messaggio con codifica 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
                ]
            }
        ]
    }
  ]
}
}
```

Configurazione del cluster Amazon MSK e della rete Amazon VPC per Lambda

Per connettere la AWS Lambda funzione al cluster Amazon MSK, devi configurare correttamente il cluster e l'Amazon Virtual Private Cloud (VPC) in cui si trova. In questa pagina viene descritto come configurare cluster e VPC. Se il cluster e il VPC sono già configurati correttamente, consulta the section called “Configurare un'origine eventi” Configurare la mappatura delle sorgenti degli eventi.

Argomenti

- Panoramica dei requisiti di configurazione di rete per le integrazioni Lambda e MSK
- Configurazione di un gateway NAT per uno strumento di origine eventi MSK
- Configurazione degli AWS PrivateLink endpoint per un'origine di eventi MSK

Panoramica dei requisiti di configurazione di rete per le integrazioni Lambda e MSK

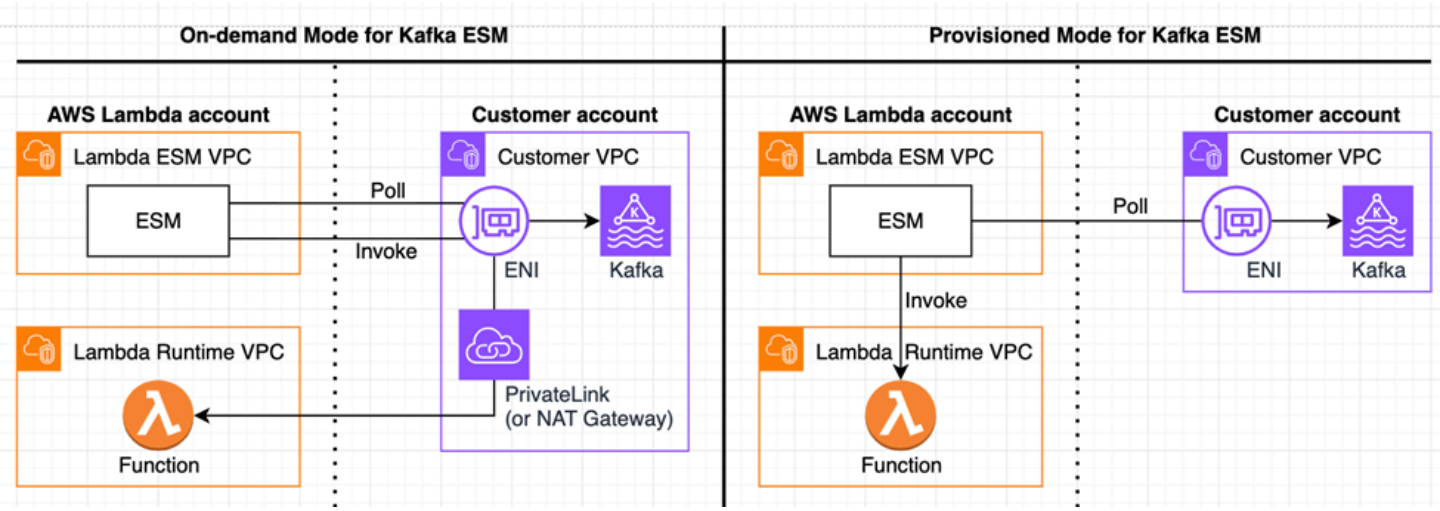
La configurazione di rete richiesta per un'integrazione Lambda e MSK dipende dall'architettura di rete dell'applicazione. Ci sono tre risorse principali coinvolte in questa integrazione: il cluster Amazon MSK, la funzione Lambda e la mappatura delle sorgenti degli eventi Lambda. Ognuna di queste risorse risiede in un VPC diverso:

- Il tuo cluster Amazon MSK risiede in genere in una sottorete privata di un VPC che gestisci.
- La tua funzione Lambda risiede in un AWS VPC gestito di proprietà di Lambda.
- La mappatura delle sorgenti degli eventi Lambda risiede in un altro AWS VPC gestito di proprietà di Lambda, separato dal VPC che contiene la funzione.

La mappatura dell'origine degli eventi è la risorsa intermedia tra il cluster MSK e la funzione Lambda. Il mapping dell'origine eventi ha due processi principali. Innanzitutto, analizza la presenza di nuovi

messaggi nel cluster MSK. Quindi richiama la funzione Lambda con questi messaggi. Poiché queste tre risorse si trovano in VPC diversi, sia le operazioni di poll che di invoke richiedono chiamate di rete tra VPC.

I requisiti di configurazione di rete per la mappatura delle sorgenti degli eventi dipendono dal fatto che utilizzi la modalità previsionata o la modalità su richiesta, come illustrato nel diagramma seguente:



Il modo in cui la mappatura delle sorgenti degli eventi Lambda analizza il cluster MSK alla ricerca di nuovi messaggi è lo stesso in entrambe le modalità. Per stabilire una connessione tra la mappatura delle sorgenti degli eventi e il cluster MSK, Lambda crea un ENI iperpiano (o ne riutilizza uno esistente, se disponibile) nella sottorete privata per stabilire una connessione sicura. Come illustrato nel diagramma, questo iperpiano ENI utilizza la configurazione della sottorete e del gruppo di sicurezza del cluster MSK, non la funzione Lambda.

Dopo aver analizzato il messaggio dal cluster, il modo in cui Lambda richiama la funzione è diverso in ciascuna modalità:

- In modalità provisioned, Lambda gestisce automaticamente la connessione tra il VPC di mappatura della sorgente degli eventi e la funzione VPC. Quindi, non hai bisogno di componenti di rete aggiuntivi per richiamare correttamente la funzione.
- In modalità on-demand, la mappatura delle sorgenti degli eventi Lambda richiama la funzione tramite un percorso attraverso il VPC gestito dal cliente. Per questo motivo, è necessario configurare un gateway NAT nella sottorete pubblica del VPC o gli AWS PrivateLinkendpoint nella sottorete privata del VPC che forniscono l'accesso a Lambda (STS) e, facoltativamente,. AWS Security Token ServiceGestione dei segreti AWS La configurazione corretta di una di queste opzioni consente una connessione tra il VPC e il VPC di runtime gestito da Lambda, necessario per richiamare la funzione.

Un gateway NAT consente alle risorse nella sottorete privata di accedere alla rete Internet pubblica. L'utilizzo di questa configurazione significa che il traffico attraversa Internet prima di richiamare la funzione Lambda. AWS PrivateLinkgli endpoint consentono alle sottoreti private di connettersi in modo sicuro ai AWS servizi o ad altre risorse VPC private senza attraversare la rete Internet pubblica. Vedi the section called “Configurazione di un gateway NAT per uno strumento di origine eventi MSK” o the section called “Configurazione degli AWS PrivateLink endpoint per un'origine di eventi MSK” per i dettagli su come configurare queste risorse.

Finora, abbiamo dato per scontato che il cluster MSK risieda in una sottorete privata all'interno del VPC, che è il caso più comune. Tuttavia, anche se il cluster MSK si trova in una sottorete pubblica all'interno del VPC, è necessario configurare gli AWS PrivateLink endpoint per abilitare una connessione sicura. La tabella seguente riassume i requisiti di configurazione della rete in base a come si configura il cluster MSK e la mappatura delle sorgenti degli eventi Lambda:

Ubicazione del cluster MSK (in VPC gestito dal cliente)	Modalità di scalazione dello strumento di mappatura dell'origine di eventi Lambda	Configurazione di rete richiesta
Sottorete privata	Modalità on demand	Gateway NAT (nella sottorete pubblica del VPC) o AWS PrivateLink endpoint (nella sottorete privata del VPC) per consentire l'accesso a Lambda e, facoltativamente, a Secrets Manager. AWS STS
Sottorete pubblica	Modalità on demand	AWS PrivateLinkendpoint (nella sottorete pubblica del tuo VPC) per abilitare l'accesso a Lambda e, facoltativamenteAWS STS, a Secrets Manager.
Sottorete privata	Modalità assegnata	Nessuno
Sottorete pubblica	Modalità assegnata	Nessuno

Inoltre, i gruppi di sicurezza associati al cluster MSK devono consentire il traffico sulle porte corrette. Assicurati di aver configurato le seguenti regole del gruppo di sicurezza:

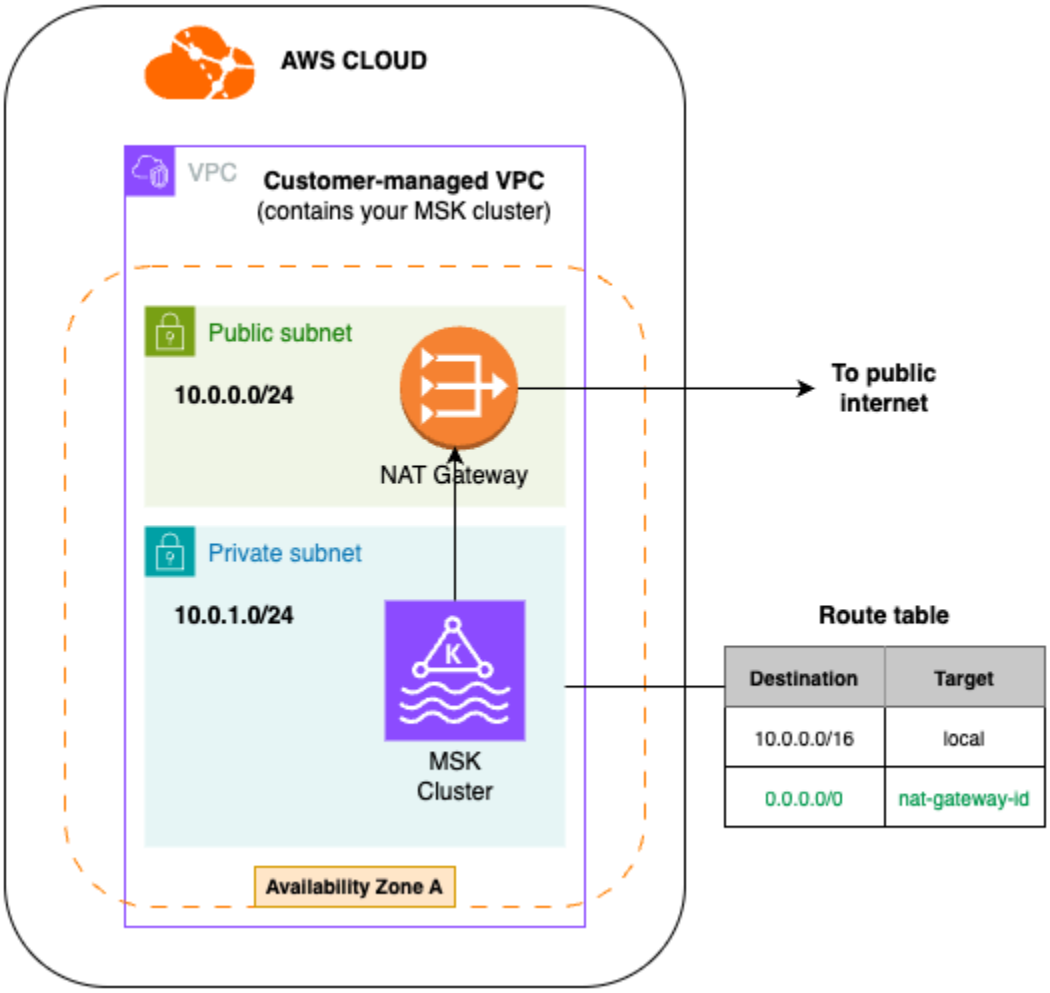
- Regole in ingresso: consenti tutto il traffico sulla porta del broker predefinita. La porta utilizzata da MSK dipende dal tipo di autenticazione sul cluster: `9098` per l'autenticazione IAM, per SASL/SCRAM e `9096` per TLS. `9094` In alternativa, è possibile utilizzare una regola del gruppo di sicurezza autoreferenziante per consentire l'accesso da istanze all'interno dello stesso gruppo di sicurezza.
- Regole in uscita: consentono tutto il traffico sulla porta `443` per destinazioni esterne se la funzione deve comunicare con altri servizi. AWS In alternativa, puoi utilizzare una regola del gruppo di sicurezza autoreferenziale per limitare l'accesso al broker se non hai bisogno di comunicare con altri servizi. AWS
- Regole di ingresso degli endpoint Amazon VPC: se utilizzi un endpoint Amazon VPC, il gruppo di sicurezza associato all'endpoint Amazon VPC deve consentire il traffico in entrata sulla porta dal gruppo di sicurezza del cluster.

Configurazione di un gateway NAT per uno strumento di origine eventi MSK

Puoi configurare un gateway NAT per consentire alla mappatura della fonte degli eventi di interrogare i messaggi dal cluster e richiamare la funzione tramite un percorso attraverso il tuo VPC. Questo è necessario solo se la mappatura delle sorgenti degli eventi utilizza la modalità on demand e il cluster risiede all'interno di una sottorete privata del VPC. Se il cluster risiede in una sottorete pubblica del VPC o la mappatura delle sorgenti degli eventi utilizza la modalità provisioning, non è necessario configurare un gateway NAT.

Le risorse in una sottorete privata richiedono un gateway NAT per accedere alla rete Internet pubblica. Se hai bisogno di connettività privata a Lambda, vedi the section called “Configurazione degli AWS PrivateLink endpoint per un'origine di eventi MSK” invece.

Dopo aver configurato il gateway NAT, è necessario configurare le tabelle di routing appropriate. Ciò consente al traffico proveniente dalla sottorete privata di indirizzarsi verso la rete Internet pubblica tramite il gateway NAT.



I seguenti passaggi ti guidano nella configurazione di un gateway NAT utilizzando la console. Ripetere questi passaggi secondo necessità per ogni zona di disponibilità (AZ).

Per configurare un gateway NAT e un routing corretto (console)

1. Segui la procedura descritta in Creare un gateway NAT, tenendo presente quanto segue:
   - I gateway NAT devono sempre trovarsi in una sottorete pubblica. Crea gateway NAT con connettività pubblica.
   - Se il cluster MSK viene replicato su più AZ, crea un gateway NAT per AZ. Ad esempio, in ogni AZ, il tuo VPC dovrebbe avere una sottorete privata contenente il cluster e una sottorete pubblica contenente il gateway NAT. Per una configurazione con tre AZ, avrai tre sottoreti private, tre sottoreti pubbliche e tre gateway NAT.
2. Dopo aver creato il gateway NAT, apri la console Amazon VPC e scegli Route tables nel menu a sinistra.

3. Selezionare Create route table (Crea tabella di instradamento).
4. Associa la tabella di routing con il VPC contenente il cluster MSK. Facoltativamente, inserisci un nome per la tabella di routing.
5. Selezionare Create route table (Crea tabella di instradamento).
6. Scegliere il ruolo appena creato.
7. Nella scheda Associazioni sottorete scegli Modifica associazioni sottorete.
   - Associa la tabella di routing alla sottorete privata contenente il cluster MSK.
8. Selezionare Modifica route.
9. Scegli Aggiungi route.
   1. Per Destinazione scegliere `0.0.0.0/0`.
   2. Per Target, scegli NAT gateway.
   3. Nella casella di ricerca scegliere il gateway NAT creato nella fase 1. Questo dovrebbe essere il gateway NAT che si trova nella stessa AZ della sottorete privata che contiene il cluster MSK (la sottorete privata associata a questa tabella di routing nel passaggio 6).
10. Scegli Save changes (Salva modifiche).

Configurazione degli AWS PrivateLink endpoint per un'origine di eventi MSK

Puoi configurare gli AWS PrivateLink endpoint per eseguire il polling dei messaggi dal tuo cluster e richiamare la funzione tramite un percorso attraverso il tuo VPC. Questi endpoint dovrebbero consentire al cluster MSK di accedere a quanto segue:

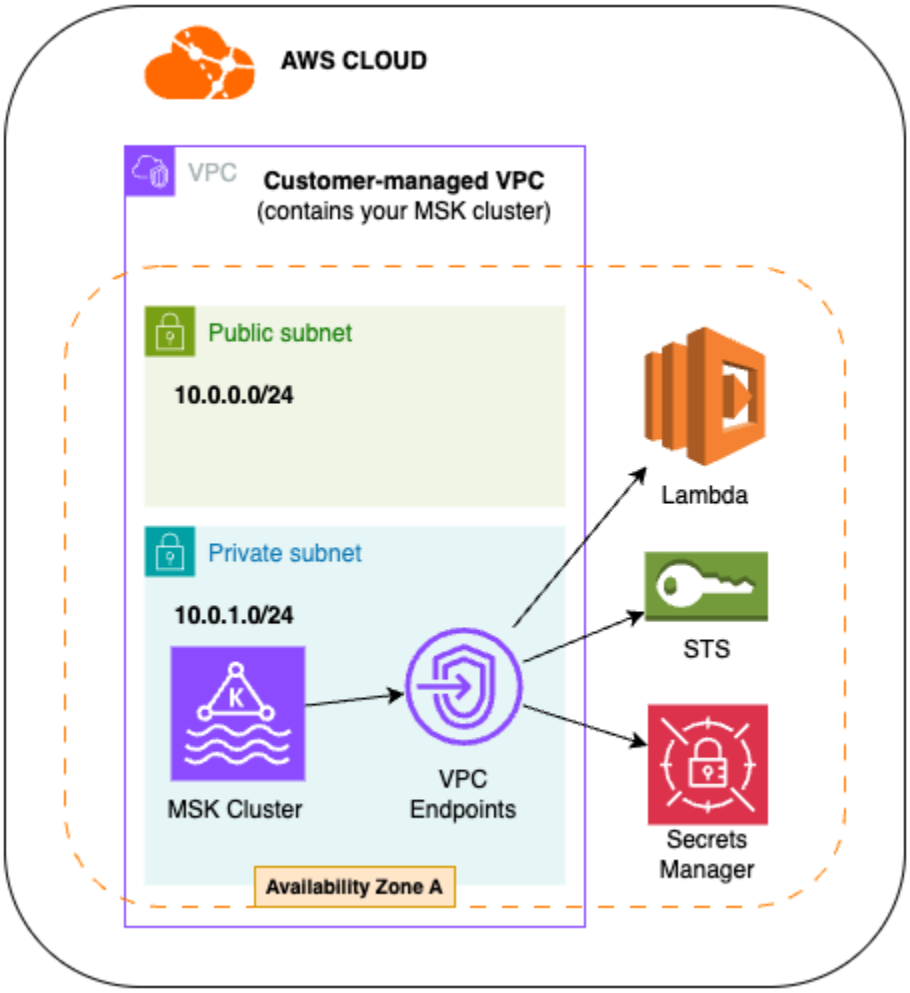
- Il servizio Lambda
- Il AWS Security Token Service(STS)
- Facoltativamente, il Gestione dei segreti AWSservizio. Questo è necessario se il segreto richiesto per l'autenticazione del cluster è archiviato in Secrets Manager.

La configurazione degli endpoint PrivateLink è necessaria solo se la mappatura delle sorgenti degli eventi utilizza la modalità su richiesta. Se la mappatura delle sorgenti degli eventi utilizza la modalità provisioned, Lambda stabilisce automaticamente le connessioni necessarie.

Gli endpoint PrivateLink consentono un accesso sicuro e privato ai servizi su. AWS AWS PrivateLink In alternativa, per configurare un gateway NAT per consentire al cluster MSK di accedere alla rete

Internet pubblica, vedere. [the section called “Configurazione di un gateway NAT per uno strumento di origine eventi MSK”]

Dopo aver configurato gli endpoint VPC, il cluster MSK dovrebbe avere accesso diretto e privato a Lambda, STS e, facoltativamente, a Secrets Manager.



I seguenti passaggi guidano l'utente nella configurazione di un endpoint PrivateLink utilizzando la console. Ripetere questi passaggi se necessario per ogni endpoint (Lambda, STS, Secrets Manager).

Per configurare gli endpoint VPC PrivateLink (console)

1. Apri la console Amazon VPC e seleziona Endpoints nel menu a sinistra.
2. Seleziona Crea endpoint.
3. Facoltativamente, inserisci un nome per il tuo endpoint.
4. Per Tipo, scegli AWS servizi.

5. In Servizi, inizia a digitare il nome del servizio. Ad esempio, per creare un endpoint per la connessione a Lambda, `lambda` digita nella casella di ricerca.
6. Nei risultati, dovresti vedere l'endpoint del servizio nella regione corrente. Ad esempio, nella regione Stati Uniti orientali (Virginia settentrionale), dovresti vedere`com.amazonaws.us-east-2.lambda`. Selezionare Servizi AWS.
7. In Impostazioni di rete, selezionare il VPC che contiene il cluster MSK.
8. In Subnet, seleziona le AZ in cui si trova il cluster MSK.
   - Per ogni AZ, in Subnet ID, scegli la sottorete privata che contiene il cluster MSK.
9. Per Gruppo di sicurezza, seleziona il gruppo di sicurezza associato al cluster database .
10. Seleziona Crea endpoint.

Per impostazione predefinita, gli endpoint Amazon VPC dispongono di policy IAM aperte che consentono un ampio accesso alle risorse. La best practice consiste nel limitare queste policy per eseguire le azioni necessarie utilizzando quell'endpoint. Ad esempio, per l'endpoint Secrets Manager, è possibile modificarne la policy in modo che solo il ruolo di esecuzione della funzione possa accedere al segreto.

Example Policy dell'endpoint VPC: endpoint 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"
        }
    ]
}
```

Per gli endpoint AWS STS e Lambda, puoi limitare il principale chiamante al principale del servizio Lambda. Tuttavia, assicurati di utilizzarlo `"Resource": "*"` in queste politiche.

Example Policy dell'endpoint VPC: endpoint AWS STS

```
{
    "Statement": [
        {
            "Action": "sts:AssumeRole",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Example Policy dell'endpoint VPC: endpoint Lambda

```
{
    "Statement": [
        {
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Configurazione delle autorizzazioni Lambda per le mappature delle sorgenti degli eventi Amazon MSK

Per accedere al cluster Amazon MSK, la mappatura delle funzioni e delle sorgenti degli eventi necessita delle autorizzazioni per eseguire varie azioni dell'API Amazon MSK. Aggiungi queste autorizzazioni al ruolo di esecuzione della funzione. Se gli utenti devono accedere, aggiungi le autorizzazioni richieste alla politica di identità per l'utente o il ruolo.

La policy gestita dai AWSLambdaMSKExecutionruoli contiene le autorizzazioni minime richieste per le mappature delle sorgenti degli eventi Amazon MSK Lambda. Per semplificare il processo di autorizzazione, puoi:

- Allega la politica di gestione dei AWSLambdaMSKExecutionruoli al tuo ruolo di esecuzione.
- Lascia che la console Lambda generi le autorizzazioni per te. Quando crei una mappatura dell'origine degli eventi Amazon MSK nella console, Lambda valuta il tuo ruolo di esecuzione e ti avvisa se mancano delle autorizzazioni. Scegli Genera autorizzazioni per aggiornare automaticamente il tuo ruolo di esecuzione. Questo non funziona se hai creato o modificato manualmente i criteri relativi ai ruoli di esecuzione o se i criteri sono associati a più ruoli. Tieni presente che potrebbero essere ancora necessarie autorizzazioni aggiuntive nel ruolo di esecuzione quando utilizzi funzionalità avanzate come On-Failure Destination o AWS Glue Schema Registry.

Argomenti

- Autorizzazioni richieste
- Autorizzazioni facoltative

Autorizzazioni richieste

Il ruolo di esecuzione della funzione Lambda deve avere le seguenti autorizzazioni richieste per le mappature delle sorgenti degli eventi Amazon MSK. Queste autorizzazioni sono incluse nella politica di gestione dei ruoli. AWSLambda MSKExecution

CloudWatch Registra le autorizzazioni

Le seguenti autorizzazioni consentono a Lambda di creare e archiviare log in CloudWatch Amazon Logs.

- registri: CreateLogGroup
- registri: CreateLogStream
- registri: PutLogEvents

autorizzazioni del cluster MSK

Le seguenti autorizzazioni consentono a Lambda di accedere al tuo cluster Amazon MSK per tuo conto:

- kafka: DescribeCluster
- kafka: V2 DescribeCluster
- kafka: GetBootstrapBrokers

Ti consigliamo di usare kafka: DescribeCluster V2 invece di kafka:. DescribeCluster L'autorizzazione v2 funziona sia con i cluster Amazon MSK con provisioning che senza server. Nella tua policy ti serve solo una di queste autorizzazioni.

Autorizzazioni VPC

Le seguenti autorizzazioni consentono a Lambda di creare e gestire interfacce di rete durante la connessione al cluster Amazon MSK:

- ec2: CreateNetworkInterface
- ec2: DescribeNetworkInterfaces
- ec2: DescribeVpcs
- ec2: DeleteNetworkInterface
- ec2: DescribeSubnets
- ec2: DescribeSecurityGroups

Autorizzazioni facoltative

La funzione Lambda potrebbe richiedere autorizzazioni per:

- Accedi ai cluster Amazon MSK con più account. Per le mappature delle sorgenti degli eventi tra account, è necessario kafka: nel ruolo di esecuzione. DescribeVpcConnection Un preside IAM che crea una mappatura delle sorgenti degli eventi tra account ha bisogno di kafka:. ListVpcConnections
- Accedi al tuo segreto SCRAM, se utilizzi l'autenticazione SASL/SCRAM. Ciò consente alla funzione di utilizzare un nome utente e una password per connettersi a Kafka.
- Descrivi il tuo segreto di Secrets Manager, se utilizzi l' SASL/SCRAM autenticazione MTLS. Ciò consente alla funzione di recuperare le credenziali o i certificati necessari per connessioni sicure.
- Accedi alla tua chiave gestita AWS KMS dal cliente, se il tuo Gestione dei segreti AWS segreto è crittografato con una chiave gestita AWS KMS dal cliente.
- Accedi ai segreti del registro degli schemi, se utilizzi un registro degli schemi con autenticazione:

  - Per il registro AWS Glue degli schemi: le esigenze `glue:GetRegistry` e le `glue:GetSchemaVersion` autorizzazioni della funzione. Questi consentono alla funzione di cercare e utilizzare le regole di formato dei messaggi memorizzate in AWS Glue.
  - Per Confluent Schema Registry con `BASIC_AUTH` o`CLIENT_CERTIFICATE_TLS_AUTH`: La tua funzione richiede l'`secretsmanager:GetSecretValue`autorizzazione per il segreto contenente le credenziali di autenticazione. Ciò consente alla funzione di recuperare i certificati username/password o necessari per accedere al Confluent Schema Registry.
  - Per i certificati CA privati: la tua funzione richiede secretsmanager: GetSecretValue autorizzazione per il segreto contenente il certificato. Ciò consente alla funzione di verificare l'identità dei registri dello schema che utilizzano certificati personalizzati.
- Accedi ai gruppi di consumatori del cluster Kafka e ai messaggi di sondaggio relativi all'argomento, se utilizzi l'autenticazione IAM per la mappatura delle sorgenti degli eventi.

Queste corrispondono alle seguenti autorizzazioni richieste:

- kafka: ListScramSecrets - Consente l'elenco dei segreti SCRAM per l'autenticazione Kafka
- secretsmanager: GetSecretValue - Consente il recupero di segreti da Secrets Manager
- KMS:Decrypt - Permette la decrittografia dei dati crittografati utilizzando AWS KMS
- glue: - Consente l'accesso al registro degli schemi GetRegistry AWS Glue
- glue: GetSchemaVersion - Consente il recupero di versioni specifiche dello schema dal AWS Glue registro degli schemi
- Kafka-Cluster:Connect - Concede il permesso di connettersi e autenticarsi al cluster
- kafka-cluster: AlterGroup - Concede il permesso di unirsi a gruppi su un cluster, equivalente all'ACL READ GROUP di Apache Kafka
- kafka-cluster: DescribeGroup - Concede il permesso di descrivere i gruppi su un cluster, equivalente all'ACL DESCRIBE GROUP di Apache Kafka
- kafka-cluster: DescribeTopic - Concede il permesso di descrivere argomenti su un cluster, equivalente all'ACL DESCRIBE TOPIC di Apache Kafka
- kafka-cluster: ReadData - Concede il permesso di leggere i dati dagli argomenti su un cluster, equivalente all'ACL READ TOPIC di Apache Kafka

Inoltre, se desideri inviare i record delle chiamate non riuscite a una destinazione in caso di errore, avrai bisogno delle seguenti autorizzazioni a seconda del tipo di destinazione:

- Per le destinazioni Amazon SQS: sqs: SendMessage - Consente l'invio di messaggi a una coda Amazon SQS
- Per le destinazioni Amazon SNS: SNS:Publish - Consente la pubblicazione di messaggi su un argomento Amazon SNS
- Per le destinazioni dei bucket Amazon S3: s3: PutObject e s3: ListBucket - Abilita la scrittura e l'elenco di oggetti in un bucket Amazon S3

Per la risoluzione degli errori di autenticazione e autorizzazione, consulta. the section called "Risoluzione dei problemi"

Configurazione delle origini eventi di Amazon MSK per Lambda

Per utilizzare un cluster Amazon MSK come origine di eventi per la tua funzione Lambda, crei una mappatura delle sorgenti di eventi che collega le due risorse. Questa pagina descrive come creare una mappatura delle sorgenti degli eventi per Amazon MSK.

Questa pagina presuppone che tu abbia già configurato correttamente il tuo cluster MSK e l'Amazon Virtual Private Cloud (VPC) in cui risiede. Se è necessario configurare il cluster o il VPC, vedere. the section called "Configurazione del cluster e della rete" Per configurare il comportamento dei tentativi di ripetizione per la gestione degli errori, consulta. the section called "Riprova le configurazioni"

Argomenti

- Utilizzo di un cluster Amazon MSK come origine eventi
- Configurazione dei metodi di autenticazione del cluster Amazon MSK in Lambda
- Creazione di una mappatura della sorgente di eventi Lambda per un'origine di eventi Amazon MSK
- Creazione di mappature delle sorgenti degli eventi tra account in Lambda
- Tutti i parametri di configurazione dell'origine degli eventi Amazon MSK in Lambda

Utilizzo di un cluster Amazon MSK come origine eventi

Quando aggiungi il cluster Apache Kafka o Amazon MSK come trigger per la funzione Lambda, il cluster viene utilizzato come origine eventi.

Lambda legge i dati degli eventi dagli argomenti di Kafka specificati `Topics` in una CreateEventSourceMappingrichiesta, in base alla posizione iniziale specificata. Dopo che l'elaborazione è avvenuta con successo, l'argomento Kafka viene salvato nel cluster Kafka.

Lambda legge i messaggi in sequenza per ogni partizione dell'argomento Kafka. Un singolo payload Lambda può contenere messaggi provenienti da più partizioni. Quando sono disponibili più record, Lambda continua a elaborare i record in batch, in base al BatchSize valore specificato in una CreateEventSourceMappingrichiesta, finché la funzione non raggiunge l'argomento.

Dopo che Lambda ha elaborato ogni batch, esegue il commit degli offset dei messaggi in quel batch. Se la funzione restituisce un errore per uno qualsiasi dei messaggi di un batch, Lambda ritenta l'intero batch di messaggi fino a quando l'elaborazione non riesce o i messaggi scadono. È possibile inviare i record per i quali tutti i nuovi tentativi falliscono a una destinazione in errore per un'elaborazione successiva.

> **Note**
>
> Anche se le funzioni Lambda generalmente prevedono un timeout massimo di 15 minuti, gli strumenti di mappatura dell'origine degli eventi per Amazon MSK, Apache Kafka autogestito, Amazon DocumentDB e Amazon MQ per ActiveMQ e RabbitMQ supportano solo funzioni con timeout massimi di 14 minuti.

Configurazione dei metodi di autenticazione del cluster Amazon MSK in Lambda

Lambda necessita dell'autorizzazione per accedere al cluster Amazon MSK, recuperare record ed eseguire altre attività. Amazon MSK supporta diversi metodi per l'autenticazione con il cluster MSK.

Metodi di autenticazione del cluster

- Accesso non autenticato
- Autenticazione SASL/SCRAM
- Autenticazione TLS reciproca
- Autenticazione IAM
- Come Lambda sceglie un broker bootstrap

Accesso non autenticato

Se nessun client accede al cluster tramite Internet, è possibile utilizzare l'accesso non autenticato.

Autenticazione SASL/SCRAM

Lambda supporta l'autenticazione SASL/SCRAM (Simple Authentication and Security Layer/Salted Challenge Response Authentication Mechanism), con la funzione hash SHA-512 e la crittografia Transport Layer Security (TLS). Affinché Lambda si connetta al cluster, memorizza le credenziali di autenticazione (nome utente e password) in un segreto di Secrets Manager e fai riferimento a questo segreto durante la configurazione della mappatura dell'origine degli eventi.

Per ulteriori informazioni sull'uso di Secrets Manager, consulta Autenticazione delle credenziali di accesso con Secrets Manager nella Amazon Managed Streaming for Apache Kafka Developer Guide.

> **Note**
>
> Amazon MSK non supporta SASL/PLAIN l'autenticazione.

Autenticazione TLS reciproca

Mutual TLS (mTLS) fornisce l'autenticazione bidirezionale tra il client e il server. Il client invia un certificato al server affinché il server verifichi il client. Il server invia inoltre un certificato al client per consentire al client di verificare il server.

Per le integrazioni Amazon MSK con Lambda, il cluster MSK funge da server e Lambda funge da client.

- Affinché Lambda verifichi il tuo cluster MSK, configuri un certificato client come segreto in Secrets Manager e fai riferimento a questo certificato nella configurazione di mappatura delle sorgenti degli eventi. Il certificato client deve essere firmato da un'autorità di certificazione (CA) presente nel trust store del server.
- Il cluster MSK invia anche un certificato server a Lambda. Il certificato del server deve essere firmato da un'autorità di certificazione (CA) nel AWS trust store.

Amazon MSK non supporta certificati server autofirmati. Tutti i broker di Amazon MSK utilizzano certificati pubblici firmati da Amazon Trust Services CAs, che Lambda considera affidabili per impostazione predefinita.

Configurazione del segreto mTLS

Il segreto CLIENT_CERTIFICATE_TLS_AUTH richiede un campo certificato e un campo chiave privata. Per una chiave privata crittografata, il segreto richiede una password per chiave privata. Il certificato e la chiave privata devono essere in formato PEM.

> **Note**
>
> Lambda supporta gli algoritmi di crittografia a chiave privata PBES1(ma non PBES2).

Il campo certificato deve contenere un elenco di certificati, a partire dal certificato client, seguito da qualsiasi certificato intermedio, per finire con il certificato root. Ogni certificato deve iniziare su una nuova riga con la struttura seguente:

```
-----BEGIN CERTIFICATE-----
        <certificate contents>
-----END CERTIFICATE-----
```

Secrets Manager supporta segreti fino a 65.536 byte, che è uno spazio sufficiente per lunghe catene di certificati.

La chiave privata deve essere in formato PKCS #8, con la struttura seguente:

```
-----BEGIN PRIVATE KEY-----
         <private key contents>
-----END PRIVATE KEY-----
```

Per una chiave privata crittografata, utilizza la struttura seguente:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
          <private key contents>
-----END ENCRYPTED PRIVATE KEY-----
```

Nell'esempio seguente viene mostrato il contenuto di un segreto per l'autenticazione mTLS utilizzando una chiave privata crittografata. Per una chiave privata crittografata, includi una password per chiave privata nel segreto.

```
{
 "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-----"
}
```

Per ulteriori informazioni su MTL per Amazon MSK e istruzioni su come generare un certificato client, consulta l'autenticazione client Mutual TLS per Amazon MSK nella Amazon Managed Streaming for Apache Kafka Developer Guide.

Autenticazione IAM

Puoi utilizzare AWS Identity and Access Management (IAM) per autenticare l'identità dei client che si connettono al cluster MSK. Con l'autenticazione IAM, Lambda si basa sulle autorizzazioni nel ruolo di esecuzione della funzione per connettersi al cluster, recuperare i record ed eseguire altre azioni necessarie. Per un esempio di policy che contiene le autorizzazioni necessarie, consulta Create authorization policies for the IAM nella Amazon Managed Streaming for Apache Kafka Developer Guide.

Se l'autenticazione IAM è attiva sul tuo cluster MSK e non fornisci un segreto, Lambda utilizza automaticamente l'autenticazione IAM.

Per ulteriori informazioni sull'autenticazione IAM in Amazon MSK, consulta IAM access control.

Come Lambda sceglie un broker bootstrap

Lambda sceglie un broker bootstrap in base ai metodi di autenticazione disponibili nel tuo cluster e se fornisci un segreto per l'autenticazione. Se fornisci un segreto per mTLS o SASL/SCRAM,

Lambda sceglie automaticamente quel metodo di autenticazione. Se non fornisci un segreto, Lambda seleziona il metodo di autenticazione più forte attivo sul tuo cluster. Di seguito è riportato l'ordine di priorità in cui Lambda seleziona un broker, dall'autenticazione più forte a quella più debole:

- mTLS (segreto fornito per mTLS)
- SASL/SCRAM (secret provided for SASL/SCRAM)
- IAM SASL (nessun segreto fornito e autenticazione IAM attiva)
- TLS non autenticato (nessun segreto fornito e autenticazione IAM non attiva)
- Testo semplice (nessun segreto fornito e autenticazione IAM e TLS non autenticato non attivi)

> **Note**
>
> Se Lambda non riesce a connettersi al tipo di broker più sicuro, non proverà a connettersi a un tipo di broker diverso (più debole). Se vuoi che Lambda scelga un tipo di broker più debole, disattiva tutti i metodi di autenticazione più forti sul tuo cluster.

Creazione di una mappatura della sorgente di eventi Lambda per un'origine di eventi Amazon MSK

Per creare una mappatura delle sorgenti degli eventi, puoi utilizzare la console Lambda, la AWS Command Line Interface (CLI) o un SDK.AWS

> **Note**
>
> Quando crei la mappatura dell'origine degli eventi, Lambda crea un ENI iperpiano nella sottorete privata che contiene il cluster MSK, permettendo a Lambda di stabilire una connessione sicura. Questo iperpiano consentito da ENI utilizza la configurazione della sottorete e del gruppo di sicurezza del cluster MSK, non la funzione Lambda.

I seguenti passaggi della console aggiungono un cluster Amazon MSK come trigger per la funzione Lambda. Sotto il cofano, questo crea una risorsa di mappatura delle sorgenti degli eventi.

Per aggiungere un trigger Amazon MSK alla tua funzione Lambda (console)

1. Apri la pagina Funzione della console Lambda.

2. Scegli il nome della funzione Lambda per la quale si desidera aggiungere un trigger Amazon MSK.
3. In Panoramica delle funzioni, scegliere Aggiungi trigger.
4. In Configurazione Trigger, scegli MSK.
5. Per specificare i dettagli del cluster Kafka, procedi come segue:
    1. Per MSK cluster (Cluster MSK) seleziona il cluster.
    2. Per Nome argomento, inserisci il nome dell'argomento Kafka da cui consumare i messaggi.
    3. Per ID del gruppo di consumatori, inserisci l'ID di un gruppo di consumatori Kafka a cui aderire, se applicabile. Per ulteriori informazioni, consulta the section called “Gruppo di consumer”.
6. Per l'autenticazione del cluster, effettua le configurazioni necessarie. Per ulteriori informazioni sull’autenticazione cluster, consulta the section called “Autenticazione cluster”.
    - Attiva Usa autenticazione se desideri che Lambda esegua l'autenticazione con il tuo cluster MSK quando stabilisci una connessione. L'autenticazione è consigliata.
    - Se utilizzi l'autenticazione, per Metodo di autenticazione, scegli il metodo di autenticazione da utilizzare.
    - Se utilizzi l'autenticazione, per la chiave Secrets Manager, scegli la chiave Secrets Manager che contiene le credenziali di autenticazione necessarie per accedere al tuo cluster.
7. In Event poller configuration, effettuate le configurazioni necessarie.
    - Scegli Attiva trigger per abilitare il trigger subito dopo la creazione.
    - Scegli se desideri configurare la modalità provisioning per la mappatura delle sorgenti degli eventi. Per ulteriori informazioni, consulta the section called “Ridimensionamento dei poller degli eventi”.
        - Se configuri la modalità provisioned, inserisci un valore per Minimum event poller, un valore per Maximum event poller e un valore opzionale per PollerGroupName specificare il raggruppamento di più eventi ESMs all'interno dello stesso VPC di origine eventi.
    - Per Posizione iniziale, scegli come vuoi che Lambda inizi a leggere dal tuo stream. Per ulteriori informazioni, consulta the section called “Posizioni di polling e flussi”.
8. In Batching, effettua le configurazioni necessarie. Per ulteriori informazioni sulla suddivisione in batch, vedere. the section called “Comportamento di batching”

1. Per Batch Size (Dimensione batch), immettere il numero massimo di messaggi da recuperare in un singolo batch.
2. Per Batch window, inserisci il numero massimo di secondi che Lambda impiega per raccogliere i record prima di richiamare la funzione.

9. In Filtraggio, effettua le configurazioni necessarie. Per ulteriori informazioni sul filtro, consulta the section called “Filtro eventi”.
   - Per i criteri di filtro, aggiungete le definizioni dei criteri di filtro per determinare se elaborare o meno un evento.
10. In Gestione degli errori, effettuate le configurazioni necessarie. Per ulteriori informazioni sulla gestione degli errori, vederethe section called “Conserva le chiamate non riuscite”.
    - Per Destinazione in caso di errore, specificare l'ARN della destinazione in caso di errore.
11. Per Tag, inserisci i tag da associare a questa mappatura della fonte degli eventi.
12. Per creare il trigger, scegli Aggiungi.

È inoltre possibile creare la mappatura della sorgente dell'evento utilizzando la AWS CLI con create-event-source-mappingil comando. L'esempio seguente crea una mappatura dell'origine degli eventi per mappare la `my-msk-function` funzione Lambda `AWSKafkaTopic` all'argomento, a partire dal `LATEST` messaggio. Questo comando utilizza inoltre l'SourceAccessConfigurationoggetto per indicare a Lambda di utilizzare l'autenticazione SASL/SCRAM durante la connessione al 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"}]'
```

Se il cluster utilizza l'autenticazione MTLS, includi un SourceAccessConfigurationoggetto che specifica `CLIENT_CERTIFICATE_TLS_AUTH` e una chiave ARN di Secrets Manager. Questo è mostrato nel seguente comando:

```
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"}]'
```

Quando il cluster utilizza l'autenticazione IAM, non è necessario un SourceAccessConfigurationoggetto. Questo è mostrato nel seguente comando:

```
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
```

Creazione di mappature delle sorgenti degli eventi tra account in Lambda

È possibile utilizzare la connettività privata multi-VPC per connettere una funzione Lambda a un cluster MSK assegnato in un altro Account AWS. Utilizza la connettività multi-VPC AWS PrivateLink, che mantiene tutto il traffico all'interno della AWS rete.

> **Note**
>
> Non è possibile creare strumenti di mappatura dell'origine degli eventi multi-account per i cluster MSK serverless.

Per creare uno strumento di mappatura dell'origine degli eventi multi-account, è necessario innanzitutto configurare la connettività multi-VPC per il cluster MSK. Quando crei lo strumento di mappatura dell'origine degli eventi, utilizza l'ARN della connessione VPC gestita anziché l'ARN del cluster, come illustrato negli esempi seguenti. L'CreateEventSourceMappingoperazione varia anche a seconda del tipo di autenticazione utilizzato dal cluster MSK.

Example — Creazione di uno strumento di mappatura dell'origine degli eventi multi-account per cluster che utilizzano l'autenticazione IAM

Quando il cluster utilizza l'autenticazione basata sui ruoli IAM, non è necessario un oggetto. SourceAccessConfiguration Esempio:

```
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 — Crea una mappatura delle sorgenti degli eventi tra account per un cluster che utilizza l'autenticazione SASL/SCRAM

Se il cluster utilizza l'autenticazione SASL/SCRAM, è necessario includere un SourceAccessConfigurationoggetto che specifica e un ARN segreto di `SASL_SCRAM_512_AUTH` Secrets Manager.

Esistono due modi per utilizzare i segreti per le mappature delle sorgenti di eventi Amazon MSK tra account diversi con autenticazione: SASL/SCRAM

- Crea un segreto nell'account della funzione Lambda e sincronizzalo con il segreto del cluster. Crea una rotazione per mantenere sincronizzati i due segreti. Questa opzione consente di controllare il segreto dall'account della funzione.
- Utilizza il segreto associato al cluster MSK. Questo segreto deve consentire l'accesso multi-account all'account della funzione Lambda. Per ulteriori informazioni, consulta la pagina Permissions to Gestione dei segreti AWS secrets for users in a different account.

```
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 — Creazione di uno strumento di mappatura dell'origine degli eventi multi-account per cluster che utilizzano l'autenticazione mTLS

Se il cluster utilizza l'autenticazione MTLS, è necessario includere un SourceAccessConfigurationoggetto che specifica `CLIENT_CERTIFICATE_TLS_AUTH` e un ARN

segreto di Secrets Manager. Il segreto può essere archiviato nell'account del cluster o nell'account della funzione 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"}]'
```

Tutti i parametri di configurazione dell'origine degli eventi Amazon MSK in Lambda

Tutti i tipi di sorgenti di eventi Lambda condividono le stesse operazioni CreateEventSourceMappinge quelle dell'UpdateEventSourceMappingAPI. Tuttavia, solo alcuni parametri si applicano ad Amazon MSK, come illustrato nella tabella seguente.

Parametro	Obbligatorio	Predefinito	Note
AmazonManagedKafkaEventSourceConfig	N	Contiene il ConsumerGroupId campo, che per impostazione predefinita è un valore univoco.	Può essere impostato solo su Create
BatchSize	N	100	Massimo: 10.000.
DestinationConfig	N	N/D	the section called “Conserva le chiamate non riuscite”
Abilitato	N	True	
BisectBatchOnFunctionError	N	False	the section called “Riprova le configurazioni”

Parametro	Obbligatorio	Predefinito	Note
FunctionResponseTypes	N	N/D	the section called “Riprova le configurazioni”
MaximumRecordAgeInSeconds	N	-1 (infinito)	the section called “Riprova le configurazioni”
MaximumRetryAttempts	N	-1 (infinito)	the section called “Riprova le configurazioni”
EventSourceArn	Y	N/D	Può essere impostato solo su Create
FilterCriteria	N	N/D	Controllare gli eventi che Lambda invia alla funzione
FunctionName	Y	N/D	
KMSKeyBraccio	N	N/D	the section called “Crittografia dei criteri di filtro”
MaximumBatchingWindowInSeconds	N	500 ms	Comportamento di batching

Parametro	Obbligatorio	Predefinito	Note
ProvisionedPollersConfig	N	`MinimumPollers` : se non specificato, il valore predefinito è 1 `MaximumPollers` : se non specificato, il valore predefinito è 200 `PollerGroupName` : N/A	the section called “Modalità provisioning”
SourceAccessConfigurations	N	Nessuna credenziale	Credenziali di autenticazione SASL/SCRAM o CLIENT_CERTIFICATE_TLS_AUTH (MutualTLS) per la tua origine eventi
StartingPosition	Y	N/D	AT_TIMESTAMP, TRIM_HORIZON o LATEST Può essere impostato solo su Create
StartingPositionTimestamp	N	N/D	Obbligatorio se StartingPosition è impostato su AT_TIMESTAMP

Parametro	Obbligatorio	Predefinito	Note
Tag	N	N/D	the section called “Tag dello strumento di mappatura dell'origine degli eventi”
Argomenti	Y	N/D	Nome argomento Kafka Può essere impostato solo su Create

> **Note**
>
> Quando si specifica una`PollerGroupName`, più unità ESMs all'interno dello stesso Amazon VPC possono condividere la capacità dell'Event Poller Unit (EPU). Puoi utilizzare questa opzione per ottimizzare i costi della modalità Provisioned per i tuoi. ESMs Requisiti per il raggruppamento ESM:
>
> - ESMs deve trovarsi all'interno dello stesso Amazon VPC
> - Massimo 100 ESMs per gruppo di sondaggi
> - Il numero massimo aggregato di sondaggi di tutti ESMs i membri di un gruppo non può superare i 2000
>
> È possibile aggiornare il file `PollerGroupName` per spostare un ESM in un gruppo diverso oppure rimuovere un ESM da un gruppo `PollerGroupName` impostandolo su una stringa vuota («»).

Tutorial: Utilizzo di uno strumento di mappatura dell'origine degli eventi Amazon MSK per richiamare una funzione Lambda

In questo tutorial verranno eseguite le seguenti operazioni:

- Crea una funzione Lambda nello stesso AWS account di un cluster Amazon MSK esistente.

- Configura la rete e l'autenticazione per consentire a Lambda di comunicare con Amazon MSK.
- Configura uno strumento di mappatura dell'origine degli eventi Lambda Amazon MSK, che esegue la funzione Lambda quando gli eventi vengono visualizzati nell'argomento.

Dopo aver completato questi passaggi, quando gli eventi vengono inviati ad Amazon MSK, potrai configurare una funzione Lambda per elaborare tali eventi automaticamente con il tuo codice Lambda personalizzato.

Cosa puoi fare con questa funzionalità?

Soluzione di esempio: utilizza uno strumento di mappatura dell'origine degli eventi MSK per fornire risultati in tempo reale ai tuoi clienti.

Considera lo scenario seguente: la tua azienda ospita un'applicazione Web in cui i clienti possono visualizzare informazioni sugli eventi dal vivo, come le partite sportive. Gli aggiornamenti delle informazioni dal gioco vengono forniti al tuo team tramite un argomento Kafka su Amazon MSK. Vuoi progettare una soluzione che utilizzi gli aggiornamenti dell'argomento MSK per fornire una visione aggiornata dell'evento dal vivo ai clienti all'interno di un'applicazione da te sviluppata. Hai deciso il seguente approccio di progettazione: le tue applicazioni client comunicheranno con un backend serverless ospitato in AWS. I client si connetteranno tramite sessioni websocket utilizzando l'API Amazon WebSocket API Gateway.

In questa soluzione, è necessario un componente che legga gli eventi MSK, esegua una logica personalizzata per preparare tali eventi per il livello dell'applicazione e quindi inoltri tali informazioni all'API Gateway. Puoi implementare questo componente con AWS Lambda, fornendo la tua logica personalizzata in una funzione Lambda, quindi chiamandolo con una mappatura dell'origine degli eventi AWS Lambda Amazon MSK.

Per ulteriori informazioni sull'implementazione di soluzioni utilizzando l'API Amazon WebSocket API Gateway, consulta i tutorial sulle WebSocket API nella documentazione di API Gateway.

Prerequisiti

Un AWS account con le seguenti risorse preconfigurate:

Per soddisfare questi prerequisiti, ti consigliamo di consultare la sezione Guida introduttiva all'uso di Amazon MSK nella documentazione di Amazon MSK.

- UN cluster Amazon MSK. Consulta Creare un cluster Amazon MSK in Guida introduttiva all'uso di Amazon MSK.

- La seguente configurazione:
  - Assicurati che l'autenticazione basata sui ruoli IAM sia abilitata nelle impostazioni di sicurezza del cluster. Ciò aumenta la sicurezza limitando la funzione Lambda al solo accesso alle risorse Amazon MSK necessarie. Questa funzionalità è abilitata per impostazione predefinita per i nuovi cluster Amazon MSK.
  - Assicurati che l'accesso pubblico sia disattivato nelle impostazioni di rete del cluster. Limitare l'accesso a Internet del cluster Amazon MSK migliora la sicurezza limitando il numero di intermediari che gestiscono i dati. Questa funzionalità è abilitata per impostazione predefinita per i nuovi cluster Amazon MSK.
- Un argomento Kafka nel tuo cluster Amazon MSK da utilizzare per questa soluzione. Consulta Creare un argomento in Guida introduttiva all'uso di Amazon MSK.
- Un host di amministrazione Kafka configurato per recuperare informazioni dal tuo cluster Kafka e inviare eventi Kafka al tuo argomento per i test, ad esempio un'istanza Amazon EC2 con la CLI di amministrazione Kafka e la libreria Amazon MSK IAM installate. Consulta Creare una macchina client in Guida introduttiva all'uso di Amazon MSK.

Dopo aver configurato queste risorse, raccogli le seguenti informazioni dal tuo account per confermare che sei pronto a continuare. AWS

- Il nome del cluster Amazon MSK. È possibile trovare queste informazioni nella console Amazon MSK.
- L'UUID del cluster, parte dell'ARN per il tuo cluster Amazon MSK, che puoi trovare nella console Amazon MSK. Segui le procedure in Elencare i cluster nella documentazione di Amazon MSK per trovare queste informazioni.
- I gruppi di sicurezza associati al cluster Amazon MSK. È possibile trovare queste informazioni nella console Amazon MSK. Nei passaggi seguenti, definiscile come tue*`clusterSecurityGroups`*.
- L'ID Amazon VPC contenente il cluster Amazon MSK. Puoi trovare queste informazioni identificando le sottoreti associate al tuo cluster Amazon MSK nella console Amazon MSK, quindi identificando l'Amazon VPC associato alla sottorete nella console Amazon VPC.
- Il nome dell'argomento Kafka usato nella tua soluzione. Puoi trovare queste informazioni chiamando il tuo cluster Amazon MSK con l'`topics` CLI di Kafka dal tuo host di amministrazione Kafka. Per ulteriori informazioni sugli argomenti della CLI, consulta Aggiungere e rimuovere argomenti nella documentazione di Kafka.
- Il nome di un gruppo di consumer per l'argomento Kafka, adatto all'uso con la funzione Lambda. Questo gruppo può essere creato automaticamente da Lambda, quindi non devi crearlo con la

CLI Kafka. Se devi gestire i tuoi gruppi di consumer, per saperne di più sulla CLI dei gruppi di consumer, consulta Gestione dei gruppi di consumer nella documentazione di Kafka.

Le seguenti autorizzazioni nel tuo AWS account:

- Autorizzazione per creare e gestire una funzione Lambda.
- Autorizzazione per creare policy IAM e associarle alla funzione Lambda.
- Autorizzazione a creare endpoint Amazon VPC e modificare la configurazione di rete nell'Amazon VPC che ospita il cluster Amazon MSK.

Installa il AWS Command Line Interface

Se non l'hai ancora installato AWS Command Line Interface, segui i passaggi indicati in Installazione o aggiornamento della versione più recente di AWS CLI per installarlo.

Per eseguire i comandi nel tutorial, sono necessari un terminale a riga di comando o una shell (interprete di comandi). In Linux e macOS, utilizza la shell (interprete di comandi) e il gestore pacchetti preferiti.

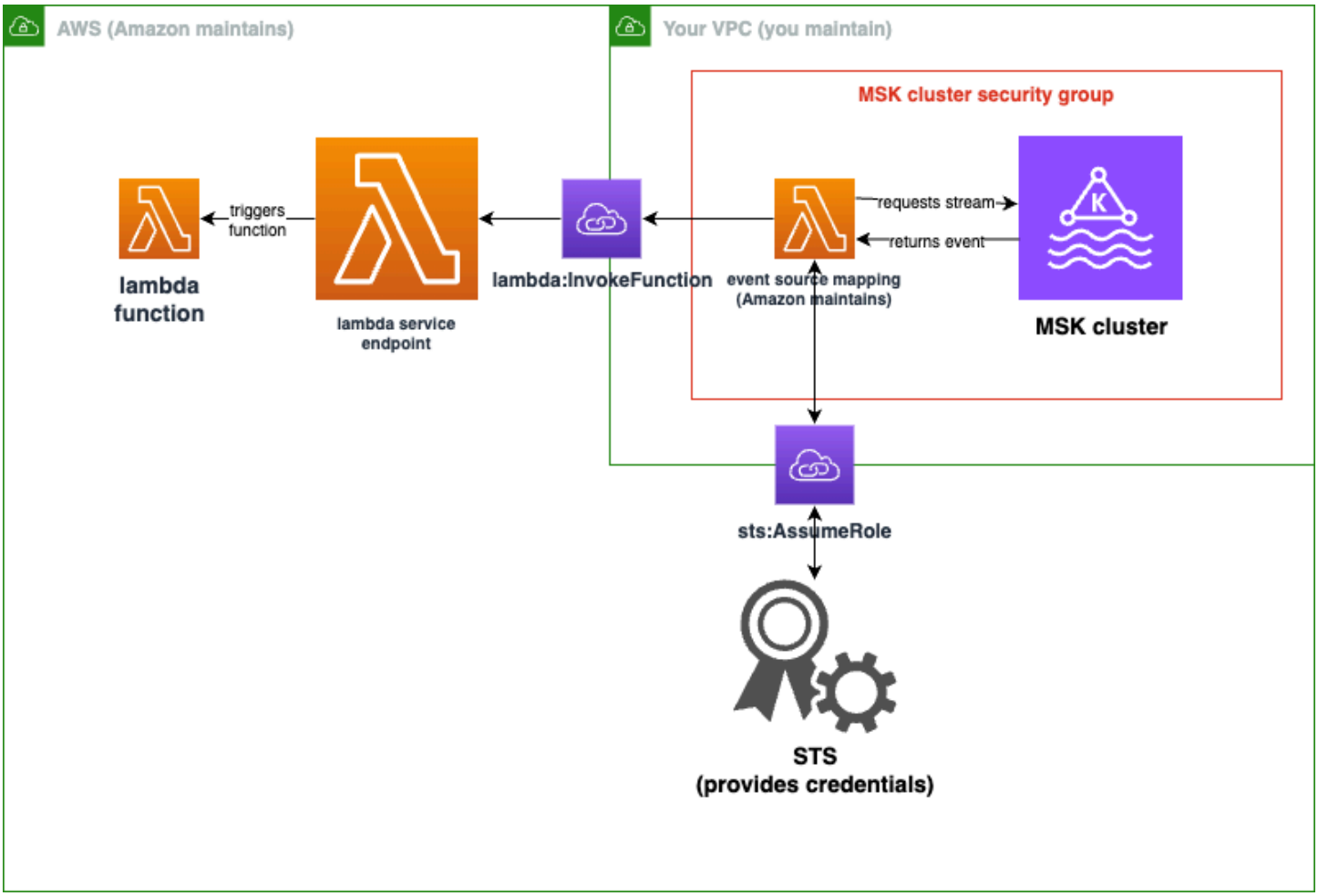
> **Note**
>
> Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere una versione integrata su Windows di Ubuntu e Bash, installa il sottosistema Windows per Linux.

Configurare la connettività di rete per consentire a Lambda di comunicare con Amazon MSK

AWS PrivateLink Usalo per connettere Lambda e Amazon MSK. È possibile farlo creando endpoint Amazon VPC di interfaccia nella console Amazon VPC. Per ulteriori informazioni sui problemi di configurazioni di rete, consulta the section called “Configurazione del cluster e della rete”.

Quando uno strumento di mappatura dell'origine degli eventi Amazon MSK viene eseguita per conto di una funzione Lambda, assume il ruolo di esecuzione della funzione Lambda. Questo ruolo IAM autorizza la mappatura per accedere a risorse protette da IAM, come il cluster Amazon MSK. Sebbene i componenti condividano un ruolo di esecuzione, la mappatura Amazon MSK e la funzione

Lambda hanno requisiti di connettività separati per le rispettive attività, come illustrato nel diagramma seguente.



Lo strumento di mappatura dell'origine degli eventi appartiene al gruppo di sicurezza del cluster Amazon MSK. In questa fase di networking, crea endpoint Amazon VPC dal tuo VPC del cluster Amazon MSK per connettere lo strumento di mappatura dell'origine degli eventi ai servizi Lambda e STS. Proteggi questi endpoint per accettare il traffico proveniente dal tuo gruppo di sicurezza del cluster Amazon MSK. Quindi, modifica i gruppi di sicurezza del cluster Amazon MSK per consentire allo strumento di mappatura dell'origine degli eventi di comunicare con il cluster Amazon MSK.

Puoi configurare la procedura seguente utilizzando la Console di gestione AWS.

Per configurare gli endpoint Amazon VPC di interfaccia per connettere Lambda e Amazon MSK

1. Crea un gruppo di sicurezza per gli endpoint Amazon VPC della tua interfaccia*endpointSecurityGroup*, che consenta il traffico TCP in entrata su 443 da. *clusterSecurityGroups* Segui la procedura in Creare un gruppo di sicurezza nella EC2

documentazione di Amazon per creare un gruppo di sicurezza. Quindi, segui la procedura in Aggiungere regole a un gruppo di sicurezza nella EC2 documentazione di Amazon per aggiungere le regole appropriate.

Crea un gruppo di sicurezza con le seguenti informazioni:

Quando aggiungi le regole in entrata, crea una regola per ogni gruppo di sicurezza in*clusterSecurityGroups*. Per ogni regola:

- Per Tipo, seleziona HTTPS.
- Per Origine, seleziona uno dei. *clusterSecurityGroups*

2. Crea un endpoint che connette il servizio Lambda all'Amazon VPC contenente il cluster Amazon MSK. Segui la procedura riportata in Creare un endpoint di interfaccia.

   Crea un endpoint di interfaccia con le seguenti informazioni:

   - Per Nome servizio, seleziona`com.amazonaws.`*regionName*`.lambda`, dove *regionName* ospita la funzione Lambda.
   - Per VPC, seleziona l'Amazon VPC contenente il cluster Amazon MSK.
   - Per i gruppi di sicurezza, seleziona*endpointSecurityGroup*, che hai creato in precedenza.
   - Per Sottoreti, seleziona le sottoreti che ospitano il tuo cluster Amazon MSK.
   - Per Policy, fornisci il seguente documento di policy, che protegge l'endpoint per l'utilizzo da parte del responsabile del servizio Lambda per l'azione `lambda:InvokeFunction`.

     ```
     {
         "Statement": [
             {
                 "Action": "lambda:InvokeFunction",
                 "Effect": "Allow",
                 "Principal": {
                     "Service": [
                         "lambda.amazonaws.com"
                     ]
                 },
                 "Resource": "*"
             }
         ]
     }
     ```

   - Assicurati Abilita nome DNS rimanga impostato.

3. Crea un endpoint che collega il AWS STS servizio all'Amazon VPC contenente il tuo cluster Amazon MSK. Segui la procedura riportata in Creare un endpoint di interfaccia.

   Crea un endpoint di interfaccia con le seguenti informazioni:

   - Per Nome del servizio, seleziona. AWS STS
   - Per VPC, seleziona l'Amazon VPC contenente il cluster Amazon MSK.
   - Per i gruppi di sicurezza, selezionare*endpointSecurityGroup*.
   - Per Sottoreti, seleziona le sottoreti che ospitano il tuo cluster Amazon MSK.
   - Per Policy, fornisci il seguente documento di policy, che protegge l'endpoint per l'utilizzo da parte del responsabile del servizio Lambda per l'azione `sts:AssumeRole`.

   ```
   {
       "Statement": [
           {
               "Action": "sts:AssumeRole",
               "Effect": "Allow",
               "Principal": {
                   "Service": [
                       "lambda.amazonaws.com"
                   ]
               },
               "Resource": "*"
           }
       ]
   }
   ```

   - Assicurati Abilita nome DNS rimanga impostato.

4. Per ogni gruppo di sicurezza associato al tuo cluster Amazon MSK, ovvero in*clusterSecurityGroups*, consenti quanto segue:

   - Consenti tutto il traffico TCP in entrata e in uscita su 9098 verso tutti, anche all'interno di *clusterSecurityGroups* se stesso.
   - Consenti tutto il traffico TCP in uscita su 443.

   Parte di questo traffico è consentito dalle regole predefinite del gruppo di sicurezza, quindi se il cluster è collegato a un singolo gruppo di sicurezza e tale gruppo ha regole predefinite, non sono necessarie regole aggiuntive. Per modificare le regole del gruppo di sicurezza, segui le procedure in Aggiungere regole a un gruppo di sicurezza nella EC2 documentazione di Amazon.

Aggiungi le regole ai tuoi gruppi di sicurezza con le seguenti informazioni:

- Per ogni regola in entrata o in uscita per la porta 9098, fornisci
  - Per Type (Tipo) seleziona Custom TCP (TCP personalizzato).
  - Per Intervallo di porte, specifica 9098.
  - Per Source, fornisci uno dei*clusterSecurityGroups*.
- Per ogni regola in entrata per la porta 443, per Tipo, seleziona HTTPS.

Crea un ruolo IAM perché Lambda legga dal tuo argomento Amazon MSK

Identifica i requisiti di autenticazione perché Lambda legga dall'argomento di Amazon MSK, quindi definiscili in una policy. Crea un ruolo che autorizzi *lambdaAuthRole* Lambda a utilizzare tali autorizzazioni. Autorizza le azioni sul tuo cluster Amazon MSK utilizzando azioni `kafka-cluster` IAM. Quindi, autorizza Lambda a eseguire le azioni di Amazon `kafka` MSK e EC2 Amazon necessarie per scoprire e connettersi al tuo cluster Amazon MSK, CloudWatch nonché le azioni in modo che Lambda possa registrare ciò che ha fatto.

Per descrivere i requisiti di autenticazione perché Lambda legga da Amazon MSK

1. Scrivi un documento di policy IAM (un documento JSON)*clusterAuthPolicy*, che consenta a Lambda di leggere il tuo argomento Kafka nel tuo cluster Amazon MSK utilizzando il tuo gruppo di consumatori Kafka. Lambda richiede che durante la lettura sia impostato un gruppo di consumer Kafka.

   Modifica il seguente modello per allinearlo ai tuoi prerequisiti:

   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"
            ]
        }
    ]
}
```

Per ulteriori informazioni, consulta the section called “Configurazione delle autorizzazioni ”. Quando scrivi la tua policy:

- Sostituisci *us-east-1* e *111122223333* con Regione AWS e Account AWS del tuo cluster Amazon MSK.
- Per*mskClusterName*, fornisci il nome del tuo cluster Amazon MSK.
- Per*cluster-uuid*, fornisci l'UUID nell'ARN per il tuo cluster Amazon MSK.
- Per*mskTopicName*, fornisci il nome del tuo argomento su Kafka.
- Per*mskGroupName*, fornisci il nome del tuo gruppo di consumatori Kafka.

2. Identifica Amazon MSK, Amazon EC2 e CloudWatch le autorizzazioni necessarie a Lambda per rilevare e connettere il tuo cluster Amazon MSK e registrare tali eventi.

   La policy gestita da `AWSLambdaMSKExecutionRole` definisce in modo permissivo le autorizzazioni richieste. Utilizzalo nelle fasi seguenti.

   In un ambiente di produzione, valuta `AWSLambdaMSKExecutionRole` per limitare la policy del ruolo di esecuzione in base al principio del privilegio minimo, quindi scrivi una policy per il tuo ruolo che sostituisca questa policy gestita.

Per i dettagli sul linguaggio della policy IAM, consulta la documentazione IAM.

Ora che hai scritto il tuo documento di policy, crea una policy IAM in modo da poterla collegare al tuo ruolo. È possibile effettuare tale operazione mediante la console utilizzando la seguente procedura.

Per creare una policy IAM dal documento della policy

1. Accedi Console di gestione AWS e apri la console IAM all'indirizzo. https://console.aws.amazon.com/iam/
2. Nel riquadro di navigazione a sinistra, seleziona Policies (Policy).
3. Scegli Crea policy.
4. Nella sezione Editor di policy, scegli l'opzione JSON.
5. Incolla*clusterAuthPolicy*.
6. Una volta terminata l'aggiunta delle autorizzazioni alla policy, scegli Successivo.
7. Nella pagina Verifica e crea, digita i valori per Nome policy e Descrizione (facoltativa) per la policy che si sta creando. Rivedi Autorizzazioni definite in questa policy per visualizzare le autorizzazioni concesse dalla policy.
8. Seleziona Crea policy per salvare la nuova policy.

Per ulteriori informazioni, consulta Creazione di policy IAM nella documentazione di IAM.

Ora che disponi delle policy IAM appropriate, crea un ruolo e associale ad esso. È possibile effettuare tale operazione mediante la console utilizzando la seguente procedura.

Per creare un ruolo di esecuzione nella console IAM

1. Aprire la pagina Roles (Ruoli) nella console IAM.
2. Scegli Crea ruolo.
3. In Tipo di entità attendibile, scegli Servizio AWS .
4. In Use case (Caso d'uso), scegli Lambda.
5. Scegli Next (Successivo).
6. Selezionare le seguenti policy:
   - *clusterAuthPolicy*
   - `AWSLambdaMSKExecutionRole`
7. Scegli Next (Successivo).
8. Per Nome ruolo, inserisci *lambdaAuthRole* e quindi scegli Crea ruolo.

Per ulteriori informazioni, consulta the section called “Ruolo di esecuzione (autorizzazioni per le funzioni per accedere ad altre risorse)”.

Creare una funzione Lambda per leggere dal tuo argomento Amazon MSK

Crea una funzione Lambda configurata per utilizzare il tuo ruolo IAM. È possibile creare la funzione Lambda utilizzando la console.

Per creare una funzione Lambda utilizzando la tua configurazione di autenticazione

1. Apri la console Lambda e seleziona Crea funzione dall'intestazione.
2. Scegli Crea da zero.
3. Per Nome della funzione, fornisci un nome appropriato a tua scelta.
4. Per Runtime, scegli l'ultima versione supportata di `Node.js` per utilizzare il codice fornito in questo tutorial.
5. Scegli Cambia ruolo di esecuzione predefinito.
6. Seleziona Utilizza un ruolo esistente.
7. Per Ruolo esistente, seleziona *`lambdaAuthRole`*.

In un ambiente di produzione, in genere è necessario aggiungere ulteriori policy al ruolo di esecuzione per la funzione Lambda per elaborare in modo significativo gli eventi Amazon MSK. Per ulteriori informazioni sull'aggiunta di policy al tuo ruolo, consulta Aggiungere o rimuovere le autorizzazioni di identità nella documentazione IAM.

Creare uno strumento di mappatura dell'origine degli eventi sulla funzione Lambda

Lo strumento di mappatura dell'origine degli eventi Amazon MSK fornisce al servizio Lambda le informazioni necessarie per richiamare Lambda quando si verificano gli eventi Amazon MSK appropriati. Puoi creare una mappatura Amazon MSK utilizzando la console. Crea un trigger Lambda, quindi lo strumento di mappatura dell'origine degli eventi viene impostato automaticamente.

Per creare un trigger Lambda (e uno strumento di mappatura dell'origine degli eventi)

1. Vai alla pagina della panoramica della tua funzione Lambda.
2. Nella sezione della panoramica della funzione, scegli Aggiungi trigger in basso a sinistra.
3. Nel menu a discesa Seleziona un'origine, seleziona Amazon MSK.
4. Non impostare l'autenticazione.
5. Per Cluster MSK seleziona il nome del cluster.
6. Per Dimensioni del batch, immetti 1. Questo passaggio semplifica il test di questa funzionalità e non rappresenta un valore ideale nella produzione.

7. Per Nome argomento, fornisci il nome del tuo argomento Kafka.
8. Per l'ID del gruppo di consumer, fornisci l'ID del tuo gruppo di consumer Kafka.

Aggiornare la funzione Lambda per leggere i dati di streaming

Lambda fornisce informazioni sugli eventi di Kafka tramite il parametro event method. Per una struttura di esempio di un evento Amazon MSK, consulta the section called “ Esempio di evento”. Dopo aver capito come interpretare gli eventi Amazon MSK inoltrati da Lambda, puoi modificare il codice della funzione Lambda per utilizzare le informazioni fornite.

Fornisci il seguente codice alla tua funzione Lambda per registrare il contenuto di un evento Lambda Amazon MSK a scopo di test:

.NET

SDK per .NET

> **Note**
>
> C'è altro da fare GitHub. Trova l’esempio completo e scopri come eseguire la configurazione e l’esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento Amazon MSK con JavaScript Lambda utilizzando.

```
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)
      })
    }
}
```

Consumo di un evento Amazon MSK con TypeScript Lambda utilizzando.

```
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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite 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> {

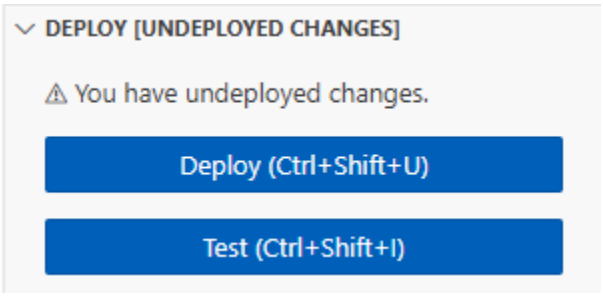
    // required to enable CloudWatch error logging by the runtime
    tracing::init_default_subscriber();
    info!("Setup CW subscriber!");

    run(service_fn(function_handler)).await
}
```

È possibile fornire il codice della funzione a Lambda utilizzando la console.

Aggiornamento del codice della funzione utilizzando l'editor di codice della console

1. Apri la pagina Funzioni della console Lambda e scegli la tua funzione.
2. Scegli la scheda Codice.
3. Nel riquadro Origine del codice, seleziona il tuo file di codice sorgente e modificalo nell'editor di codice integrato.
4. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



Testa la tua funzione Lambda per verificare che sia connessa al tuo argomento Amazon MSK

Ora puoi verificare se la tua Lambda viene richiamata o meno dall'origine dell'evento CloudWatch controllando i registri degli eventi.

Per verificare se la funzione Lambda viene richiamata

1. Usa il tuo host di amministrazione Kafka per generare eventi Kafka utilizzando la CLI `kafka-console-producer`. Per ulteriori informazioni, consulta Scrivere alcuni eventi nell'argomento della documentazione di Kafka. Invia un numero sufficiente di eventi per riempire il batch definito dalla dimensione del batch per lo strumento di mappatura dell'origine degli eventi definito nel passaggio precedente, altrimenti Lambda aspetterà che vengano richiamate ulteriori informazioni.
2. Se la funzione viene eseguita, Lambda scrive cosa è successo a. CloudWatch Nella console, passa alla pagina dei dettagli della funzione Lambda.
3. Selezionare la scheda Configurazione.
4. Dalla barra laterale, seleziona Strumenti di monitoraggio e operazioni.
5. Identifica il gruppo di CloudWatch log in Logging configuration. Il gruppo di log dovrebbe iniziare con `/aws/lambda`. Scegli il link del gruppo di log.
6. Nella CloudWatch console, controlla gli eventi di registro per gli eventi di registro che Lambda ha inviato al flusso di log. Identifica se ci sono eventi del log contenenti il messaggio del tuo evento Kafka, come nell'immagine seguente. In tal caso, hai collegato correttamente una funzione Lambda ad Amazon MSK con uno strumento di mappatura dell'origine degli eventi Lambda.

```
2020-08-06T15:06:18.861-04:00	START RequestId: 88ebae59-be0c-4e22-9db7-4154b437e43a Version: $LATEST
2020-08-06T15:06:18.866-04:00	2020-08-06T19:06:18.866Z 88ebae59-be0c-4e22-9db7-4154b437e43a INFO Key: mytopic-0
2020-08-06T15:06:18.866-04:00	2020-08-06T19:06:18.866Z	88ebae59-be0c-4e22-9db7-4154b437e43a	INFO	Record: {
  topic: 'mytopic',
  partition: 0,
  offset: 38,
  timestamp: 1596740777633,
  timestampType: 'CREATE_TIME',
  value: 'TWVzc2FnZSAjMQ=='
}
2020-08-06T15:06:18.866-04:00	2020-08-06T19:06:18.866Z 88ebae59-be0c-4e22-9db7-4154b437e43a INFO Message: Message #1
2020-08-06T15:06:18.890-04:00	END RequestId: 88ebae59-be0c-4e22-9db7-4154b437e43a
```

Utilizzo di Lambda con Apache Kafka gestito dal cliente

Questo argomento descrive come utilizzare Lambda con un cluster Kafka autogestito. In AWS terminologia, un cluster autogestito include cluster Kafka non AWS ospitati. Ad esempio, puoi ospitare il tuo cluster Kafka con un provider di servizi cloud come Confluent Cloud o Redpanda.

Questo capitolo spiega come utilizzare un cluster Apache Kafka autogestito come origine di eventi per la funzione Lambda. Il processo generale per l'integrazione di Apache Kafka autogestito con Lambda prevede i seguenti passaggi:

1. Configurazione del cluster e della rete: per prima cosa, configura il cluster Apache Kafka autogestito con la configurazione di rete corretta per consentire a Lambda di accedere al cluster.
2. Configurazione della mappatura delle sorgenti degli eventi: quindi, crea la risorsa di mappatura delle sorgenti degli eventi di cui Lambda ha bisogno per connettere in modo sicuro il cluster Apache Kafka alla tua funzione.
3. Configurazione delle funzioni e delle autorizzazioni: infine, assicurati che la funzione sia configurata correttamente e disponga delle autorizzazioni necessarie nel suo ruolo di esecuzione.

Apache Kafka come origine eventi funziona in modo simile all'utilizzo di Amazon Simple Queue Service (Amazon SQS) o Amazon Kinesis. Lambda interroga internamente i nuovi messaggi dell'origine eventi, quindi richiama in modo sincrono la funzione Lambda di destinazione. Lambda legge i messaggi in batch e li fornisce alla funzione come payload di evento. La dimensione massima del batch è configurabile (l'impostazione predefinita è 100 messaggi). Per ulteriori informazioni, consulta the section called “Comportamento di batching”.

Per ottimizzare il throughput dello strumento di mappatura dell'origine degli eventi di Apache Kafla autogestito, configura la modalità provisioning. In modalità provisioning, puoi definire il numero minimo e massimo di poller di eventi assegnati allo strumento di mappatura dell'origine degli eventi. Ciò può migliorare la capacità dello strumento di mappatura dell'origine degli eventi per gestire picchi di messaggi imprevisti. Per ulteriori informazioni, consulta Modalità provisioning.

> **Warning**
>
> Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per

ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center. AWS

Per le origini eventi basate su KAFKA, Lambda supporta i parametri di controllo dell'elaborazione, come le finestre di batch e le dimensioni del batch. Per ulteriori informazioni, consulta Comportamento di batching.

Per un esempio di come utilizzare Kafka autogestito come fonte di eventi, vedi Utilizzo di Apache Kafka ospitato autonomamente come fonte di eventi per sul blog di Compute. AWS Lambda AWS

Argomenti

- Esempio di evento
- Configurazione del cluster e della rete Apache Kafka autogestiti per Lambda
- Configurazione delle autorizzazioni del ruolo di esecuzione Lambda
- Configurazione di origini eventi Apache Kafka autogestito per Lambda

Esempio di evento

Lambda invia il batch di messaggi nel parametro evento quando richiama la funzione Lambda. Il payload evento contiene un array di messaggi. Ogni elemento dell'array contiene i dettagli dell'argomento Kafka e dell'identificatore dello shard Kafka, insieme a una data/ora e a un messaggio con codifica 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
                  ]
               }
            ]
         }
      ]
   }
}
```

Configurazione del cluster e della rete Apache Kafka autogestiti per Lambda

Per connettere la funzione Lambda al cluster Apache Kafka autogestito, devi configurare correttamente il cluster e la rete in cui si trova. Questa pagina spiega come configurare il cluster e la rete. Se il cluster e la rete sono già configurati correttamente, consulta the section called “Configurare un'origine eventi” Configurare la mappatura delle sorgenti degli eventi.

Argomenti

- Un cluster Apache Kafka autogestito.
- Configurare la sicurezza della rete

Un cluster Apache Kafka autogestito.

Puoi ospitare il tuo cluster Apache Kafka autogestito con provider di servizi cloud come Confluent Cloud o Redpanda oppure eseguirlo sulla tua infrastruttura. Assicurati che il cluster sia configurato correttamente e accessibile dalla rete a cui si connetterà la mappatura delle sorgenti degli eventi Lambda.

Configurare la sicurezza della rete

Per concedere a Lambda l'accesso completo ad Apache Kafka autogestito tramite lo strumento di mappatura dell'origine degli eventi, il cluster deve utilizzare un endpoint pubblico (indirizzo IP pubblico) oppure devi fornire l'accesso all'Amazon VPC in cui hai creato il cluster.

Quando usi Apache Kafka autogestito con Lambda, crea endpoint VPC AWS PrivateLink che forniscono alla funzione l'accesso alle risorse del tuo Amazon VPC.

> **Note**
>
> Gli endpoint VPC AWS PrivateLink sono necessari per le funzioni con lo strumento di mappatura dell'origine degli eventi delle origini degli eventi che utilizzano la modalità predefinita (on demand) per i poller degli eventi. Se lo strumento di mappatura dell'origine degli eventi utilizza la modalità provisioning, non è necessario configurare gli endpoint VPC AWS PrivateLink.

Crea un endpoint per fornire l'accesso alle seguenti risorse:

- Lambda: crea un endpoint per il principale del servizio Lambda.
- AWS STS: crea un endpoint per AWS STS in modo che un principale del servizio possa assumere un ruolo per conto tuo.
- Secrets Manager: se il tuo cluster utilizza Secrets Manager per archiviare le credenziali, crea un endpoint per Secrets Manager.

In alternativa, configura un gateway NAT su ogni sottorete pubblica in Amazon VPC. Per ulteriori informazioni, consulta the section called “Accesso Internet per funzioni del VPC”.

Quando crei uno strumento di mappatura dell'origine degli eventi per Apache Kafka autogestito, Lambda verifica se le interfacce di rete elastiche (ENI) sono già presenti per le sottoreti e i gruppi di sicurezza configurati per il tuo Amazon VPC. Se Lambda trova ENI esistenti, prova a riutilizzarle. Altrimenti, Lambda crea nuove ENI per connettersi all'origine dell'evento e richiamare la tua funzione.

> **Note**
>
> Le funzioni Lambda vengono sempre eseguite all'interno di VPC di proprietà del servizio Lambda. La configurazione VPC della funzione non influisce sullo strumento di mappatura

dell'origine degli eventi. Solo la configurazione di rete dell'origine dell'evento determina il modo in cui Lambda si connette all'origine dell'evento.

Configura i gruppi di sicurezza per l'Amazon VPC contenente il tuo cluster. Per impostazione predefinita, Apache Kafka autogestito utilizza le seguenti porte: `9092`.

- Regole in ingresso: consenti tutto il traffico sulla porta del broker predefinita per il gruppo di sicurezza associato all'origine eventi. In alternativa, puoi utilizzare una regola del gruppo di sicurezza autoreferenziante per consentire l'accesso da istanze all'interno dello stesso gruppo di sicurezza.
- Regole in uscita: consenti tutto il traffico sulla porta `443` per destinazioni esterne se la funzione deve comunicare con AWS i servizi. In alternativa, puoi anche utilizzare una regola del gruppo di sicurezza autoreferenziale per limitare l'accesso al broker se non hai bisogno di comunicare con altri servizi. AWS
- Regole di ingresso degli endpoint Amazon VPC: se utilizzi un endpoint Amazon VPC, il gruppo di sicurezza associato all'endpoint Amazon VPC deve consentire il traffico in entrata sulla porta `443` dal gruppo di sicurezza del cluster.

Se il cluster utilizza l'autenticazione, puoi anche limitare la policy degli endpoint per l'endpoint Secrets Manager. Per chiamare l'API Secrets Manager, Lambda utilizza il ruolo della funzione, non il principale del servizio Lambda.

Example Policy dell'endpoint VPC: endpoint 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"
          }
      ]
```

```
  }
```

Quando utilizzi gli endpoint Amazon VPC, AWS indirizza le chiamate API per invocare la funzione utilizzando l'interfaccia di rete elastica (ENI) dell'endpoint. Il principale del servizio Lambda deve chiamare `lambda:InvokeFunction` su tutti i ruoli e le funzioni che utilizzano tali ENI.

Per impostazione predefinita, gli endpoint Amazon VPC dispongono di policy IAM aperte che consentono un ampio accesso alle risorse. La best practice consiste nel limitare queste policy per eseguire le azioni necessarie utilizzando quell'endpoint. Per garantire che lo strumento di mappatura dell'origine degli eventi sia in grado di invocare la funzione Lambda, la policy degli endpoint VPC deve consentire al principale del servizio Lambda di chiamare `sts:AssumeRole` e `lambda:InvokeFunction`. Limitare le policy degli endpoint VPC per consentire solo le chiamate API provenienti dall'organizzazione impedisce il corretto funzionamento dello strumento di mappatura dell'origine degli eventi, pertanto `"Resource": "*"` è richiesto in queste policy.

Il seguente esempio di policy degli endpoint VPC mostra come concedere l'accesso richiesto al principale del servizio Lambda per gli endpoint AWS STS e Lambda.

Example Policy dell'endpoint VPC: endpoint AWS STS

```
{
      "Statement": [
          {
              "Action": "sts:AssumeRole",
              "Effect": "Allow",
              "Principal": {
                  "Service": [
                      "lambda.amazonaws.com"
                  ]
              },
              "Resource": "*"
          }
      ]
    }
```

Example Policy dell'endpoint VPC: endpoint Lambda

```
{
      "Statement": [
          {
              "Action": "lambda:InvokeFunction",
```

```
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Configurazione delle autorizzazioni del ruolo di esecuzione Lambda

Oltre ad accedere al cluster Kafka autogestito, la funzione Lambda necessita delle autorizzazioni per eseguire varie azioni API. Aggiungere queste autorizzazioni al ruolo di esecuzione della funzione. Se i tuoi utenti devono accedere a qualsiasi azione API, aggiungi le autorizzazioni richieste alla politica di identità per l'utente o il ruolo AWS Identity and Access Management (IAM).

Argomenti

- Autorizzazioni necessarie per la funzione Lambda
- Autorizzazioni facoltative per la funzione Lambda
- Aggiunta di autorizzazioni al ruolo di esecuzione
- Concessione di accesso agli utenti con una policy IAM

Autorizzazioni necessarie per la funzione Lambda

Per creare e archiviare i log in un gruppo di log in Amazon CloudWatch Logs, la funzione Lambda deve disporre delle seguenti autorizzazioni nel ruolo di esecuzione:

- registri: CreateLogGroup
- registri: CreateLogStream
- registri: PutLogEvents

Autorizzazioni facoltative per la funzione Lambda

La funzione Lambda potrebbe richiedere autorizzazioni per:

- Descrivere il segreto di Secrets Manager.

- Accedi alla tua chiave AWS Key Management Service (AWS KMS) gestita dal cliente.
- Accedere ad Amazon VPC.
- Invia i record delle chiamate non riuscite a una destinazione.

Secrets Manager e AWS KMS autorizzazioni

A seconda del tipo di controllo degli accessi che stai configurando per i tuoi broker Kafka, la tua funzione Lambda potrebbe richiedere l'autorizzazione per accedere al tuo segreto di Secrets Manager o per decrittografare la tua chiave gestita dal cliente. AWS KMS Per accedere a queste risorse, il ruolo di esecuzione della funzione deve disporre delle seguenti autorizzazioni:

- gestore dei segreti: GetSecretValue
- kms:Decrypt

Autorizzazioni VPC

Se soltanto gli utenti all'interno di un VPC possono accedere al cluster Apache Kafka autogestito, la funzione Lambda deve disporre dell'autorizzazione per accedere alle risorse di Amazon VPC. Queste risorse includono la VPC, le sottoreti, i gruppi di sicurezza e le interfacce di rete. Per accedere a queste risorse, il ruolo di esecuzione della funzione deve disporre delle seguenti autorizzazioni:

- ec2: CreateNetworkInterface
- ec2: DescribeNetworkInterfaces
- ec2: DescribeVpcs
- ec2: DeleteNetworkInterface
- ec2: DescribeSubnets
- ec2: DescribeSecurityGroups

Aggiunta di autorizzazioni al ruolo di esecuzione

Per accedere ad altri AWS servizi utilizzati dal cluster Apache Kafka autogestito, Lambda utilizza le politiche di autorizzazione definite nel ruolo di esecuzione della funzione Lambda.

Per impostazione predefinita, Lambda non è autorizzato a eseguire le operazioni richieste o facoltative per un cluster Apache Kafka autogestito. Dovrai creare e definire queste operazioni in una

policy di attendibilità IAM per il ruolo di esecuzione. Questo esempio mostra come creare una policy che consente a Lambda di accedere alle risorse Amazon VPC.

JSON

```
{
        "Version":"2012-10-17",
        "Statement":[
           {
              "Effect":"Allow",
              "Action":[
                 "ec2:CreateNetworkInterface",
                 "ec2:DescribeNetworkInterfaces",
                 "ec2:DescribeVpcs",
                 "ec2:DeleteNetworkInterface",
                 "ec2:DescribeSubnets",
                 "ec2:DescribeSecurityGroups"
              ],
              "Resource":"*"
           }
        ]
     }
```

Concessione di accesso agli utenti con una policy IAM

Per impostazione predefinita, gli utenti e i ruoli non dispongono dell'autorizzazione per eseguire operazioni API di origine eventi. Per concedere l'accesso agli utenti dell'organizzazione o dell'account, è possibile creare o aggiornare una policy basata sull'identità. Per ulteriori informazioni, consulta Controlling access to AWS resources using policies nella IAM User Guide.

Per la risoluzione degli errori di autenticazione e autorizzazione, consultathe section called “Risoluzione dei problemi”.

Configurazione di origini eventi Apache Kafka autogestito per Lambda

Per utilizzare un cluster Apache Kafka autogestito come origine di eventi per la funzione Lambda, è necessario creare una mappatura delle sorgenti degli eventi che collega le due risorse. Questa pagina descrive come creare una mappatura delle sorgenti degli eventi per Apache Kafka autogestito.

Questa pagina presuppone che tu abbia già configurato correttamente il tuo cluster Kafka e la rete in cui risiede. Se devi configurare il tuo cluster o la tua rete, vedi. the section called “Configurazione del cluster e della rete”

Argomenti

- Utilizzo di un cluster Apache Kafka autogestito come fonte di eventi
- Configurazione dei metodi di autenticazione del cluster in Lambda
- Creazione di una mappatura della sorgente di eventi Lambda per un'origine di eventi Apache Kafka autogestita
- Tutti i parametri di configurazione dell'origine degli eventi Apache Kafka autogestiti in Lambda

Utilizzo di un cluster Apache Kafka autogestito come fonte di eventi

Quando aggiungi il cluster Apache Kafka o Amazon MSK come trigger per la funzione Lambda, il cluster viene utilizzato come origine eventi.

Lambda legge i dati degli eventi dagli argomenti di Kafka specificati `Topics` in una CreateEventSourceMappingrichiesta, in base alla posizione iniziale specificata. Dopo che l'elaborazione è avvenuta con successo, l'argomento Kafka viene salvato nel cluster Kafka.

Lambda legge i messaggi in sequenza per ogni partizione dell'argomento Kafka. Un singolo payload Lambda può contenere messaggi provenienti da più partizioni. Quando sono disponibili più record, Lambda continua a elaborare i record in batch, in base al BatchSize valore specificato in una CreateEventSourceMappingrichiesta, finché la funzione non raggiunge l'argomento.

Dopo che Lambda ha elaborato ogni batch, esegue il commit degli offset dei messaggi in quel batch. Se la funzione restituisce un errore per uno qualsiasi dei messaggi di un batch, Lambda ritenta l'intero batch di messaggi fino a quando l'elaborazione non riesce o i messaggi scadono. È possibile inviare i record per i quali tutti i nuovi tentativi falliscono a una destinazione in errore per un'elaborazione successiva.

> **Note**
>
> Anche se le funzioni Lambda generalmente prevedono un timeout massimo di 15 minuti, gli strumenti di mappatura dell'origine degli eventi per Amazon MSK, Apache Kafka autogestito, Amazon DocumentDB e Amazon MQ per ActiveMQ e RabbitMQ supportano solo funzioni con timeout massimi di 14 minuti.

Configurazione dei metodi di autenticazione del cluster in Lambda

Lambda supporta diversi metodi per l'autenticazione al cluster Apache Kafka autogestito. Assicurarsi di configurare il cluster Kafka in modo da utilizzare uno dei seguenti metodi di autenticazione supportati. Per ulteriori informazioni sulla sicurezza con Kafka, consultare la sezione Sicurezza della documentazione di Kafka.

Autenticazione SASL/SCRAM

Lambda supporta l'autenticazione SASL/SCRAM (Simple Authentication and Security Layer/Salted Challenge Response Authentication Mechanism) con crittografia Transport Layer Security (TLS) (). `SASL_SSL` Lambda invia le credenziali crittografate per l'autenticazione con il cluster. Lambda non supporta il testo in SASL/SCRAM chiaro (). `SASL_PLAINTEXT` Per ulteriori informazioni sull' SASL/SCRAM autenticazione, consulta RFC 5802.

Lambda supporta SASL/PLAIN anche l'autenticazione. Poiché questo meccanismo utilizza credenziali in chiaro, la connessione al server deve utilizzare la crittografia TLS per garantire che le credenziali siano protette.

Per l'autenticazione SASL, è necessario archiviare le credenziali di accesso come segreto in Gestione dei segreti AWS. Per ulteriori informazioni sull'utilizzo di Secrets Manager, consulta Create an Gestione dei segreti AWS secret nella Guida Gestione dei segreti AWS per l'utente.

> **Important**
>
> Per utilizzare Secrets Manager per l'autenticazione, i segreti devono essere archiviati nella stessa AWS area della funzione Lambda.

Autenticazione TLS reciproca

MTLS (Mutual TLS) fornisce l'autenticazione bidirezionale tra client e server. Il client invia un certificato al server affinché il server verifichi il client e il server invia un certificato al client affinché il client verifichi il server.

In Apache Kafka autogestito Lambda agisce come client. È possibile configurare un certificato client (come segreto in Secrets Manager) per autenticare Lambda con i broker Kafka. Il certificato client deve essere firmato da una CA nell'archivio trust del server.

Il cluster Kafka invia un certificato server a Lambda per autenticare i broker con Lambda. Il certificato del server può essere un certificato CA pubblico o un certificato CA/self-signed certificate. The public

CA certificate must be signed by a certificate authority (CA) that's in the Lambda trust store. For a private CA/self firmato privato, è possibile configurare il certificato CA root del server (come segreto in Secrets Manager). Lambda utilizza il certificato root per verificare i broker Kafka.

Per ulteriori informazioni su mTLS, consultare Introduzione dell'autenticazione TLS reciproca per Amazon MSK come origine eventi.

Configurazione del segreto del certificato client

Il segreto CLIENT_CERTIFICATE_TLS_AUTH richiede un campo certificato e un campo chiave privata. Per una chiave privata crittografata, il segreto richiede una password per chiave privata. Il certificato e la chiave privata devono essere in formato PEM.

> **Note**
>
> Lambda supporta gli algoritmi di crittografia a chiave privata PBES1(ma non PBES2).

Il campo certificato deve contenere un elenco di certificati, a partire dal certificato client, seguito da qualsiasi certificato intermedio, per finire con il certificato root. Ogni certificato deve iniziare su una nuova riga con la struttura seguente:

```
-----BEGIN CERTIFICATE-----
            <certificate contents>
-----END CERTIFICATE-----
```

Secrets Manager supporta segreti fino a 65.536 byte, che è uno spazio sufficiente per lunghe catene di certificati.

La chiave privata deve essere in formato PKCS #8, con la struttura seguente:

```
-----BEGIN PRIVATE KEY-----
             <private key contents>
-----END PRIVATE KEY-----
```

Per una chiave privata crittografata, utilizza la struttura seguente:

```
-----BEGIN ENCRYPTED PRIVATE KEY-----
              <private key contents>
-----END ENCRYPTED PRIVATE KEY-----
```

Nell'esempio seguente viene mostrato il contenuto di un segreto per l'autenticazione mTLS utilizzando una chiave privata crittografata. Per una chiave privata crittografata, includere la password per chiave privata nel segreto.

```
{"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-----"
}
```

Configurazione del segreto del certificato CA root del server

Questo segreto viene creato se i broker Kafka utilizzano la crittografia TLS con certificati firmati da una CA privata. Puoi utilizzare la crittografia TLS per l'autenticazione VPC SASL/SCRAM, SASL/PLAIN o MTLS.

Il segreto del certificato CA root del server richiede un campo che contiene il certificato CA root del broker Kafka in formato PEM. Il seguente esempio illustra la struttura del segreto.

```
{"certificate":"-----BEGIN CERTIFICATE-----
MIID7zCCAtegAwIBAgIBADANBgkqhkiG9w0BAQsFADCBmDELMAkGA1UEBhMCVVMx
EDAOBgNVBAgTB0FyaXpvbmExEzARBgNVBAcTClNjb3R0c2RhbGUxJTAjBgNVBAoT
HFN0YXJmaWVsZCBUZWNobm9sb2dpZXMsIEluYy4xOzA5BgNVBAMTMlN0YXJmaWVs
ZCBTZXJ2aWNlcyBSb290IENlcnRpZmljYXRlIEF1dG...
-----END CERTIFICATE-----"
}
```

Creazione di una mappatura della sorgente di eventi Lambda per un'origine di eventi Apache Kafka autogestita

Per creare una mappatura delle sorgenti degli eventi, puoi utilizzare la console Lambda, la AWS Command Line Interface (CLI) o un SDK.AWS

I seguenti passaggi della console aggiungono un cluster Apache Kafka autogestito come trigger per la funzione Lambda. Sotto il cofano, questo crea una risorsa di mappatura delle sorgenti degli eventi.

Prerequisiti

- Un cluster Apache Kafka autogestito. Lambda supporta la versione 0.10.1.0 e successive di Apache Kafka.
- Un ruolo di esecuzione con autorizzazione ad accedere alle AWS risorse utilizzate dal cluster Kafka autogestito.

Aggiunta di un cluster Kafka autogestito (console)

Segui questi passaggi per aggiungere il cluster Apache Kafka autogestito e un argomento Kafka come trigger per la funzione Lambda.

Per aggiungere un trigger Apache Kafka alla funzione Lambda (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere il nome della funzione Lambda.
3. In Panoramica delle funzioni, scegliere Aggiungi trigger.
4. In Configurazione trigger, effettua le operazioni seguenti:
    a. Scegliere il tipo di trigger Apache Kafka.
    b. Per Server di bootstrap, inserisci l'indirizzo composto dalla coppia host e porta di un broker Kafka nel cluster, quindi scegli Aggiungi. Ripeti la procedura per ogni broker Kafka del cluster.
    c. Per Nome argomento, inserisci il nome dell'argomento Kafka utilizzato per memorizzare i record nel cluster.
    d. Se configuri la modalità provisioned, inserisci un valore per Minimum event poller, un valore per Maximum event poller e un valore opzionale per PollerGroupName specificare il raggruppamento di più eventi ESMs all'interno dello stesso VPC di origine eventi.

e. (Facoltativo) In Dimensioni batch, inserisci il numero massimo di record da ricevere in un singolo batch.

f. Per Finestra batch, immetti il tempo massimo in secondi per la raccolta dei registri da parte di Lambda prima di richiamare la funzione.

g. (Facoltativo) Per ID gruppo di consumer, inserisci l'ID di un gruppo di consumer Kafka a cui aderire.

h. (Facoltativo) Per Posizione di inizio, scegli Più recente per iniziare a leggere il flusso dal record più recente, Orizzonte di taglio per iniziare dal primo record disponibile o In corrispondenza del timestamp per specificare un timestamp da cui iniziare la lettura.

i. (Facoltativo) Per VPC, scegli Amazon VPC per il cluster Kafka. Quindi, scegli Sottoreti VPC e Gruppi di sicurezza VPC.

   Questa impostazione è necessaria soltanto se gli utenti del VPC accedono ai broker.

j. (Facoltativo) Per Autenticazione, scegli Aggiungi e quindi esegui le seguenti operazioni:

   i. Scegli il protocollo di accesso o di autenticazione dei broker Kafka del cluster.

      - Se il tuo broker Kafka utilizza l'autenticazione, scegli BASIC_AUTH. SASL/PLAIN
      - Se il tuo broker utilizza SASL/SCRAM l'autenticazione, scegli uno dei protocolli. SASL_SCRAM
      - Se stai configurando l'autenticazione mTLS, scegli il protocollo CLIENT_CERTIFICATE_TLS_AUTH.

   ii. Per SASL/SCRAM l'autenticazione MTLS, scegli la chiave segreta Secrets Manager che contiene le credenziali per il tuo cluster Kafka.

k. (Facoltativo) Per Crittografia, scegli il segreto di Secrets Manager contenente il certificato CA root utilizzato dai broker Kafka per la crittografia TLS, se i broker Kafka utilizzano certificati firmati da una CA privata.

   Questa impostazione si applica alla crittografia TLS per SASL/PLAIN e all' SASL/SCRAM autenticazione MTLS.

l. Per creare il trigger in uno stato disabilitato per il test (scelta consigliata), deselezionare Abilita trigger. Oppure, per attivare immediatamente il trigger, selezionareAbilita trigger.

5. Per creare il trigger, scegli Aggiungi.

Aggiunta di un cluster Kafka autogestito (AWS CLI)

Usa i seguenti AWS CLI comandi di esempio per creare e visualizzare un trigger Apache Kafka autogestito per la tua funzione Lambda.

Utilizzo di SASL/SCRAM

Se gli utenti Kafka accedono ai tuoi broker Kafka tramite Internet, specifica il segreto Secrets Manager che hai creato per l'autenticazione. SASL/SCRAM L'esempio seguente utilizza il create-event-source-mapping AWS CLI comando per mappare una funzione Lambda denominata `my-kafka-function` a un argomento di Kafka denominato. `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"]}}'
```

Utilizzo di un VPC

Se solo gli utenti Kafka all'interno del proprio VPC accedono ai broker Kafka, dovrai specificare VPC, sottorete e gruppo di sicurezza del VPC. L'esempio seguente utilizza il create-event-source-mapping AWS CLI comando per mappare una funzione Lambda denominata `my-kafka-function` a un argomento di Kafka denominato. `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"]}}'
```

Visualizzazione dello stato utilizzando il AWS CLI

L'esempio seguente utilizza il get-event-source-mapping AWS CLI comando per descrivere lo stato della mappatura dell'origine degli eventi creata.

```
aws lambda get-event-source-mapping
            --uuid dh38738e-992b-343a-1077-3478934hjkfd7
```

Tutti i parametri di configurazione dell'origine degli eventi Apache Kafka autogestiti in Lambda

Tutti i tipi di sorgenti di eventi Lambda condividono le stesse operazioni CreateEventSourceMappinge quelle dell'UpdateEventSourceMappingAPI. Tuttavia, solo alcuni parametri si applicano ad Apache Kafka autogestito, come illustrato nella tabella seguente.

Parametro	Obbligatorio	Predefinito	Note
BatchSize	N	100	Massimo: 10.000.
DestinationConfig	N	N/D	the section called “Conserva le chiamate non riuscite”
Abilitato	N	True	
FilterCriteria	N	N/D	Controllare gli eventi che Lambda invia alla funzione
FunctionName	Y	N/D	
KMSKeyArn	N	N/D	the section called “Crittografia dei criteri di filtro”
MaximumBatchingWindowInSeconds	N	500 ms	Comportamento di batching
ProvisionedPollersConfig	N	`MinimumPollers` : se non specificato, il valore predefinito è 1 `MaximumPollers` : se non specificato, il	the section called “Modalità provisioning”

Parametro	Obbligatorio	Predefinito	Note
		valore predefinito è 200 `PollerGroupName` : N/A	
SelfManagedEventSource	Y	N/D	Elenco dei broker Kafka. Può essere impostato solo su Create
SelfManagedKafkaEventSourceConfig	N	Contiene il ConsumerGroupId campo che per impostazione predefinita è un valore univoco.	Può essere impostato solo su Create
SourceAccessConfigurations	N	Nessuna credenziale	Informazioni sul VPC o credenziali di autenticazione per il cluster Per SASL_PLAIN, imposta su BASIC_AUTH
StartingPosition	Y	N/D	AT_TIMESTAMP, TRIM_HORIZON o LATEST Può essere impostato solo su Create

Parametro	Obbligatorio	Predefinito	Note
StartingPositionTimestamp	N	N/D	Obbligatorio se StartingPosition è impostato su AT_TIMESTAMP
Tag	N	N/D	the section called “Tag dello strumento di mappatura dell'origine degli eventi”
Argomenti	Y	N/D	Nome dell'argomento Può essere impostato solo su Create

Note

Quando si specifica una`PollerGroupName`, più unità ESMs all'interno dello stesso Amazon VPC possono condividere la capacità dell'Event Poller Unit (EPU). Puoi utilizzare questa opzione per ottimizzare i costi della modalità Provisioned per i tuoi. ESMs Requisiti per il raggruppamento ESM:

- ESMs deve trovarsi all'interno dello stesso Amazon VPC
- Massimo 100 ESMs per gruppo di sondaggi
- Il numero massimo aggregato di sondaggi di tutti ESMs i membri di un gruppo non può superare i 2000

È possibile aggiornare il file `PollerGroupName` per spostare un ESM in un gruppo diverso oppure rimuovere un ESM da un gruppo `PollerGroupName` impostandolo su una stringa vuota («»).

Modalità di ridimensionamento del poller di eventi di Apache Kafka in Lambda

Puoi scegliere tra due modalità di ridimensionamento dei poller degli eventi per Amazon MSK e mappature delle sorgenti di eventi Apache Kafka autogestite:

- Modalità on-demand (impostazione predefinita)
- Modalità provisioning

Modalità on-demand (impostazione predefinita)

Quando crei inizialmente l'origine degli eventi Kafka, Lambda assegna un numero predefinito di poller di eventi per elaborare tutte le partizioni nell'argomento Kafka. Lambda aumenta o riduce automaticamente il numero di sondaggi di eventi in base al caricamento dei messaggi.

In un intervallo di un minuto, Lambda valuta il ritardo dell'offset di tutte le partizioni dell'argomento. Se il ritardo dell'offset è troppo alto, lo shard sta ricevendo messaggi più velocemente di quanto Lambda possa elaborarli. Se necessario, Lambda aggiunge o rimuove i poller di eventi dall'argomento. Questo processo di dimensionamento automatico di aggiunta o rimozione dei poller degli eventi avviene entro tre minuti dalla valutazione.

Se la funzione Lambda di destinazione è limitata, Lambda riduce il numero di poller di eventi. Questa operazione riduce il carico di lavoro sulla funzione riducendo il numero di messaggi che i poller di eventi possono recuperare e inviare alla funzione.

Modalità provisioning

Per i carichi di lavoro in cui è necessario ottimizzare il throughput dello strumento di mappatura dell'origine degli eventi, è possibile utilizzare la modalità provisioning. In modalità provisioning, vengono definiti i limiti minimi e massimi per la quantità di poller di eventi assegnati. Questi poller di eventi con provisioning sono dedicati allo strumento di mappatura dell'origine degli eventi e possono gestire picchi di messaggi imprevisti tramite un dimensionamento automatico reattivo. Ti consigliamo di utilizzare la modalità provisioning per i carichi di lavoro Kafka che hanno requisiti di prestazioni rigorosi.

In Lambda, un event poller è un'unità di calcolo con funzionalità di throughput che variano in base al tipo di origine dell'evento. Per Amazon MSK e Apache Kafka autogestito, ogni event poller può gestire fino al 5% MB/sec del throughput o fino a 5 chiamate simultanee. Ad esempio, se la fonte

dell'evento produce un payload medio di 1 MB e la durata media della funzione è di 1 secondo, un singolo event poller di eventi Kafka può supportare 5 MB/sec velocità effettiva e 5 chiamate Lambda simultanee (presupponendo che non vi sia alcuna trasformazione del payload). Per Amazon SQS, ogni event poller può gestire fino all'1% MB/sec del throughput o fino a 10 chiamate simultanee. L'utilizzo della modalità provisioned comporta costi aggiuntivi in base all'utilizzo dell'event poller. Per i dettagli sui prezzi, vedere Prezzi di AWS Lambda.

> **Note**
>
> Quando si utilizza la modalità provisioned, non è necessario creare endpoint AWS PrivateLink VPC o concedere le autorizzazioni associate come parte della configurazione di rete.

In modalità provisioning, l'intervallo di valori accettati per il numero minimo di poller di event (`MinimumPollers`) è compreso tra 1 e 200, inclusi. L'intervallo di valori accettati per il numero massimo di poller di eventi (`MaximumPollers`) è compreso tra 1 e 2.000, inclusi. `MaximumPollers` deve essere maggiore o uguale a `MinimumPollers`. Inoltre, per mantenere l'elaborazione ordinata all'interno delle partizioni, Lambda limita a `MaximumPollers` il numero di partizioni indicato nell'argomento.

Per ulteriori informazioni sulla scelta dei valori minimi e massimi appropriati di poller di eventi, consulta Best practice.

Puoi configurare la modalità di provisioning per la mappatura delle sorgenti degli eventi Kafka utilizzando la console o l'API Lambda.

Per configurare la modalità provisioning per una mappatura delle sorgenti di eventi esistente (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliete la funzione con la mappatura delle sorgenti degli eventi per cui desiderate configurare la modalità di provisioning.
3. Scegli la scheda Configurazione, quindi scegli Trigger.
4. Scegli la mappatura delle sorgenti degli eventi per cui desideri configurare la modalità di provisioning, quindi scegli Modifica.
5. In modalità Provisioned, seleziona Configura.

- Per Numero minimo di poller di eventi, inserisci un valore compreso tra 1 e 200. Se non si specifica un valore, Lambda assegna il valore predefinito 1.
- Per Numero massimo di poller di eventi, inserisci un valore compreso tra 1 e 2.000. Questo valore deve essere maggiore o uguale al valore specificato in Numero minimo di poller di eventi. Se non si specifica un valore, Lambda assegna il valore predefinito 200.

6. Scegli Save (Salva).

È possibile configurare la modalità di provisioning a livello di codice utilizzando l'ProvisionedPollerConfigoggetto presente in. EventSourceMappingConfiguration Ad esempio, il seguente comando UpdateEventSourceMappingCLI configura un `MinimumPollers` valore di 5 e un `MaximumPollers` valore di 100.

```
aws lambda update-event-source-mapping \
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
    --provisioned-poller-config '{"MinimumPollers": 5, "MaximumPollers": 100}'
```

Dopo aver configurato la modalità provisioning, puoi osservare l'utilizzo dei poller di eventi per il tuo carico di lavoro monitorando il parametro `ProvisionedPollers`. Per ulteriori informazioni, consulta the section called “Parametri dello strumento di mappatura dell'origine degli eventi”.

Per disabilitare la modalità provisioning e tornare alla modalità predefinita (su richiesta), puoi utilizzare il seguente comando CLI UpdateEventSourceMapping:

```
aws lambda update-event-source-mapping \
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
    --provisioned-poller-config '{}'
```

Funzionalità avanzate di gestione degli errori e prestazioni

Per le mappature delle sorgenti di eventi Kafka con la modalità provisioning abilitata, puoi configurare funzionalità aggiuntive per migliorare la gestione e le prestazioni degli errori:

- Configurazioni di nuovi tentativi: controlla il modo in cui Lambda gestisce i record non riusciti con il numero massimo di tentativi di ripetizione, limiti di età dei record, suddivisione in batch e risposte batch parziali.
- Destinazioni Kafka in caso di errore: invia i record non riusciti a un argomento di Kafka per elaborarli o analizzarli in un secondo momento.

Best practice e considerazioni sull'utilizzo della modalità provisioning

La configurazione ottimale dei poller di eventi minimi e massimi per lo strumento di mappatura dell'origine degli eventi dipende dai requisiti delle prestazioni dell'applicazione. Ti consigliamo di iniziare con i poller di eventi minimi di default per definire il profilo delle prestazioni. Modifica la configurazione in base ai modelli di elaborazione dei messaggi osservati e al profilo delle prestazioni desiderato.

Per carichi di lavoro con picchi di traffico e requisiti delle prestazioni rigorosi, aumenta il numero minimo di poller di eventi per gestire picchi improvvisi di messaggi. Per determinare i poller di eventi minimi richiesti, considerate i messaggi al secondo del carico di lavoro e la dimensione media del payload e utilizzate la capacità di throughput di un singolo event poller (fino a 5) come riferimento. MBps

Per mantenere l'elaborazione ordinata all'interno di uno shard, Lambda limita il numero massimo di poller di eventi al numero di shard nell'argomento. Inoltre, il numero massimo di poller di eventi a cui lo strumento di mappatura dell'origine degli eventi può scalare dipende dalle impostazioni di simultaneità della funzione.

Quando attivi la modalità provisioned, aggiorna le impostazioni di rete per rimuovere gli endpoint AWS PrivateLink VPC e le autorizzazioni associate.

Ottimizzazione dei costi per la modalità Provisioned

Prezzi in modalità provisioned

La modalità Provisioned viene addebitata in base ai poller di eventi minimi forniti e ai poller di eventi utilizzati durante la scalabilità automatica. Gli addebiti vengono calcolati utilizzando un'unità di fatturazione denominata Event Poller Unit (EPU). Si paga in base al numero e alla durata dell' EPUs utilizzo, misurati in. Event-Poller-Unit-hours È possibile utilizzare la modalità Provisioned con un singolo ESM per applicazioni sensibili alle prestazioni oppure è possibile raggrupparne più di uno all'interno dello stesso VPC per condividere ESMs la capacità e i costi dell'EPU. Approfondiremo due funzionalità che ti aiutano a ottimizzare i costi della modalità Provisioned. Per i dettagli sui prezzi, consulta i prezzi di AWS Lambda.

Utilizzo migliorato dell'EPU

Ogni EPU supporta fino a 20 capacità di MB/s throughput per il polling degli eventi e supporta un valore predefinito di 10 poller di eventi. Quando si crea una modalità Provisioned per Kafka ESM

impostando un numero minimo e massimo di poller, viene utilizzato il numero minimo di poller per il provisioning EPUs in base all'impostazione predefinita di 10 poller di eventi per EPU. Tuttavia, ogni event poller può scalare in modo indipendente fino a supportare fino al 5% MB/s della capacità di throughput, il che potrebbe richiedere una minore densità di event poller su una specifica EPU e innescare la scalabilità di. EPUs Il numero di event poller allocati su un'EPU dipende dalla capacità di calcolo consumata da ciascun event poller. Questo approccio di utilizzo migliorato dell'EPU consente ai poller di eventi con requisiti di throughput diversi di utilizzare la capacità dell'EPU in modo efficace, riducendo i costi per tutti. ESMs

Raggruppamento ESM

Per ottimizzare ulteriormente i costi della modalità Provisioned, è possibile raggruppare più sistemi Kafka ESMs per condividere la capacità EPU. Con il raggruppamento ESM e l'utilizzo avanzato dell'EPU, è possibile ridurre i costi della modalità Provisioned fino al 90% per carichi di lavoro a basso throughput rispetto all'esecuzione in modalità ESM singola. Tutto ciò ESMs che richiede meno di 1 capacità EPU trarrà vantaggio dal raggruppamento ESM. Ciò richiede ESMs in genere un numero minimo di event poller per supportare le proprie esigenze di throughput. Questa funzionalità ti consentirà di adottare la modalità Provisioned per tutti i tuoi carichi di lavoro Kafka e di beneficiare di funzionalità come la convalida dello schema, il filtraggio Avro/Protobuf degli eventi, le chiamate a bassa latenza e la gestione avanzata degli errori disponibili solo in modalità Provisioned.

Quando configuri il `PollerGroupName` parametro con lo stesso valore per più utenti ESMs all'interno dello stesso Amazon VPC, questi ESMs condividono le risorse EPU anziché richiedere una capacità EPU dedicata. Puoi raggruppare fino a 100 ESMs per gruppo di poller e il numero massimo di poller aggregato per tutti ESMs i membri di un gruppo non può superare i 2000.

Per configurare il raggruppamento ESM (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere la funzione.
3. Scegli Configurazione, quindi scegli Trigger.
4. Quando crei una nuova mappatura della sorgente degli eventi Kafka o ne modifichi una esistente, seleziona Configura in modalità Provisioned.
5. Per Numero minimo di poller di eventi, inserisci un valore compreso tra 1 e 200.
6. Per Numero massimo di poller di eventi, inserisci un valore compreso tra 1 e 2.000.
7. Per il nome del gruppo Poller, inserisci un identificatore per il gruppo. Usa lo stesso nome per gli altri ESMs che vuoi raggruppare.

8. Scegli Save (Salva).

Per configurare il raggruppamento ESM (CLI AWS)

L'esempio seguente crea un ESM con un gruppo di poller denominato: `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"
  }'
```

Per aggiungere un altro ESM allo stesso gruppo (condividendo la capacità EPU), usa lo stesso: 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**
>
> È possibile aggiornare il file `PollerGroupName` per spostare un ESM in un gruppo diverso o rimuovere un ESM da un gruppo passando una stringa vuota («») per: `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": ""
  }'
```

Considerazioni sulla strategia di raggruppamento

- Limite dell'applicazione: gruppo ESMs che appartiene alle stesse applicazioni o servizi per una migliore allocazione e gestione dei costi. Prendi in considerazione l'utilizzo di convenzioni di denominazione come `app-name-environment` (ad esempio,). `order-processor-prod`
- Schema del traffico: evita i raggruppamenti ESMs con un throughput elevato e un andamento di traffico con picchi di traffico, poiché ciò potrebbe portare a un conflitto di risorse.
- Blast radius: considera l'impatto se l'infrastruttura condivisa presenta problemi. Tutti i ESMs membri dello stesso gruppo sono interessati dalle limitazioni delle risorse condivise. Per i carichi di lavoro mission critical, potresti voler utilizzare gruppi separati o dedicati. ESMs

Esempio di ottimizzazione dei costi

Considerate uno scenario in cui ne avete 10 ESMs, ciascuno configurato con 1 event poller e un throughput inferiore a 2 MB/s:

Senza raggruppamento:

- Ogni ESM richiede la propria EPU
- Totale EPUs necessario: 10
- Costo per EPU: 0,185 USD/ora negli Stati Uniti orientali (Virginia settentrionale)
- Costo EPU mensile (720 ore): 10 × 720 × 0,185 USD = 1.332 USD

Con raggruppamento:

- Tutti e 10 ESMs condividono la capacità dell'EPU
- 10 sondaggi per eventi rientrano in 1 EPU (con nuovi 10 sondaggi per ogni supporto EPU)
- Totale necessario: 1 EPUs
- Costo mensile dell'EPU (720 ore): 1 × 720 × 0,185 USD = 133,20 USD
- Risparmio sui costi: 90% (1.198,80$ di risparmio al mese)

Sondaggi di Apache Kafka e posizioni iniziali dello streaming in Lambda

Il parametro startingPosition indica a Lambda quando iniziare a leggere i messaggi dal tuo stream Amazon MSK o Apache Kafka autogestito. Ci sono tre opzioni tra cui scegliere:

- Più recente: Lambda inizia a leggere subito dopo il record più recente nell'argomento Kafka.
- Orizzonte di taglio: inizia a leggere l'argomento con l'ultimo record non tagliato nella partizione. Questo è il record meno recente nella partizione.
- Al timestamp: Lambda avvia la lettura da una posizione definita da un timestamp, in secondi di tempo Unix. Utilizzate il parametro startingPositionTimestamp per specificare il timestamp.

Tieni presente che il polling dei flussi durante la creazione e gli aggiornamenti dello strumento di mappatura dell'origine degli eventi alla fine è coerente.

- Durante la creazione dello strumento di mappatura dell'origine degli eventi, potrebbero essere necessari alcuni minuti per l'avvio degli eventi di polling dal flusso.
- Durante gli aggiornamenti dello strumento di mappatura dell'origine degli eventi, potrebbero essere necessari alcuni minuti per l'avvio degli eventi di polling dal flusso.

Questo comportamento implica che se specifichi `LATEST` come posizione iniziale del flusso, lo strumento di mappatura dell'origine degli eventi potrebbe perdere eventi durante la creazione o gli aggiornamenti. Per garantire che nessun evento venga perso, specificate o. `TRIM_HORIZON` `AT_TIMESTAMP`

ID gruppo di consumer personalizzabile

Quando configuri Amazon MSK o Apache Kafka autogestita come origine, puoi specificare un ID gruppo di consumer. Questo ID gruppo di consumer è un identificatore esistente per il gruppo di consumer Kafka a cui desideri che la tua funzione Lambda aderisca. Puoi utilizzare questa funzione per migrare senza problemi qualsiasi configurazione di elaborazione dei record Kafka in corso da altri utenti a Lambda.

Kafka distribuisce messaggi a tutti i consumer di un gruppo di consumer. Se si specifica un ID di gruppo di consumatori con altri consumatori attivi, Lambda riceve solo una parte dei messaggi dall'argomento Kafka. Se desideri che Lambda gestisca tutti i messaggi dell'argomento, disattiva tutti gli altri sondaggi in quel gruppo di consumer.

Inoltre, se specifichi un ID gruppo di consumer e Kafka trova un gruppo di consumer esistente valido con lo stesso ID, Lambda ignora il parametro per la mappatura dell'origine eventi. Inizia invece ad elaborare i record in base alla compensazione impegnata del gruppo di consumer. Se specifichi un ID gruppo di consumer e Kafka non riesce a trovare un gruppo di consumer esistente, Lambda configura l'origine eventi con la specificata `StartingPosition`.

L'ID gruppo di consumer deve essere univoco tra tutte le origini eventi Kafka. Dopo aver creato una mappatura dell'origine eventi Kafka con l'ID gruppo di consumer specificato, non sarà più possibile aggiornare questo valore.

Filtraggio eventi da origini eventi Amazon MSK e Apache Kafka autogestito

Puoi utilizzare il filtraggio degli eventi per controllare quali record di un flusso o di una coda Lambda invia alla funzione. Per informazioni generali sul funzionamento del filtraggio eventi, consulta the section called “Filtro eventi”.

> **Note**
>
> Amazon MSK e le mappature origine eventi Apache Kafka autogestito. `value`

Argomenti

- Nozioni di base del filtro evento

Nozioni di base del filtro evento

Supponiamo che un produttore stia scrivendo messaggi su un argomento nel tuo cluster Amazon MSK, in formato JSON valido o come stringhe semplici. Un record di esempio sarebbe simile al seguente, con il messaggio convertito in una stringa codificata Base64 nel campo `value`.

```
{
    "mytopic-0":[
        {
            "topic":"mytopic",
            "partition":0,
            "offset":15,
            "timestamp":1545084650987,
            "timestampType":"CREATE_TIME",
            "value":"SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
            "headers":[]
        }
    ]
}
```

Supponiamo che il produttore Apache Kafka stia scrivendo messaggi sul tuo argomento nel seguente formato JSON.

```
{
    "device_ID": "AB1234",
    "session":{
```

```
        "start_time": "yyyy-mm-ddThh:mm:ss",
        "duration": 162
    }
}
```

Puoi utilizzare la chiave `value` per filtrare i record. Supponiamo di voler filtrare solo i record in cui `device_ID` inizia con le lettere AB. L'oggetto `FilterCriteria` dovrebbe avere la struttura seguente.

```
{
    "Filters": [
        {
            "Pattern": "{ \"value\" : { \"device_ID\" : [ { \"prefix\": \"AB\" } ] } }"
        }
    ]
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
    "value": {
        "device_ID": [ { "prefix": "AB" } ]
      }
}
```

Puoi aggiungere il filtro utilizzando la console, la AWS CLI o un modello AWS SAM.

Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in Collegamento dei criteri di filtro a una mappatura dell'origine evento (console) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "value" : { "device_ID" : [ { "prefix":  "AB" } ] } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtraggio utilizzando l'AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
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\" } ] } }"}]}'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : { \"device_ID\" :
 [ { \"prefix\":  \"AB\" } ] } }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML dell'origine degli eventi.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "value" : { "device_ID" : [ { "prefix":  "AB" } ] } }'
```

Con Amazon MSK, puoi anche filtrare i record in cui il messaggio è una stringa semplice. Supponiamo di voler ignorare i messaggi la cui stringa è "errore". L'oggetto `FilterCriteria` dovrebbe avere la struttura seguente.

```
{
    "Filters": [
        {
            "Pattern": "{ \"value\" : [ { \"anything-but\": [ \"error\" ] } ] }"
        }
    ]
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
    "value": [
        {
```

```
        "anything-but": [ "error" ]
        }
    ]
}
```

Puoi aggiungere il filtro utilizzando la console, la AWS CLI o un modello AWS SAM.

Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in Collegamento dei criteri di filtro a una mappatura dell'origine evento (console) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "value" : [ { "anything-but": [ "error" ] } ] }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtraggio utilizzando l'AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
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\" ] } ] }"}]}'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"value\" : [ { \"anything-but\":
 [ \"error\" ] } ] }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML dell'origine degli eventi.

```
FilterCriteria:
  Filters:
```

```
    - Pattern: '{ "value" : [ { "anything-but": [ "error" ] } ] }'
```

I messaggi Kafka e Amazon MQ devono essere stringhe codificate in UTF-8, stringhe semplici o in formato JSON. Questo perché Lambda decodifica gli array di byte Kafka in UTF-8 prima di applicare i criteri di filtro. Se i messaggi utilizzano un'altra codifica, ad esempio UTF-16 o ASCII o se il formato del messaggio non corrisponde al formato `FilterCriteria`, Lambda elabora solo i filtri di metadati. La tabella seguente riepiloga il comportamento specifico:

Formato messaggio in arrivo	Formato del modello di filtro per le proprietà di messaggi	Operazione risultante
Stringa normale	Stringa normale	Filtri Lambda in base ai criteri di filtro.
Stringa normale	Nessun modello di filtro per le proprietà dei dati	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.
Stringa normale	JSON valido	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.
JSON valido	Stringa normale	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.
JSON valido	Nessun modello di filtro per le proprietà dei dati	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.
JSON valido	JSON valido	Filtri Lambda in base ai criteri di filtro.
Stringa codificata non UTF-8	JSON, stringa semplice o nessun modello	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.

Utilizzo dei registri degli schemi con le sorgenti di eventi Kafka in Lambda

I registri degli schemi ti aiutano a definire e gestire gli schemi dei flussi di dati. Uno schema definisce la struttura e il formato di un registro di dati. Nel contesto delle mappature delle sorgenti degli eventi Kafka, puoi configurare un registro degli schemi per convalidare la struttura e il formato dei messaggi Kafka rispetto a schemi predefiniti prima che raggiungano la tua funzione Lambda. Ciò aggiunge un livello di governance dei dati all'applicazione e consente di gestire in modo efficiente i formati di dati, garantire la conformità allo schema e ottimizzare i costi tramite il filtraggio degli eventi.

Questa funzionalità è compatibile con tutti i linguaggi di programmazione, ma considera questi punti importanti:

- Powertools for Lambda fornisce supporto specifico per Java, Python e mantiene la coerenza con i modelli di sviluppo Kafka esistenti TypeScript e consente l'accesso diretto agli oggetti aziendali senza codice di deserializzazione personalizzato
- Questa funzionalità è disponibile solo per le mappature delle sorgenti degli eventi che utilizzano la modalità provisioned. Il registro degli schemi non supporta i mapping delle sorgenti degli eventi in modalità su richiesta. Se si utilizza la modalità provisioning e si dispone di un registro degli schemi configurato, non è possibile passare alla modalità su richiesta a meno che non si rimuova prima la configurazione del registro dello schema. Per ulteriori informazioni, consulta the section called “Modalità provisioning”
- È possibile configurare un solo registro degli schemi per Event Source Mapping (ESM). L'utilizzo di un registro degli schemi con l'origine degli eventi Kafka può aumentare l'utilizzo della Lambda Event Poller Unit (EPU), che è una dimensione tariffaria per la modalità Provisioned.

Argomenti

- Opzioni del registro dello schema
- Come Lambda esegue la convalida dello schema per i messaggi Kafka
- Configurazione di un registro degli schemi Kafka
- Filtraggio per Avro e Protobuf
- Formati di payload e comportamento di deserializzazione
- Utilizzo di dati deserializzati nelle funzioni Lambda
- Metodi di autenticazione per il registro degli schemi
- Gestione degli errori e risoluzione dei problemi relativi al registro dello schema

Opzioni del registro dello schema

Lambda supporta le seguenti opzioni di registro dello schema:

- AWS Glue Registro degli schemi
- Registro degli schemi Confluent Cloud
- Registro degli schemi Confluent autogestito

Il registro degli schemi supporta la convalida dei messaggi nei seguenti formati di dati:

- Apache Avro
- Buffer di protocollo (Protobuf)
- Schema JSON (JSON-SE)

Per utilizzare un registro degli schemi, assicurati innanzitutto che la mappatura delle sorgenti degli eventi sia in modalità provisioning. Quando si utilizza un registro dello schema, Lambda aggiunge i metadati relativi allo schema al payload. Per ulteriori informazioni, consulta Formati di payload e comportamento di deserializzazione.

Come Lambda esegue la convalida dello schema per i messaggi Kafka

Quando configuri un registro degli schemi, Lambda esegue i seguenti passaggi per ogni messaggio Kafka:

1. Lambda analizza il record di Kafka dal tuo cluster.
2. Lambda convalida gli attributi dei messaggi selezionati nel record rispetto a uno schema specifico nel registro degli schemi.
   - Se lo schema associato al messaggio non viene trovato nel registro, Lambda invia il messaggio a un DLQ con codice motivo. `SCHEMA_NOT_FOUND`
3. Lambda deserializza il messaggio in base alla configurazione del registro dello schema per convalidarlo. Se il filtraggio degli eventi è configurato, Lambda esegue quindi il filtraggio in base ai criteri di filtro configurati.
   - Se la deserializzazione fallisce, Lambda invia il messaggio a un DLQ con il codice motivo. `DESERIALIZATION_ERROR` Se non è configurato alcun DLQ, Lambda rilascia il messaggio.
4. Se il messaggio è convalidato dal registro dello schema e non viene filtrato in base ai criteri di filtro, Lambda richiama la funzione con il messaggio.

Questa funzionalità ha lo scopo di convalidare i messaggi già prodotti utilizzando client Kafka integrati con un registro degli schemi. Ti consigliamo di configurare i tuoi produttori Kafka in modo che utilizzino il registro degli schemi per creare messaggi formattati correttamente.

Configurazione di un registro degli schemi Kafka

I seguenti passaggi della console aggiungono una configurazione del registro dello schema Kafka alla mappatura delle sorgenti degli eventi.

Per aggiungere una configurazione del registro dello schema Kafka alla mappatura delle sorgenti degli eventi (console)

1. Apri la pagina Funzione della console Lambda.
2. Scegliere Configuration (Configurazione).
3. Scegli Triggers.
4. Seleziona la mappatura della sorgente degli eventi Kafka per cui desideri configurare un registro dello schema e scegli Modifica.
5. In Event poller configuration, scegli Configura il registro dello schema. La mappatura delle sorgenti degli eventi deve essere in modalità provisioning per visualizzare questa opzione.
6. Per l'URI del registro degli schemi, inserisci l'ARN del registro degli AWS Glue schemi o l'URL HTTPS del registro degli schemi di Confluent Cloud o del registro degli schemi Confluent Self-Managed.
7. I seguenti passaggi di configurazione indicano a Lambda come accedere al registro degli schemi. Per ulteriori informazioni, consulta ???.
   - Per il tipo di configurazione Access, scegli il tipo di autenticazione che Lambda utilizza per accedere al registro dello schema.
   - Per l'URI di configurazione di Access, inserisci l'ARN del segreto Secrets Manager per l'autenticazione con il registro dello schema, se applicabile. Assicurati che il ruolo di esecuzione della funzione contenga le autorizzazioni corrette.
8. Il campo Encryption si applica solo se il registro dello schema è firmato da un'Autorità di certificazione (CA) privata o da un'autorità di certificazione (CA) che non si trova nel Lambda Trust Store. Se applicabile, fornisci la chiave segreta contenente il certificato CA privato utilizzato dal registro dello schema per la crittografia TLS.

9. Per il formato di registrazione degli eventi, scegli come desideri che Lambda fornisca i record della tua funzione dopo la convalida dello schema. Per ulteriori informazioni, consulta Esempi di formato Payload.

   - Se scegli JSON, Lambda fornisce gli attributi selezionati nell'attributo di convalida dello schema riportato di seguito in formato JSON standard. Per gli attributi che non selezioni, Lambda li fornisce così come sono.
   - Se scegli SOURCE, Lambda fornisce gli attributi selezionati nell'attributo di convalida dello schema riportato di seguito nel formato di origine originale.

10. Per l'attributo di convalida dello schema, seleziona gli attributi del messaggio che desideri che Lambda convalidi e deserializzi utilizzando il registro dello schema. È necessario selezionare almeno uno dei valori KEY o VALUE. Se hai scelto JSON per il formato di registrazione degli eventi, Lambda deserializza anche gli attributi del messaggio selezionati prima di inviarli alla tua funzione. Per ulteriori informazioni, consulta Formati di payload e comportamento di deserializzazione.
11. Scegli Save (Salva).

Puoi anche utilizzare l'API Lambda per creare o aggiornare la mappatura delle sorgenti degli eventi con una configurazione del registro dello schema. Gli esempi seguenti mostrano come configurare un registro di schemi AWS Glue o Confluent utilizzando AWS CLI, che corrisponde alle operazioni UpdateEventSourceMappinge CreateEventSourceMappingAPI nell'API Reference:AWS Lambda

> **Important**
>
> Se si sta aggiornando un campo di configurazione del registro dello schema utilizzando AWS CLI o l'`update-event-source-mapping`API, è necessario aggiornare tutti i campi della configurazione del registro dello schema.

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

Per rimuovere una configurazione del registro dello schema dalla mappatura dell'origine degli eventi, puoi utilizzare il UpdateEventSourceMappingcomando AWS Lambda CLI nell'API Reference.

```
aws lambda update-event-source-mapping \
```

```
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
    --amazon-managed-kafka-event-source-mapping '{
        "SchemaRegistryConfig" : {}
    }'
```

Filtraggio per Avro e Protobuf

Quando si utilizzano i formati Avro o Protobuf con un registro degli schemi, è possibile applicare il filtro degli eventi alla funzione Lambda. I modelli di filtro vengono applicati alla rappresentazione JSON classica deserializzata dei dati dopo la convalida dello schema. Ad esempio, con uno schema Avro che definisce i dettagli del prodotto, incluso il prezzo, puoi filtrare i messaggi in base al valore del prezzo:

> **Note**
>
> Quando viene deserializzato, Avro viene convertito in JSON standard, il che significa che non può essere riconvertito direttamente in un oggetto Avro. Se devi eseguire la conversione in un oggetto Avro, usa invece il formato SOURCE.
> Per la deserializzazione di Protobuf, i nomi dei campi nel JSON risultante corrispondono a quelli definiti nello schema, anziché essere convertiti in camel case come fa normalmente Protobuf. Tienilo a mente quando crei modelli di filtraggio.

```
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\"] } }"
            }
        ]
    }'
```

In questo esempio, lo schema di filtro analizza l'`value`oggetto, associando i messaggi `field_1` con `"value1"` e `field_2` con. `"value2"` I criteri di filtro vengono valutati rispetto ai dati deserializzati, dopo che Lambda converte il messaggio dal formato Avro a JSON.

Per informazioni più dettagliate sul filtraggio degli eventi, consulta Filtraggio degli eventi Lambda.

Formati di payload e comportamento di deserializzazione

Quando si utilizza un registro degli schemi, Lambda fornisce il payload finale alla funzione in un formato simile al normale payload degli eventi, con alcuni campi aggiuntivi. I campi aggiuntivi dipendono dal parametro. `SchemaValidationConfigs` Per ogni attributo selezionato per la convalida (chiave o valore), Lambda aggiunge i metadati dello schema corrispondenti al payload.

> **Note**
>
> Devi aggiornare il tuo aws-lambda-java-eventsalla versione 3.16.0 o successiva per utilizzare i campi dei metadati dello schema.

Ad esempio, se convalidi il `value` campo, Lambda aggiunge un campo `valueSchemaMetadata` chiamato al tuo payload. Allo stesso modo, per il `key` campo, Lambda aggiunge un campo chiamato. `keySchemaMetadata` Questi metadati contengono informazioni sul formato dei dati e sull'ID dello schema utilizzati per la convalida:

```
"valueSchemaMetadata": {
    "dataFormat": "AVRO",
    "schemaId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

Il `EventRecordFormat` parametro può essere impostato su `JSON` o`SOURCE`, il che determina in che modo Lambda gestisce i dati convalidati dallo schema prima di consegnarli alla tua funzione. Ogni opzione offre diverse funzionalità di elaborazione:

- `JSON`- Lambda deserializza gli attributi convalidati in formato JSON standard, rendendo i dati pronti per l'uso diretto nelle lingue con supporto JSON nativo. Questo formato è ideale quando non è necessario conservare il formato binario originale o lavorare con classi generate.
- `SOURCE`- Lambda conserva il formato binario originale dei dati come stringa con codifica Base64, che consente la conversione diretta in oggetti Avro o Protobuf. Questo formato è essenziale quando si lavora con linguaggi fortemente tipizzati o quando è necessario mantenere tutte le funzionalità degli schemi Avro o Protobuf.

In base a queste caratteristiche di formato e alle considerazioni specifiche della lingua, consigliamo i seguenti formati:

Formati consigliati basati sul linguaggio di programmazione

Lingua	Avro	Protobuf	JSON
Java	SOURCE	SOURCE	SOURCE
Python	JSON	JSON	JSON
NodeJS	JSON	JSON	JSON
.NET	SOURCE	SOURCE	SOURCE
Altri	JSON	JSON	JSON

Le sezioni seguenti descrivono questi formati in dettaglio e forniscono esempi di payload per ogni formato.

Formato JSON

Se scegli `JSON` come`EventRecordFormat`, Lambda convalida e deserializza gli attributi del messaggio che hai selezionato nel campo (gli attributi). `SchemaValidationConfigs` `key` and/or `value` Lambda fornisce questi attributi selezionati come stringhe con codifica base64 della loro rappresentazione JSON standard nella funzione.

> **Note**
>
> Quando viene deserializzato, Avro viene convertito in JSON standard, il che significa che non può essere riconvertito direttamente in un oggetto Avro. Se devi eseguire la conversione in un oggetto Avro, usa invece il formato SOURCE.
> Per la deserializzazione di Protobuf, i nomi dei campi nel JSON risultante corrispondono a quelli definiti nello schema, anziché essere convertiti in camel case come fa normalmente Protobuf. Tienilo a mente quando crei modelli di filtraggio.

Di seguito viene mostrato un esempio di payload, supponendo che tu scelga `JSON` come e che entrambi gli attributi `key` e `value` siano: `EventRecordFormat` `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
                ]
            }
        ]
    }
  ]
 }
}
```

In questo esempio:

- Entrambi `key` `value` sono stringhe codificate in base64 della loro rappresentazione JSON dopo la deserializzazione.
- Lambda include i metadati dello schema per entrambi gli attributi in e. `keySchemaMetadata` `valueSchemaMetadata`
- La tua funzione può decodificare le `value` stringhe `key` and per accedere ai dati JSON deserializzati.

Il formato JSON è consigliato per i linguaggi che non sono fortemente tipizzati, come Python o Node.js. Questi linguaggi dispongono del supporto nativo per la conversione di JSON in oggetti.

Formato sorgente

Se scegli `SOURCE` come`EventRecordFormat`, Lambda convalida comunque il record rispetto al registro dello schema, ma fornisce i dati binari originali alla tua funzione senza deserializzazione. Questi dati binari vengono forniti come stringa codificata in Base64 dei dati in byte originali, con i metadati aggiunti dal produttore rimossi. Di conseguenza, è possibile convertire direttamente i dati binari non elaborati in oggetti Avro e Protobuf all'interno del codice della funzione. Ti consigliamo di utilizzare Powertools for AWS Lambda, che deserializzerà i dati binari grezzi e ti fornirà direttamente gli oggetti Avro e Protobuf.

Ad esempio, se configuri Lambda per convalidare entrambi `value` gli attributi `key` e ma utilizzi il `SOURCE` formato, la funzione riceve un payload come questo:

```
{
    "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
                        ]
                    }
                ]
            }
        ]
    }
}
```

In questo esempio:

- Entrambi `value` contengono `key` i dati binari originali come stringhe codificate in Base64.
- La tua funzione deve gestire la deserializzazione utilizzando le librerie appropriate.

La scelta di `SOURCE` for `EventRecordFormat` è consigliata se utilizzate oggetti generati da AVRO o da ProtoBuf, specialmente con le funzioni Java. Questo perché Java è fortemente tipizzato e richiede deserializzatori specifici per i formati Avro e Protobuf. Nel codice della funzione, puoi usare la tua libreria Avro o Protobuf preferita per deserializzare i dati.

Utilizzo di dati deserializzati nelle funzioni Lambda

Powertools for AWS Lambda consente di deserializzare i record Kafka nel codice della funzione in base al formato utilizzato. Questa utilità semplifica l'utilizzo dei record di Kafka gestendo la conversione dei dati e fornendo oggetti. ready-to-use

Per utilizzare Powertools for AWS Lambda nella tua funzione, devi aggiungere Powertools come livello o includerlo come dipendenza quando crei la tua funzione Lambda. AWS Lambda Per istruzioni di configurazione e ulteriori informazioni, consultate Powertools per la AWS Lambda documentazione relativa alla vostra lingua preferita:

- Powertools per AWS Lambda (Java)
- Utensili elettrici per AWS Lambda (Python)
- Utensili elettrici per () AWS Lambda TypeScript
- Utensili elettrici per AWS Lambda (.NET)

> **Note**
>
> Quando si lavora con l'integrazione del registro dello schema, è possibile scegliere `SOURCE` o `JSON` formattare. Ogni opzione supporta diversi formati di serializzazione, come illustrato di seguito:

Formato	Supporta
SOURCE	Avro e Protobuf (utilizzando l'integrazione del Lambda Schema Registry)
JSON	Dati JSON

Quando si utilizza il `JSON` formato `SOURCE` o, è possibile utilizzare Powertools per aiutare AWS a deserializzare i dati nel codice della funzione. Ecco alcuni esempi di come gestire diversi formati di dati:

AVRO

Esempio di 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";
    }

}
```

Esempio di 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 esempio:

```
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,
        },
    }
);
```

Esempio.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 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

Esempio di 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";
    }

}
```

Esempio di 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 esempio:

```
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,
        },
    }
);
```

Esempio.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

Esempio di 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";
    }
}
```

Esempio di 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 esempio:

```
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',
        },
    }
);
```

Esempio.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";
    }
}
```

Metodi di autenticazione per il registro degli schemi

Per utilizzare un registro degli schemi, Lambda deve essere in grado di accedervi in modo sicuro. Se lavori con un registro di AWS Glue schemi, Lambda si affida all'autenticazione IAM. Ciò significa che il ruolo di esecuzione della funzione deve disporre delle seguenti autorizzazioni per accedere al registro: AWS Glue

- GetRegistrynel AWS Glue Web API Reference
- GetSchemaVersionnel AWS Glue Web API Reference

Esempio della policy IAM richiesta:

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "glue:GetRegistry",
                "glue:GetSchemaVersion"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

> **Note**
>
> Per i registri AWS Glue dello schema, se fornisci `AccessConfigs` un AWS Glue registro, Lambda restituirà un'eccezione di convalida.

Se lavori con un registro di schemi Confluent, puoi scegliere uno dei tre metodi di autenticazione supportati per il parametro del tuo oggetto: `Type` KafkaSchemaRegistryAccessConfig

- BASIC_AUTH — Lambda utilizza nome utente e password o chiave API e autenticazione segreta API per accedere al registro. Se scegli questa opzione, fornisci l'ARN di Secrets Manager contenente le tue credenziali nel campo URI.
- CLIENT_CERTIFICATE_TLS_AUTH — Lambda utilizza l'autenticazione TLS reciproca con i certificati client. Per utilizzare questa opzione, Lambda deve accedere sia al certificato che alla chiave privata. Fornisci l'ARN di Secrets Manager contenente queste credenziali nel campo URI.
- NO_AUTH — Il certificato CA pubblico deve essere firmato da un'autorità di certificazione (CA) presente nel Lambda Trust Store. Per un certificato CA privato/autofirmato, si configura il certificato CA principale del server. Per utilizzare questa opzione, ometti il parametro. `AccessConfigs`

Inoltre, se Lambda necessita dell'accesso a un certificato CA privato per verificare il certificato TLS del registro dello schema, scegli `SERVER_ROOT_CA_CERT` `Type` e fornisci l'ARN di Secrets Manager al certificato nel campo URI.

> **Note**
>
> Per configurare l'`SERVER_ROOT_CA_CERT`opzione nella console, fornisci l'ARN segreto contenente il certificato nel campo Crittografia.

La configurazione di autenticazione per il registro degli schemi è separata da qualsiasi autenticazione configurata per il cluster Kafka. È necessario configurarli entrambi separatamente, anche se utilizzano metodi di autenticazione simili.

Gestione degli errori e risoluzione dei problemi relativi al registro dello schema

Quando si utilizza un registro degli schemi con l'origine degli eventi Amazon MSK, è possibile che si verifichino diversi errori. Questa sezione fornisce indicazioni sui problemi più comuni e su come risolverli.

Errori di configurazione

Questi errori si verificano durante l'impostazione della configurazione del registro dello schema.

È richiesta la modalità di provisioning

- Messaggio di errore: `SchemaRegistryConfig is only available for Provisioned Mode. To configure Schema Registry, please enable Provisioned Mode by specifying MinimumPollers in ProvisionedPollerConfig.`

- Risoluzione: abilita la modalità provisioning per la mappatura delle sorgenti degli eventi configurando il parametro in. `MinimumPollers` `ProvisionedPollerConfig`

URL del registro dello schema non valido

- Messaggio di errore: `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.`

- Risoluzione: fornire un URL HTTPS valido per Confluent Schema Registry o un ARN valido per Schema Registry. AWS Glue

Formato di registrazione degli eventi non valido o mancante

Messaggio di errore: `EventRecordFormat is a required field for SchemaRegistryConfig. Please provide one of supported format types: SOURCE, JSON.`

Risoluzione: specifica SOURCE o JSON come file EventRecordFormat nella configurazione del registro dello schema.

Attributi di convalida duplicati

Messaggio di errore: `Duplicate KEY/VALUE Attribute in SchemaValidationConfigs. SchemaValidationConfigs must contain at most one KEY/VALUE Attribute.`

Risoluzione: rimuovi gli attributi KEY o VALUE duplicati dal tuo. SchemaValidationConfigs Ogni tipo di attributo può apparire solo una volta.

Configurazione di convalida mancante

Messaggio di errore: `SchemaValidationConfigs is a required field for SchemaRegistryConfig.`

Risoluzione: aggiungi SchemaValidationConfigs alla tua configurazione, specificando almeno un attributo di convalida (KEY o VALUE).

Errori di accesso e autorizzazione

Questi errori si verificano quando Lambda non può accedere al registro dello schema a causa di problemi di autorizzazione o autenticazione.

AWS Glue Accesso al registro degli schemi negato

Messaggio di errore: `Cannot access Glue Schema with provided role. Please ensure the provided role can perform the GetRegistry and GetSchemaVersion Actions on your schema.`

Risoluzione: aggiungi le autorizzazioni richieste (`glue:GetRegistryeglue:GetSchemaVersion`) al ruolo di esecuzione della funzione.

Accesso al Confluent Schema Registry negato

Messaggio di errore: `Cannot access Confluent Schema with the provided access configuration.`

Risoluzione: verifica che le credenziali di autenticazione (archiviate in Secrets Manager) siano corrette e dispongano delle autorizzazioni necessarie per accedere al registro dello schema.

Registro degli schemi tra più account AWS Glue

Messaggio di errore: `Cross-account Glue Schema Registry ARN not supported.`

Risoluzione: utilizza un registro AWS Glue degli schemi che si trova nello stesso AWS account della funzione Lambda.

Registro degli schemi tra regioni AWS Glue

Messaggio di errore: `Cross-region Glue Schema Registry ARN not supported.`

Risoluzione: utilizza un registro AWS Glue degli schemi che si trova nella stessa area della funzione Lambda.

Problemi di accesso segreto

Messaggio di errore: `Lambda received InvalidRequestException from Secrets Manager.`

Risoluzione: verifica che il ruolo di esecuzione della funzione disponga dell'autorizzazione ad accedere al segreto e che il segreto non sia crittografato con una AWS KMS chiave predefinita se si accede da un account diverso.

Errori di connessione

Questi errori si verificano quando Lambda non riesce a stabilire una connessione al registro dello schema.

Problemi di connettività VPC

Messaggio di errore: `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.`

Risoluzione: configura la rete VPC per consentire le connessioni al registro dello schema utilizzando AWS PrivateLink un gateway NAT o un peering VPC.

Errore di handshake TLS

Messaggio di errore: `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.`

Risoluzione: verifica che i certificati CA e i certificati client (per MTL) siano corretti e configurati correttamente in Secrets Manager.

Throttling

Messaggio di errore: `Receiving throttling errors when accessing the schema registry. Please increase API TPS limits for your schema registry.`

Risoluzione: aumenta i limiti di velocità delle API per il registro degli schemi o riduci la frequenza delle richieste provenienti dall'applicazione.

Errori del registro dello schema autogestiti

Messaggio di errore: `Lambda received an internal server an unexpected error from the provided self-managed schema registry.`

Risoluzione: verifica lo stato e la configurazione del server del registro degli schemi autogestito.

Elaborazione a bassa latenza per sorgenti di eventi Kafka

AWS Lambdasupporta nativamente l'elaborazione di eventi a bassa latenza per applicazioni che richiedono latenze end-to-end costanti inferiori a 100 millisecondi. Questa pagina fornisce dettagli di configurazione e consigli per abilitare flussi di lavoro a bassa latenza.

Elaborazione a bassa latenza

Per abilitare l'elaborazione a bassa latenza su una mappatura delle sorgenti di eventi Kafka, è richiesta la seguente configurazione di base:

- Abilita la modalità provisioning; Per ulteriori informazioni, consulta Modalità provisioning.
- Imposta il `MaximumBatchingWindowInSeconds` parametro della mappatura della sorgente degli eventi su 0. Per ulteriori informazioni, consulta Comportamento di batching.

Ottimizzazione del tuo Kafka ESM a bassa latenza

Considerate i seguenti consigli per ottimizzare la mappatura delle sorgenti di eventi Kafka per una bassa latenza:

Configurazione della modalità provisioning

In modalità provisioning per la mappatura dell'origine degli eventi Kafka, Lambda consente di ottimizzare il throughput dello strumento di mappatura dell'origine degli eventi configurando un numero minimo e massimo di risorse chiamate poller di eventi. Un event poller (o poller) rappresenta una risorsa di calcolo che è alla base di una mappatura delle sorgenti di eventi nella modalità fornita e alloca un throughput fino a 5 MB/s. Ogni event poller supporta fino a 5 chiamate Lambda simultanee.

Per determinare la configurazione ottimale del poller per la vostra applicazione, tenete conto della velocità di ingestione massima e dei requisiti di elaborazione. Diamo un'occhiata a un esempio semplificato:

Con un batch di 20 record e una durata media della funzione target di 50 ms, ogni poller può gestire 2.000 record al secondo entro il limite di 5 MB/s. Viene calcolato come: (20 record × 1000 ms/50 ms) × 5 chiamate Lambda simultanee. Pertanto, se la velocità di ingestione massima desiderata è di 20.000 record al secondo, sono necessari almeno 10 event poller.

> **Note**
>
> Si consiglia di fornire sondaggi di eventi aggiuntivi come buffer per evitare di operare costantemente alla massima capacità.

La modalità Provisioned ridimensiona automaticamente i poller degli eventi in base ai modelli di traffico all'interno di poller di eventi minimi e massimi configurati, che possono attivare il ribilanciamento e quindi introdurre una latenza aggiuntiva. È possibile disabilitare l'auto-scaling configurando lo stesso valore per il poller di eventi minimo e massimo.

Ulteriori considerazioni

Ecco alcune considerazioni aggiuntive:

- L'avvio a freddo dall'invocazione della funzione di destinazione Lambda può potenzialmente aumentare la latenza end-to-end; Per ridurre questo rischio, prendete in considerazione l'attivazione di provisioned concurrency o SnapStart sulla funzione di destinazione della mappatura delle sorgenti degli eventi. Inoltre, ottimizza l'allocazione della memoria della funzione per garantire esecuzioni coerenti e ottimali.
- Quando `MaximumBatchingWindowInSeconds` è impostato su 0, Lambda elaborerà immediatamente tutti i record disponibili senza attendere di riempire la dimensione completa del batch. Ad esempio, se la dimensione del batch è impostata su 1.000 record ma sono disponibili solo 100 record, Lambda elaborerà tali 100 record immediatamente anziché attendere che si accumulino tutti i 1.000 record.

> **Important**
>
> La configurazione ottimale per l'elaborazione a bassa latenza varia in modo significativo in base al carico di lavoro specifico. Ti consigliamo vivamente di testare diverse configurazioni con il tuo carico di lavoro effettivo per determinare le impostazioni migliori per il tuo caso d'uso.

Configurazione dei controlli di gestione degli errori per le sorgenti di eventi Kafka

Puoi configurare il modo in cui Lambda gestisce gli errori e i nuovi tentativi per le mappature delle sorgenti degli eventi Kafka. Queste configurazioni consentono di controllare il modo in cui Lambda elabora i record non riusciti e gestisce il comportamento dei tentativi.

Configurazioni di nuovi tentativi disponibili

Le seguenti configurazioni di riprova sono disponibili sia per Amazon MSK che per le sorgenti di eventi Kafka autogestite:

- Numero massimo di tentativi: il numero massimo di tentativi Lambda quando la funzione restituisce un errore. Questo non conta il tentativo di invocazione iniziale. L'impostazione predefinita è -1 (infinito).
- Età massima di registrazione: l'età massima di un record che Lambda invia alla tua funzione. L'impostazione predefinita è -1 (infinito).
- Suddividi il batch in caso di errore: quando la funzione restituisce un errore, dividi il batch in due batch più piccoli e riprova ciascuno separatamente. Questo aiuta a isolare i record problematici.
- Risposta batch parziale: consenti alla funzione di restituire informazioni sui record di un batch che non è riuscita a elaborare, in modo che Lambda possa riprovare solo i record non riusciti.

Configurazione dei controlli di gestione degli errori (console)

Puoi configurare il comportamento dei tentativi durante la creazione o l'aggiornamento di una mappatura della sorgente degli eventi Kafka nella console Lambda.

Per configurare il comportamento di ripetizione dei tentativi per una sorgente di eventi Kafka (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliete il nome della funzione.
3. Esegui una delle seguenti operazioni:
    - Per aggiungere un nuovo trigger Kafka, in Panoramica delle funzioni, scegli Aggiungi trigger.
    - Per modificare un trigger Kafka esistente, scegli il trigger, quindi scegli Modifica.

4. In Event poller configuration, seleziona la modalità provisioned per configurare i controlli di gestione degli errori:
   a. Per Riprovare, inserisci il numero massimo di tentativi (0-10000 o -1 per un numero infinito).
   b. Per Durata massima di registrazione, inserisci l'età massima in secondi (60-604800 o -1 per un numero infinito).
   c. Per abilitare la suddivisione in batch quando si verificano errori, seleziona Dividi batch in caso di errore.
   d. Per abilitare la risposta parziale in batch, selezionare ReportBatchItemFailures.
5. Scegliete Aggiungi o Salva.

Configurazione del comportamento dei tentativi ()AWS CLI

Utilizzate i seguenti AWS CLI comandi per configurare il comportamento dei tentativi di ripetizione per le mappature delle sorgenti degli eventi Kafka.

Creazione di una mappatura delle sorgenti degli eventi con configurazioni di nuovo tentativo

L'esempio seguente crea una mappatura delle sorgenti degli eventi Kafka autogestita con controlli di gestione degli errori:

```
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"
```

Per le sorgenti di eventi Amazon MSK:

```
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"
```

Aggiornamento delle configurazioni dei tentativi

Usa il `update-event-source-mapping` comando per modificare le configurazioni dei tentativi per una mappatura dell'origine degli eventi esistente:

```
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

La risposta parziale in batch, nota anche come ReportBatchItemFailures, è una funzionalità chiave per la gestione degli errori nell'integrazione di Lambda con i sorgenti Kafka. Senza questa funzionalità, quando si verifica un errore in uno degli elementi di un batch, comporta la rielaborazione di tutti i messaggi in quel batch. Con la risposta batch parziale abilitata e implementata, il gestore restituisce gli identificatori solo per i messaggi non riusciti, permettendo a Lambda di riprovare solo quegli elementi specifici. Ciò fornisce un maggiore controllo sul modo in cui vengono elaborati i batch contenenti messaggi non riusciti.

Per segnalare gli errori relativi ai batch, utilizzerai questo schema JSON:

```
{
  "batchItemFailures": [
    {
      "itemIdentifier": {
        "topic-partition": "topic-partition_number",
        "offset": 100
```

```
      }
    },
    ...
  ]
}
```

> **Important**
>
> Se restituisci un codice JSON o null valido vuoto, la mappatura dell'origine dell'evento considererà un batch come elaborato correttamente. Qualsiasi topic-partition_number o offset non valido restituito che non era presente nell'evento richiamato verrà considerato un errore e l'intero batch verrà riprovato.

I seguenti esempi di codice mostrano come implementare una risposta batch parziale per le funzioni Lambda che ricevono eventi da sorgenti Kafka. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

Ecco un'implementazione del gestore Python Lambda che mostra questo approccio:

```
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] = {
```

```
                    "topic-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
```

Ecco una versione di Node.js:

```
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 = {
          "topic-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 };
```

Acquisizione di batch scartati per un'origine eventi di Amazon MSK e Apache Kafka autogestito

Per mantenere i record delle chiamate non riuscite allo strumento di mappatura dell'origine degli eventi, aggiungi una destinazione allo strumento di mappatura dell'origine degli eventi della funzione. Ogni record inviato alla destinazione è un documento JSON contenente i metadati relativi alla chiamata non riuscita. Per le destinazioni Amazon S3, Lambda invia insieme ai metadati anche l'intero record di invocazione. Puoi configurare qualsiasi argomento Amazon SNS, coda Amazon SQS, bucket Amazon S3 o Kafka come destinazione.

Con le destinazioni Amazon S3, puoi utilizzare la funzionalità Notifiche eventi Amazon S3 per ricevere notifiche quando gli oggetti vengono caricati nel bucket S3 di destinazione. Puoi anche configurare Notifiche eventi S3 per richiamare un'altra funzione Lambda per eseguire l'elaborazione automatica su batch non riusciti.

Il tuo ruolo di esecuzione deve avere le autorizzazioni per la destinazione:

- Per una destinazione SQS: sqs: SendMessage
- Per una destinazione SNS: sns:publish
- Per una destinazione S3: s3: e s3: PutObject ListBucket
- Per una destinazione Kafka: kafka-cluster: WriteData

È possibile configurare un argomento di Kafka come destinazione in caso di errore per le mappature delle sorgenti degli eventi Kafka. Quando Lambda non è in grado di elaborare i record dopo estenuanti tentativi o quando i record superano l'età massima, Lambda invia i record non riusciti all'argomento Kafka specificato per un'elaborazione successiva. Fai riferimento a the section called “Destinazione Kafka in caso di errore”.

È necessario implementare un endpoint VPC per il servizio di destinazione in caso di errore all'interno del VPC del cluster Kafka.

Inoltre, se hai configurato una chiave KMS sulla destinazione, Lambda necessita delle seguenti autorizzazioni, a seconda del tipo di destinazione:

- Se hai abilitato la crittografia con la tua chiave KMS per una destinazione S3, è richiesto kms:. GenerateDataKey Se la chiave KMS e la destinazione del bucket S3 si trovano in un account diverso dalla funzione Lambda e dal ruolo di esecuzione, configura la chiave KMS in modo che consideri attendibile il ruolo di esecuzione da consentire. kms: GenerateDataKey

- Se hai abilitato la crittografia con la tua chiave KMS per la destinazione SQS, sono necessari KMS:decrypt e kms:. GenerateDataKey Se la chiave KMS e la destinazione della coda SQS si trovano in un account diverso dalla funzione Lambda e dal ruolo di esecuzione, configura la chiave KMS in modo che consideri attendibile il ruolo di esecuzione per consentire kms: Decrypt,, kms: GenerateDataKey kms: e kms:. DescribeKey ReEncrypt
- Se hai abilitato la crittografia con la tua chiave KMS per la destinazione SNS, sono obbligatori KMS:Decrypt e kms:. GenerateDataKey Se la chiave KMS e la destinazione dell'argomento SNS si trovano in un account diverso dalla funzione Lambda e dal ruolo di esecuzione, configura la chiave KMS in modo che consideri attendibile il ruolo di esecuzione per consentire kms: Decrypt, kms:GenerateDataKey, kms: e kms:. DescribeKey ReEncrypt

Configurazione delle destinazioni in caso di errore per una mappatura della sorgente degli eventi Kafka

Per configurare una destinazione in caso di errore tramite la console, completa i seguenti passaggi:

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. In Function overview (Panoramica delle funzioni), scegliere Add destination (Aggiungi destinazione).
4. Per Origine, scegli Chiamata allo strumento di mappatura dell'origine degli eventi.
5. Per Strumento di mappatura dell'origine degli eventi, scegli un'origine dell'evento configurata per questa funzione.
6. Per Condizione, seleziona In caso di errore. Per le chiamate allo strumento di mappatura dell'origine degli eventi, questa è l'unica condizione accettata.
7. Per Tipo di destinazione, scegli il tipo di destinazione a cui Lambda deve inviare i record di chiamata.
8. Per Destination (Destinazione), scegliere una risorsa.
9. Scegli Save (Salva).

È inoltre possibile configurare una destinazione in errore utilizzando la AWS CLI. Ad esempio, il create-event-source-mappingcomando seguente aggiunge una mappatura dell'origine degli eventi con una destinazione SQS in caso di errore a: `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"}}'
```

Il update-event-source-mappingcomando seguente aggiunge una destinazione S3 in caso di errore all'origine dell'evento associata all'input: `uuid`

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": "arn:aws:s3:::dest-bucket"}}'
```

Per rimuovere una destinazione, fornisci una stringa vuota come argomento del parametro `destination-config`:

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

Best practice di sicurezza per destinazioni Amazon S3

L'eliminazione di un bucket S3 configurato come destinazione senza rimuovere la destinazione dalla configurazione della funzione può creare un rischio per la sicurezza. Se un altro utente conosce il nome del bucket di destinazione, può ricreare il bucket nel proprio Account AWS. I record delle invocazioni non riuscite verranno inviati al relativo bucket, esponendo potenzialmente i dati della tua funzione.

> **Warning**
>
> Per garantire che i record di invocazione della tua funzione non possano essere inviati a un bucket S3 in un altro Account AWS, aggiungi una condizione al ruolo di esecuzione della funzione che limiti le `s3:PutObject` autorizzazioni ai bucket del tuo account.

Di seguito viene illustrato un esempio di policy IAM che limita le autorizzazioni `s3:PutObject` della funzione ai bucket presenti nell'account. Questa policy fornisce inoltre a Lambda l'autorizzazione `s3:ListBucket` necessaria per utilizzare un bucket S3 come destinazione.

```
{
```

```
    "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"
                }
            }
        }
    ]
}
```

Per aggiungere una politica di autorizzazioni al ruolo di esecuzione della tua funzione utilizzando Console di gestione AWS o AWS CLI, consulta le istruzioni nelle seguenti procedure:

Console

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Seleziona la funzione Lambda di cui si desidera modificare il ruolo di esecuzione.
3. Nella scheda Configurazione scegli Autorizzazioni.
4. Nella scheda Ruolo di esecuzione, seleziona il nome del ruolo della funzione per aprire la pagina della console IAM del ruolo.
5. Aggiungi una policy di autorizzazioni di base al ruolo completando le seguenti operazioni:
    a. Nel riquadro Policy di autorizzazioni, scegli Aggiungi autorizzazioni, poi Crea policy in linea.
    b. Nell'editor delle policy, seleziona JSON.

c. Incolla la policy che desideri aggiungere nell'editor (sostituendo il codice JSON esistente), quindi scegli Avanti.

d. In Dettagli della policy, specifica un nome per la policy.

e. Scegli Crea policy.

AWS CLI

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (CLI)

1. Crea un documento di policy JSON con le autorizzazioni richieste e salvalo in una directory locale.
2. Utilizza il comando della CLI `put-role-policy` di IAM per aggiungere le autorizzazioni per il ruolo di esecuzione di una funzione. Esegui il comando seguente dalla directory in cui hai salvato il documento di policy JSON e sostituisci il nome del ruolo, il nome della policy e il documento di policy con i tuoi valori.

```
aws iam put-role-policy \
--role-name my_lambda_role \
--policy-name LambdaS3DestinationPolicy \
--policy-document file://my_policy.json
```

Record di invocazione SNS e SQS di esempio

L'esempio seguente mostra il contenuto che Lambda invia a un argomento SNS o una coda SQS di destinazione per una chiamata non riuscita all'origine dell'evento Kafka. Ciascuna delle chiavi in `recordsInfo` contiene sia l'argomento sia lo shard di Kafka, separati da un trattino. Ad esempio, per la chiave `"Topic-0"`, `Topic` è l'argomento di Kafka e `0` è lo shard. Per ogni argomento e partizione, è possibile utilizzare i dati di offset e timestamp per individuare i record di chiamata originali.

```
{
    "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",
            }
        }
    }
}
```

Record di invocazione di esempio di destinazione S3

Se le destinazioni sono S3, Lambda invia alla destinazione l'intero record di chiamata insieme ai metadati. L'esempio seguente mostra ciò che Lambda invia a un bucket S3 di destinazione per una chiamata non riuscita all'origine dell'evento Kafka. Oltre a tutti i campi dell'esempio precedente per le destinazioni SQS e SNS, il campo `payload` contiene il record di chiamata originale come stringa JSON con escape.

```
{
    "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**
>
> Ti consigliamo di abilitare il controllo delle versioni S3 sul bucket di destinazione.

Usare un argomento di Kafka come destinazione in caso di errore

È possibile configurare un argomento di Kafka come destinazione in caso di errore per le mappature delle sorgenti degli eventi Kafka. Quando Lambda non è in grado di elaborare i record dopo estenuanti tentativi o quando i record superano l'età massima, Lambda invia i record non riusciti all'argomento Kafka specificato per un'elaborazione successiva.

Come funziona una destinazione Kafka in caso di errore

Quando si configura un argomento di Kafka come destinazione in caso di errore, Lambda funge da produttore Kafka e scrive i record non riusciti nell'argomento di destinazione. Questo crea un pattern DLT (Dead Letter Topic) all'interno dell'infrastruttura Kafka.

- Stesso requisito del cluster: l'argomento di destinazione deve esistere nello stesso cluster Kafka degli argomenti di origine.
- Contenuto effettivo dei record: le destinazioni Kafka ricevono i record effettivamente non riusciti insieme ai metadati di errore.
- Prevenzione della ricorsione: Lambda previene i loop infiniti bloccando le configurazioni in cui gli argomenti di origine e destinazione coincidono.

Configurazione di una destinazione Kafka in caso di errore

È possibile configurare un argomento di Kafka come destinazione in caso di errore durante la creazione o l'aggiornamento di una mappatura dell'origine degli eventi Kafka.

Configurazione di una destinazione Kafka (console)

Configurare un argomento di Kafka come destinazione in caso di errore (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliete il nome della funzione.
3. Esegui una delle seguenti operazioni:
   - Per aggiungere un nuovo trigger Kafka, in Panoramica delle funzioni, scegli Aggiungi trigger.
   - Per modificare un trigger Kafka esistente, scegli il trigger, quindi scegli Modifica.
4. In Impostazioni aggiuntive, per Destinazione in caso di errore, scegli l'argomento Kafka.
5. Per Nome argomento, inserisci il nome dell'argomento Kafka a cui desideri inviare i record non riusciti.

6. Scegli Aggiungi o Salva.

Configurazione di una destinazione Kafka ()AWS CLI

Utilizzate il `kafka://` prefisso per specificare un argomento di Kafka come destinazione in caso di errore.

Creazione di una mappatura della sorgente degli eventi con la destinazione Kafka

L'esempio seguente crea una mappatura dell'origine degli eventi Amazon MSK con un argomento Kafka come destinazione in caso di errore:

```
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"}}'
```

Per Kafka autogestito, usa la stessa sintassi:

```
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"}}'
```

Aggiornamento di una destinazione Kafka

Usa il `update-event-source-mapping` comando per aggiungere o modificare una destinazione Kafka:

```
aws lambda update-event-source-mapping \
  --uuid 12345678-1234-1234-1234-123456789012 \
  --destination-config '{"OnFailure":{"Destination":"kafka://failed-records-topic"}}'
```

Formato di registrazione per una destinazione Kafka

Quando Lambda invia record non riusciti a un argomento di Kafka, ogni messaggio contiene sia metadati sull'errore che il contenuto effettivo del record.

Metadati di errore

I metadati includono informazioni sul motivo per cui il record ha avuto esito negativo e dettagli sul batch originale:


```
{
  "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": "49601189658422359378836298521827638475320189012309704722",
      "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=="
        }
      ]
    }
  }
}
```

Comportamento delle chiavi di partizione

Lambda utilizza la stessa chiave di partizione del record originale per la produzione sull'argomento di destinazione. Se il record originale non aveva alcuna chiave, Lambda utilizza il partizionamento round-robin predefinito di Kafka su tutte le partizioni disponibili nell'argomento di destinazione.

Requisiti e limitazioni

- È richiesta la modalità di provisioning: una destinazione Kafka in caso di errore è disponibile solo per le mappature delle sorgenti degli eventi con la modalità provisioning abilitata.
- Solo stesso cluster: l'argomento di destinazione deve esistere nello stesso cluster Kafka degli argomenti di origine.
- Autorizzazioni per gli argomenti: la mappatura dei sorgenti degli eventi deve disporre delle autorizzazioni di scrittura sull'argomento di destinazione. Esempio:

```
{
    "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": "*"
        }
    ]
}
```

- Nessuna ricorsione: il nome dell'argomento di destinazione non può essere lo stesso dei nomi degli argomenti di origine.

Risoluzione degli errori relativi allo strumento di mappatura dell'origine degli eventi di Apache Kafka autogestito

Negli argomenti seguenti vengono forniti suggerimenti per la risoluzione dei problemi relativi a errori e problemi che potrebbero verificarsi durante l'utilizzo di Amazon MSK o Apache Kafka autogestito con Lambda.

Per ulteriori informazioni sulla risoluzione dei problemi, visita il Knowledge Center AWS.

Errori di autenticazione e autorizzazione

Se manca una delle autorizzazioni necessarie per l'utilizzo dei dati dal cluster Kafka, Lambda visualizza un messaggio di errore nella mappatura della fonte evento in LastProcessingResult.

Messaggi di errore

- Il cluster non è riuscito ad autorizzare Lambda
- Autenticazione SASL non riuscita
- Il server non è riuscito ad autenticare Lambda
- Lambda non è riuscita ad autenticare il server
- Il certificato o la chiave privata forniti non sono validi

Il cluster non è riuscito ad autorizzare Lambda

Per SASL/SCRAM o mTLS, questo errore indica che l'utente fornito non dispone di tutte le seguenti autorizzazioni della lista di controllo accessi Kafka (ACL) richieste:

- Cluster DescribeConfigs
- Descrivi il gruppo
- Leggi il gruppo
- Descrivi l'argomento
- Leggi l'argomento

Quando si crea ACL Kafka con le autorizzazioni `kafka-cluster` richieste, è necessario specificare l'argomento e il gruppo come risorse. Il nome dell'argomento deve corrispondere all'argomento nella mappatura dell'origine eventi. Il nome del gruppo deve corrispondere all'UUID della mappatura dell'origine eventi.

Dopo avere aggiunto le autorizzazioni richieste al ruolo di esecuzione, potrebbero essere necessari alcuni minuti affinché le modifiche entrino in vigore.

Autenticazione SASL non riuscita

Per SASL/SCRAM o SASL/PLAIN, questo errore indica che le credenziali di accesso fornite non sono valide.

Per il controllo accessi IAM, al ruolo di esecuzione manca l'autorizzazione `kafka-cluster:Connect` per il cluster MSK. Aggiungi questa autorizzazione al ruolo e specifica l'Amazon Resource Name (ARN) del cluster come risorsa.

Potresti visualizzare questo errore in modo intermittente. Il cluster rifiuta le connessioni dopo che il numero di connessioni TCP supera la quota del servizio Amazon MSK. Lambda cessa e ritenta finché una connessione non ha esito positivo. Dopo che Lambda si connette al cluster e ha eseguito il polling dei registri, l'ultimo risultato di elaborazione cambia in `OK`.

Il server non è riuscito ad autenticare Lambda

Questo errore indica che il broker Kafka non è riuscito ad autenticare Lambda. Questo errore può verificarsi per uno dei seguenti motivi:

- Non è stato fornito un certificato client per l'autenticazione mTLS.
- È stato fornito un certificato client, ma i broker Kafka non sono configurati per l'utilizzo dell'autenticazione mTLS.
- Un certificato client non è attendibile per i broker Kafka.

Lambda non è riuscita ad autenticare il server

Questo errore indica che Lambda non è riuscita ad autenticare il broker Kafka. Questo errore può verificarsi per uno dei seguenti motivi:

- I broker Kafka utilizzano certificati autofirmati o una CA privata, ma non hanno fornito il certificato CA root del server.
- Il certificato CA root del server non corrisponde alla CA root che ha firmato il certificato del broker.
- La convalida del nome host non è riuscita perché il certificato del broker non contiene il nome DNS o l'indirizzo IP del broker come nome alternativo dell'oggetto.

Il certificato o la chiave privata forniti non sono validi

Questo errore indica che il consumatore Kafka non ha potuto utilizzare il certificato o la chiave privata fornita. Assicurati che il certificato e la chiave utilizzino il formato PEM e che la crittografia della chiave privata utilizzi un algoritmo PBES1.

Errori della mappatura dell'origine eventi

Quando aggiungi il cluster Apache Kafka come origine eventi per la funzione Lambda, se la funzione rileva un errore, il consumer Kafka arresta l'elaborazione dei record. I consumatori di uno shard dell'argomento sono quelli che sottoscrivono, leggono ed elaborano i record. Gli altri consumatori Kafka possono continuare a elaborare i record, a condizione che non riscontrino lo stesso errore.

Per determinare la causa di un consumatore interrotto, controlla il campo `StateTransitionReason` nella risposta di `EventSourceMapping`. Nell'elenco seguente vengono descritti gli errori dell'origine eventi che è possibile ricevere:

`ESM_CONFIG_NOT_VALID`

La configurazione della mappatura della fonte evento non è valida.

`EVENT_SOURCE_AUTHN_ERROR`

Lambda non ha potuto autenticare la fonte evento.

`EVENT_SOURCE_AUTHZ_ERROR`

Lambda non dispone delle autorizzazioni necessarie per accedere alla fonte evento.

`FUNCTION_CONFIG_NOT_VALID`

La configurazione della funzione non è valida.

> **Note**
>
> Se i record degli eventi Lambda superano il limite di dimensione consentito di 6 MB, potrebbero non venire elaborati.

Richiamo di funzione Lambda utilizzando un endpoint Gateway Amazon API

È possibile creare un'API web con un endpoint HTTP per la funzione Lambda utilizzando Amazon API Gateway. API Gateway fornisce strumenti per creare e documentare siti Web APIs che instradano le richieste HTTP verso le funzioni Lambda. È possibile proteggere l'accesso all'API con controlli di autenticazione e autorizzazione. APIs Puoi servire il traffico su Internet o essere accessibile solo all'interno del tuo VPC.

> **Tip**
>
> Lambda offre due modi per richiamare una funzione tramite un endpoint HTTP: API Gateway e funzione Lambda. URLs In caso di dubbi su quale sia il metodo migliore per il tuo caso d'uso, consulta the section called “API Gateway vs funzione URLs”.

Le risorse nell'API definiscono uno o più metodi, ad esempio GET o POST. I metodi hanno un'integrazione che instrada le richieste a una funzione Lambda o a un altro tipo di integrazione. È possibile definire ogni risorsa e metodo singolarmente oppure utilizzare tipi di risorse e metodi speciali per soddisfare tutte le richieste che si adattano a un modello. Una risorsa proxy cattura tutti i percorsi sottostanti una risorsa. Il metodo ANY cattura tutti i metodi HTTP.

Sections

- Scelta di un tipo di API
- Aggiunta di un endpoint alla funzione Lambda
- Integrazione proxy
- Formato dell'evento
- Formato della risposta
- Permissions
- Applicazione di esempio
- Il gestore di eventi di Powertools for Lambda AWS
- Tutorial: uso di Lambda con Amazon API Gateway
- Gestione degli errori con un'API di API Gateway
- Selezionare un metodo per richiamare la funzione Lambda tramite una richiesta HTTP

Scelta di un tipo di API

API Gateway supporta tre tipi di funzioni APIs che richiamano funzioni Lambda:

- API HTTP: un'API leggera e a bassa RESTful latenza.
- API REST: un'API personalizzabile e ricca di funzionalità RESTful .
- WebSocket API: un'API Web che mantiene connessioni persistenti con i client per comunicazioni full-duplex.

HTTP APIs e REST APIs elaborano entrambe RESTful APIs le richieste HTTP e restituiscono risposte. APIs Gli HTTP sono più recenti e sono creati con l'API API Gateway versione 2. Le seguenti funzionalità sono nuove per HTTP: APIs

Funzionalità API HTTP

- Distribuzioni automatiche: quando si modificano routing o integrazioni, le modifiche vengono distribuite automaticamente alle fasi in cui è abilitata la distribuzione automatica.
- Fase predefinita: è possibile creare una fase predefinita (`$default`) che invii le richieste nel percorso root dell'URL dell'API. Per le fasi denominate, è necessario includere il nome dello stage all'inizio del percorso.
- Configurazione CORS: è possibile configurare l'API in modo da aggiungere intestazioni CORS alle risposte in uscita, invece di aggiungerle manualmente nel codice della funzione.

APIs I REST sono i classici RESTful APIs che API Gateway ha supportato sin dal lancio. APIs Attualmente REST offre più funzionalità di personalizzazione, integrazione e gestione.

Funzionalità REST API

- Tipi di integrazione: REST APIs supporta integrazioni Lambda personalizzate. Con un'integrazione personalizzata, è possibile inviare solo il corpo della richiesta alla funzione, o applicare un modello di trasformazione al corpo della richiesta prima di inviarlo alla funzione.
- Controllo degli accessi: REST APIs supporta più opzioni per l'autenticazione e l'autorizzazione.
- Monitoraggio e tracciamento: REST APIs supporta il AWS X-Ray tracciamento e opzioni di registrazione aggiuntive.

Per un confronto dettagliato, consulta Choose between HTTP APIs and REST APIs nella API Gateway Developer Guide.

WebSocket APIs utilizza anche l'API API Gateway versione 2 e supporta un set di funzionalità simile. Utilizza un' WebSocket API per le applicazioni che traggono vantaggio da una connessione persistente tra il client e l'API. WebSocket APIs forniscono una comunicazione full-duplex, il che significa che sia il client che l'API possono inviare messaggi in modo continuo senza attendere una risposta.

HTTP APIs supporta un formato di eventi semplificato (versione 2.0). Per un esempio di evento da un'API HTTP, consulta Creare integrazioni AWS Lambda proxy per HTTP APIs in API Gateway.

Per ulteriori informazioni, consulta Creare integrazioni AWS Lambda proxy per HTTP APIs in API Gateway.

Aggiunta di un endpoint alla funzione Lambda

Per aggiungere un endpoint pubblico alla funzione Lambda

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. In Function overview (Panoramica delle funzioni), scegliere Add trigger (Aggiungi trigger).
4. Selezionare API Gateway.
5. Scegliere Create an API (Crea una nuova API) o Use an existing API (Usa un'API esistente).
    a. Nuova API: per API type (Tipo di API), scegliere HTTP API (API HTTP). Per ulteriori informazioni, consulta Scelta di un tipo di API.
    b. API esistente: seleziona l'API dal menu a discesa o inserisci l'ID API (ad esempio, r3pmxmplak).
6. Per Security (Sicurezza), scegliere Open (Apri).
7. Scegliere Add (Aggiungi).

Integrazione proxy

Gli API Gateway APIs sono composti da fasi, risorse, metodi e integrazioni. La fase e la risorsa determinano il percorso dell'endpoint:

Formato del percorso API

- `/prod/`: la fase e la risorsa radice di `prod`.
- `/prod/user`: la fase di `prod` e la risorsa di `user`.
- `/dev/{proxy+}` – Qualunque routing della fase `dev`.
- `/`— (HTTP APIs) La fase e la risorsa principale predefinite.

Un'integrazione Lambda mappa una combinazione di percorso e metodo HTTP a una funzione Lambda. È possibile configurare API Gateway in modo da passare il corpo della richiesta HTTP così com'è (integrazione personalizzata), o da incapsulare il corpo della richiesta in un documento che include tutte le informazioni sulla richiesta, incluse intestazioni, risorsa, percorso e metodo.

Per ulteriori informazioni, consulta Integrazioni del proxy Lambda di API Gateway.

Formato dell'evento

Amazon API Gateway richiama la funzione in modo sincrono con un evento che contiene una rappresentazione JSON della richiesta HTTP. Per un'integrazione personalizzata, l'evento è il corpo della richiesta. Per un'integrazione proxy, l'evento ha una struttura definita. Per un esempio di evento proxy proveniente da una REST API di API Gateway, consulta Formato di input di una funzione Lambda per l'integrazione del proxy nella Guida per gli sviluppatori di API Gateway.

Formato della risposta

API Gateway attende una risposta dalla funzione e inoltra il risultato al chiamante. Per un'integrazione personalizzata, è possibile definire una risposta di integrazione e una risposta al metodo per convertire l'output dalla funzione a una risposta HTTP. Per un'integrazione proxy, la funzione deve rispondere con una rappresentazione della risposta in un formato specifico.

L'esempio seguente mostra un oggetto risposta da una funzione Node.js. L'oggetto risposta rappresenta una risposta HTTP riuscita che contiene un documento JSON.

Example index.mjs: oggetto risposta di integrazione 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}"
  }
```

Il runtime Lambda serializza l'oggetto di risposta in JSON e lo invia all'API. L'API analizza la risposta e la utilizza per creare una risposta HTTP, che quindi la invia al client che ha effettuato la richiesta originale.

Example Risposta 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
  }
```

Permissions

Amazon API Gateway ottiene l'autorizzazione a richiamare la funzione dalla policy basata su risorse della funzione. È possibile concedere l'autorizzazione a un'intera API o concedere un accesso limitato a una fase, una risorsa o un metodo.

Quando aggiungi un'API alla funzione utilizzando la console Lambda, la console API Gateway o in un modello AWS SAM , la policy basata su risorse della funzione viene aggiornata automaticamente. Di seguito è riportata una policy di funzioni di esempio.

Example Policy di funzione

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/"
        }
      }
    }
  ]
}
```

È possibile gestire manualmente le autorizzazioni delle policy di funzione con le seguenti operazioni API:

- AddPermission
- RemovePermission
- GetPolicy

Per concedere l'autorizzazione di chiamata a un'API esistente, utilizzare il comando `add-permission`. Esempio:

```
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/"
```

Verrà visualizzato l'output seguente:

```
{
    "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**
>
> Se la funzione e l'API sono diverse Regioni AWS, l'identificatore di regione nell'ARN di origine deve corrispondere alla regione della funzione, non alla regione dell'API. Quando API Gateway richiama una funzione, utilizza un ARN di risorsa basato sull'ARN dell'API, ma modificato per corrispondere alla regione della funzione.

L'ARN di origine in questo esempio concede l'autorizzazione a un'integrazione nel metodo GET della risorsa root nella fase predefinita di un'API, con ID `mnh1xmpli7`. È possibile utilizzare un asterisco nell'ARN di origine per concedere autorizzazioni a più fasi, metodi o risorse.

Modelli di risorse

- `mnh1xmpli7/*/GET/*`: metodo GET su tutte le risorse in tutte le fasi.
- `mnh1xmpli7/prod/ANY/user`: metodo ANY sulla risorsa `user` nella fase `prod`.
- `mnh1xmpli7/*/*/*`: qualsiasi metodo su tutte le risorse in tutte le fasi.

Per informazioni dettagliate sulla visualizzazione delle policy e sulla rimozione delle istruzioni, vedere Visualizzazione delle policy IAM basate sulle risorse in Lambda.

Applicazione di esempio

L'app di esempio API Gateway with Node.js include una funzione con un AWS SAM modello che crea un'API REST con AWS X-Ray tracciamento abilitato. Include anche script per l'implementazione, il richiamo della funzione, il test dell'API e la pulizia.

Il gestore di eventi di Powertools for Lambda AWS

Il gestore di eventi del toolkit Powertools for AWS Lambda fornisce routing, middleware, configurazione CORS, generazione di specifiche OpenAPI, convalida delle richieste, gestione degli errori e altre funzioni utili durante la scrittura di funzioni Lambda richiamate da un endpoint API Gateway (HTTP o REST). L'utilità Event Handler è disponibile per Python TypeScript e/. JavaScript Per ulteriori informazioni, vedete l'API REST di Event Handler nella documentazione di Powertools for AWS Lambda (Python) e l'API REST di Event Handler nella documentazione di 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,
});
```

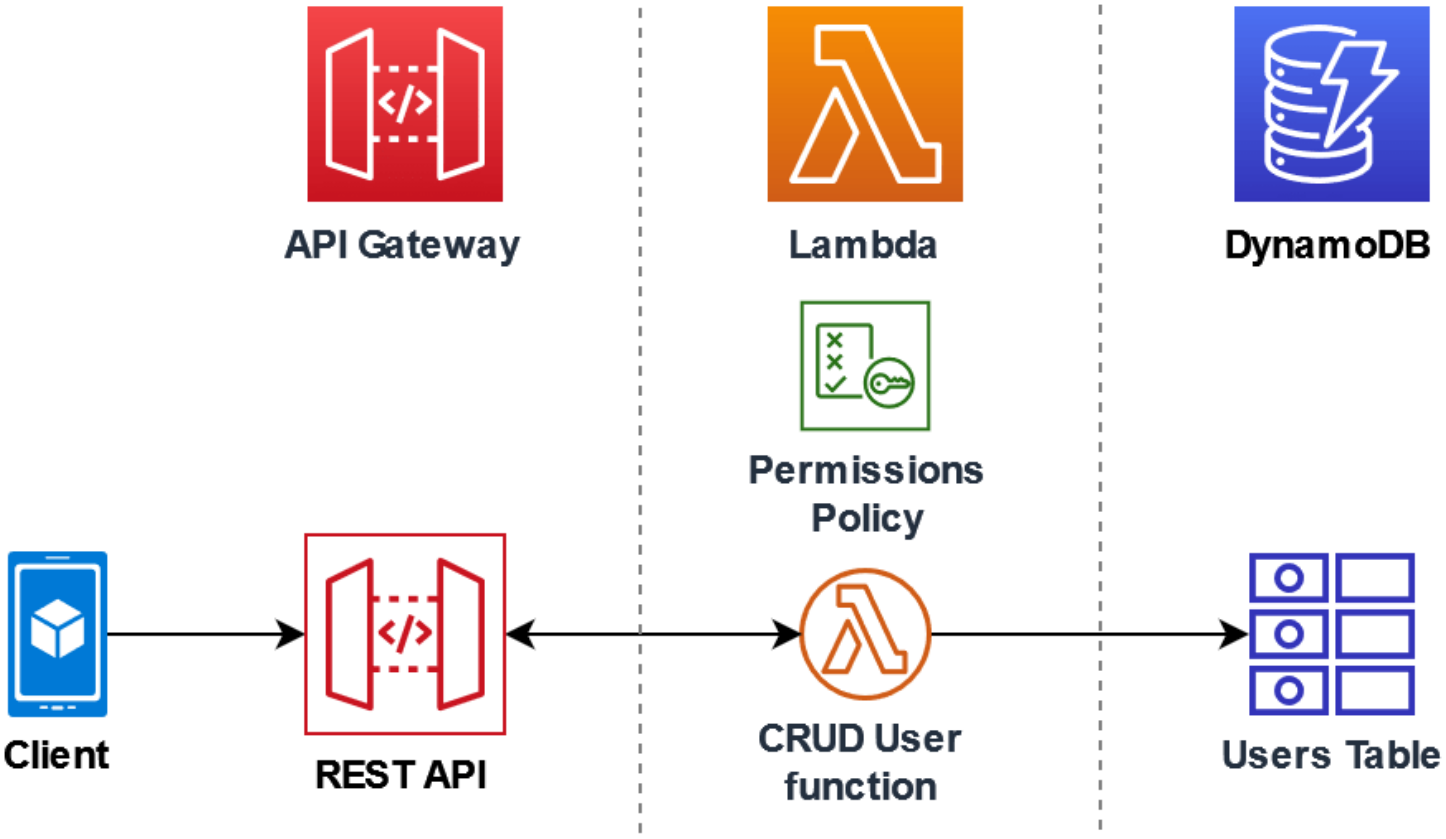
```
const app = new Router({ logger });

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: uso di Lambda con Amazon API Gateway

In questo tutorial, viene creata una REST API tramite la quale viene richiamata una funzione Lambda utilizzando una richiesta HTTP. La funzione Lambda eseguirà operazioni di creazione, aggiornamento ed eliminazione (CRUD) su una tabella DynamoDB. Questa funzione viene fornita qui a scopo dimostrativo, ma imparerai a configurare una REST API di Gateway API in grado di richiamare qualsiasi funzione Lambda.



L'utilizzo di Gateway API fornisce agli utenti un endpoint HTTP sicuro per richiamare la funzione Lambda e può aiutare a gestire grandi volumi di chiamate alla funzione limitando il traffico e convalidando e autorizzando automaticamente le chiamate API. API Gateway fornisce anche controlli di sicurezza flessibili utilizzando AWS Identity and Access Management (IAM) e Amazon Cognito. Ciò è utile nei casi d'uso in cui è richiesta un'autorizzazione preventiva per le chiamate all'applicazione.

> **Tip**
>
> Lambda offre due modi per richiamare una funzione tramite un endpoint HTTP: API Gateway e funzione Lambda. URLs In caso di dubbi su quale sia il metodo migliore per il tuo caso d'uso, consulta the section called “API Gateway vs funzione URLs”.

Per completare questo tutorial, saranno completate le seguenti fasi:

1. Crea e configura una funzione Lambda in Python o Node.js per eseguire operazioni su una tabella DynamoDB.
2. Crea una REST API in Gateway API per connetterti alla funzione Lambda.
3. Crea una tabella DynamoDB e testala con la funzione Lambda nella console.
4. Implementa la tua API e testa la configurazione completa usando curl in un terminale.

Completando queste fasi, imparerai come utilizzare Gateway API per creare un endpoint HTTP in grado di richiamare in modo sicuro una funzione Lambda su qualsiasi scala. Imparerai anche come distribuire la tua API e come testarla nella console e inviando una richiesta HTTP tramite un terminale.

Creazione di una policy di autorizzazione

Prima di poter creare un ruolo di esecuzione per la tua funzione Lambda, devi prima creare una politica di autorizzazioni per concedere alla funzione il permesso di accedere alle risorse richieste. AWS Per questo tutorial, la policy consente a Lambda di eseguire operazioni CRUD su una tabella DynamoDB e scrivere su Amazon Logs. CloudWatch

Come creare la policy

1. Apri la pagina Policies (Policy) nella console IAM.
2. Scegli Crea policy.

3. Scegliere la scheda JSON e quindi incollare la seguente policy personalizzata nell'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. Scegliere Next: Tags (Successivo: Tag).
5. Scegliere Next:Review (Successivo: Rivedi).
6. In Rivedi policy, per Nome della policy inserisci **`lambda-apigateway-policy`**.
7. Scegli Crea policy.

Creazione di un ruolo di esecuzione

Un ruolo di esecuzione è un ruolo AWS Identity and Access Management (IAM) che concede a una funzione Lambda l'autorizzazione all' Servizi AWS accesso e alle risorse. Per consentire alla funzione di eseguire operazioni su una tabella DynamoDB, collega la policy di autorizzazione che hai creato nella fase precedente.

Creazione di un ruolo di esecuzione e collegamento di una policy di autorizzazione personalizzata

1. Aprire la pagina Roles (Ruoli) della console IAM.
2. Scegli Crea ruolo.
3. Per il tipo di entità attendibile, scegli Servizio AWS , quindi per il caso d'uso seleziona Lambda.
4. Scegli Next (Successivo).
5. Nella casella di ricerca delle policy, immettere **`lambda-apigateway-policy`**.
6. Nei risultati della ricerca, seleziona la policy creata (`lambda-apigateway-policy`), quindi scegli Next (Successivo).
7. In Role details (Dettagli del ruolo), per Role name (Nome del ruolo), specifica **`lambda-apigateway-role`**, quindi scegli Create role (Crea ruolo).

Creazione della funzione Lambda

1. Apri la pagina Funzioni della console Lambda e scegli Crea funzione.
2. Scegli Crea da zero.
3. Nel campo Function name (Nome funzione), immettere `LambdaFunctionOverHttps`.
4. Per Runtime, scegli l'ultimo runtime di Node.js o di Python.
5. In Autorizzazioni espandere Modifica ruolo di esecuzione predefinito.
6. Scegli Utilizza un ruolo esistente, quindi seleziona il ruolo **`lambda-apigateway-role`** creato in precedenza.
7. Scegli Crea funzione.
8. Nel riquadro Fonte codice, sostituisci il codice predefinito con il seguente codice Node.js o Python.

Node.js

L'`region`impostazione deve corrispondere al Regione AWS luogo in cui si distribuisce la funzione e si crea la tabella 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
    '''

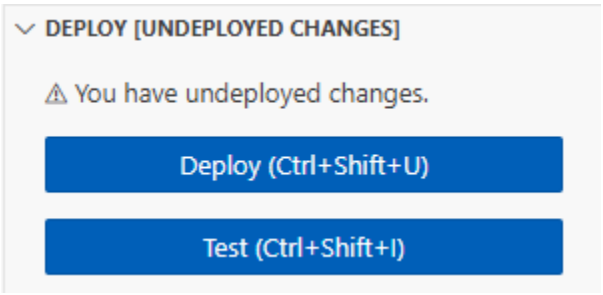
    operation = event['operation']
    payload = event['payload']

    if operation in operations:
        return operations[operation](payload)

    else:
        raise ValueError(f'Unrecognized operation "{operation}"')
```

> **Note**
>
> In questo esempio, il nome della tabella DynamoDB è definito come variabile nel codice della funzione. In un'applicazione reale, la best practice consiste nell'inviare questo parametro come variabile d'ambiente ed evitare di codificare il nome della tabella. Per ulteriori informazioni, consulta Uso AWS Lambda delle variabili di ambiente.

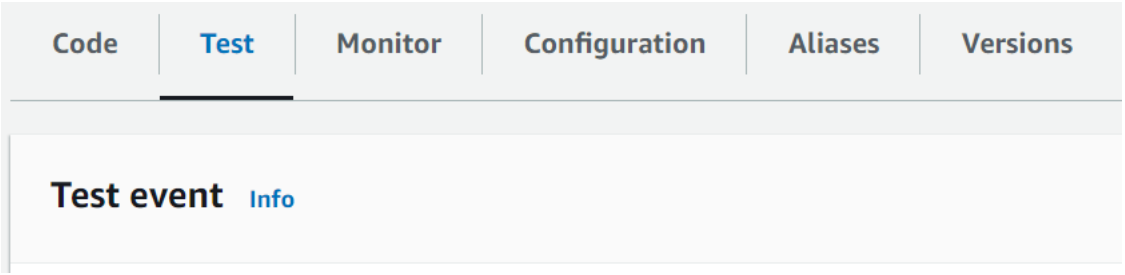
9. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



Test della funzione

Prima di integrare la tua funzione con Gateway API, verifica di aver implementato correttamente la funzione. Utilizza la console Lambda per inviare un evento di test alla tua funzione.

1. Nella pagina della console Lambda della funzione, scegli la scheda Test.



2. Scorri alla sezione JSON dell'evento e sostituisci l’evento predefinito con il seguente. Questo evento corrisponde alla struttura prevista dalla funzione Lambda.

```
{
    "operation": "echo",
    "payload": {
        "somekey1": "somevalue1",
        "somekey2": "somevalue2"
    }
}
```

3. Scegli Test (Esegui test).
4. In Esecuzione funzione: riuscita, espandi Dettagli. Dovrebbe essere visualizzata la seguente risposta:

```
{
  "somekey1": "somevalue1",
  "somekey2": "somevalue2"
```

```
}
```

Creazione di una REST API utilizzando API Gateway

In questa fase, viene creata la REST API di Gateway API che sarà utilizzata per richiamare la funzione Lambda.

Per creare l'API

1. Apri la console API Gateway.
2. Seleziona Create API (Crea API).
3. Nella casella REST API, scegliere Build.
4. In Dettagli API, lasciare selezionata Nuova API e per Nome API, inserire **`DynamoDBOperations`**.
5. Seleziona Create API (Crea API).

Creazione di una risorsa sulla REST API

Per aggiungere un metodo HTTP all'API, è necessario prima creare una risorsa su cui quel metodo possa operare. Qui viene creata la risorsa per gestire la tabella DynamoDB.

Per creare la risorsa

1. Nella console API Gateway, nella pagina Risorse dell'API, scegliere Crea risorsa.
2. In Dettagli risorsa, per Nome risorsa, inserire **`DynamoDBManager`**.
3. Scegliere Create Resource (Crea risorsa).

Creazione di un metodo HTTP POST

In questa fase, viene creato un metodo (`POST`) per la risorsa `DynamoDBManager`. Il metodo `POST` viene collegato alla funzione Lambda in modo che quando il metodo riceve una richiesta HTTP, Gateway API richiama la funzione Lambda.

> **Note**
>
> Ai fini di questo tutorial, viene utilizzato un metodo HTTP (`POST`) per richiamare una singola funzione Lambda che esegue tutte le operazioni sulla tabella DynamoDB. In un'applicazione

reale, la best practice consiste nell'utilizzare una funzione Lambda e un metodo HTTP diversi per ogni operazione. Per ulteriori informazioni, consulta Il monolite Lambda in Serverless Land.

Creazione del metodo POST

1. Nella pagina Risorse dell'API, assicurarsi che la risorsa `/DynamoDBManager` sia evidenziata. Quindi, nel riquadro Metodi, scegliere Crea metodo.
2. Per Metodo HTTP scegliere POST.
3. Per Tipo di integrazione lasciare selezionato Funzione Lambda.
4. Per la funzione Lambda, scegliere nome della risorsa Amazon (ARN) per la funzione (`LambdaFunctionOverHttps`).
5. Scegli Crea metodo.

Creazione di una tabella DynamoDB

Crea una tabella DynamoDB vuota su cui la funzione Lambda eseguirà le operazioni CRUD.

Creare la tabella DynamoDB

1. Aprire la pagina Tables (Tabelle) della console DynamoDB.
2. Scegliere Create table (Crea tabella).
3. In Table details (Dettagli tabella), effettuare le seguenti operazioni:
    1. Nel campo Table name (Nome tabella) immetti **`lambda-apigateway`**.
    2. Per Partition key (Chiave di partizione), immettere **`id`** e mantenere il tipo di dati impostato come String (Stringa).
4. In Table settings (Impostazioni tabella), mantieni le impostazioni predefinite.
5. Scegliere Create table (Crea tabella).

Test dell'integrazione di Gateway API, Lambda e DynamoDB

A questo punto sei pronto per testare l'integrazione del metodo API di Gateway API con la funzione Lambda e la tabella DynamoDB. Utilizzando la console di Gateway API, invii le richieste direttamente al metodo `POST` utilizzando la funzione di test della console. In questo passaggio, viene utilizzata

prima un'operazione `create` per aggiungere un nuovo elemento alla tabella DynamoDB, quindi viene utilizzata un'operazione `update` per modificare l'elemento.

Test 1: creazione di un nuovo elemento nella tabella DynamoDB

1. Nella console di Gateway API, scegli l'API (`DynamoDBOperations`).
2. Scegli il metodo POST sotto la risorsa `DynamoDBManager`.
3. Seleziona la scheda Test. Potrebbe essere necessario scegliere il pulsante freccia destra per visualizzare la scheda.
4. In Metodo di prova, lasciare vuote le stringhe di query e le intestazioni. Per Corpo della richiesta, incollare il file JSON seguente:

```
{
  "operation": "create",
  "payload": {
    "Item": {
      "id": "1234ABCD",
      "number": 5
    }
  }
}
```

5. Scegli Test (Esegui test).

   I risultati visualizzati al termine del test devono mostrare lo stato `200`. Questo codice di stato indica che l'operazione `create` è riuscita.

   Come verifica, controlla che la tabella DynamoDB contenga il nuovo elemento.

6. Apri la pagina Tables (Tabelle) della console DynamoDB e scegli la tabella `lambda-apigateway`.
7. Scegli Explore table items (Esplora elementi della tabella). Nel riquadro Items returned (Elementi restituiti), dovresti vedere un elemento con l'id `1234ABCD` e il numero `5`. Esempio:



Test 2: aggiornamento dell'elemento nella tabella DynamoDB

1. Nella Console API Gateway, tornare alla scheda Test del metodo POST.
2. In Metodo di prova, lasciare vuote le stringhe di query e le intestazioni. Per Corpo della richiesta, incollare il file JSON seguente:

```
{
    "operation": "update",
    "payload": {
        "Key": {
            "id": "1234ABCD"
        },
        "UpdateExpression": "SET #num = :newNum",
        "ExpressionAttributeNames": {
            "#num": "number"
        },
        "ExpressionAttributeValues": {
            ":newNum": 10
        }
    }
}
```

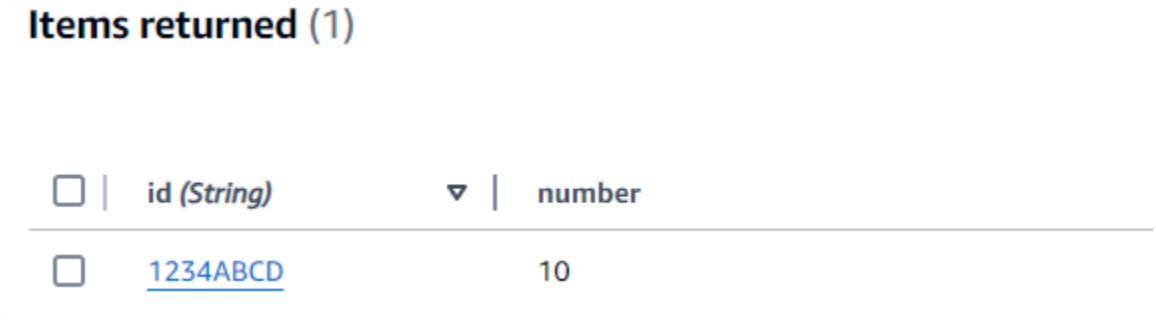
3. Scegli Test (Esegui test).

   I risultati visualizzati al termine del test devono mostrare lo stato `200`. Questo codice di stato indica che l'operazione `update` è riuscita.

   Per confermare, controlla che l'elemento nella tua tabella DynamoDB sia stato modificato.

4. Apri la pagina Tables (Tabelle) della console DynamoDB e scegli la tabella `lambda-apigateway`.

5. Scegli Explore table items (Esplora elementi della tabella). Nel riquadro Items returned (Elementi restituiti), dovresti vedere un elemento con l'id `1234ABCD` e il numero `10`.



Distribuzione dell'API

Per consentire al client di chiamare l'API, è necessario creare una implementazione e una fase associata. Una fase rappresenta un'istantanea dell'API, inclusi i suoi metodi e le sue integrazioni.

Per distribuire l'API

1. Apri la APIspagina della console API Gateway e scegli l'`DynamoDBOperations`API.
2. Nella pagina Risorse per la tua API, scegliere Distribuzione dell'API.
3. Per Fase, scegliere *Nuova fase* quindi per Nome fase specificare **`test`**.
4. Seleziona Implementa.
5. Nel riquadro Editor fase (Editor fasi) del test, copia l'URL di richiamo. Questo verrà utilizzato nella fase successiva per richiamare la tua funzione utilizzando una richiesta HTTP.

Uso di curl per richiamare la funzione tramite le richieste HTTP

A questo punto è possibile richiamare la funzione Lambda inviando una richiesta HTTP all'API. In questo passaggio, verrà creato un nuovo elemento nella tabella DynamoDB e quindi verranno eseguite le operazioni di lettura, aggiornamento ed eliminazione su tale elemento.

Per creare un elemento nella tabella DynamoDB utilizzando curl

1. Apri un terminale o un prompt dei comandi sul computer locale ed esegui il comando `curl` seguente utilizzando l'URL di invocazione che hai copiato nel passaggio precedente. Questo comando utilizza le seguenti opzioni:

- `-H`: aggiunge un'intestazione personalizzata alla richiesta. Qui specifica il tipo di contenuto come JSON.
- `-d`: invia i dati nel corpo della richiesta. Per impostazione predefinita, questa opzione utilizza un metodo HTTP POST.

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}}}'
```

Se l'operazione ha avuto esito positivo, dovrebbe essere restituita una risposta con un codice di stato HTTP di 200.

2. È inoltre possibile utilizzare la console DynamoDB per verificare che il nuovo elemento sia nella tabella effettuando le operazioni seguenti:

   1. Apri la pagina Tables (Tabelle) della console DynamoDB e scegli la tabella `lambda-apigateway`.
   2. Scegli Explore table items (Esplora elementi della tabella). Nel riquadro Items returned (Elementi restituiti), dovresti vedere un elemento con l'id `5678EFGH` e il numero `15`.

Per leggere l'elemento nella tabella DynamoDB utilizzando curl

- Nel terminale o nel prompt dei comandi, esegui il comando `curl` seguente per leggere il valore dell'elemento appena creato. Utilizzo dei propri URL di richiamo

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\"}}}'
```

Dovresti vedere un output simile a uno dei seguenti a seconda che tu abbia scelto il codice della funzione Node.js o 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}}
```

Per aggiornare l'elemento nella tabella DynamoDB utilizzando curl

1. Nel terminale o nel prompt dei comanti, esegui il comando `curl` seguente per aggiornare l'elemento appena creato modificando il valore `number`. Utilizzo dei propri URL di richiamo

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. Per verificare che il valore di `number` per l'elemento sia stato aggiornato, esegui un altro comando di lettura:

   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\"}}}'
   ```

Per eliminare l'elemento nella tabella DynamoDB utilizzando curl

1. Nel terminale o nel prompt dei comandi, esegui il comando `curl` seguente per eliminare l'elemento appena creato. Utilizzo dei propri URL di richiamo

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. Verifica che l'operazione di eliminazione è andata a buon fine. Nel riquadro Elementi restituiti della pagina Esplora elementi della console DynamoDB, verifica che l'elemento con id `5678EFGH` non sia più presente nella tabella.

Pulizia delle risorse (facoltativo)

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili a tuo Account AWS carico.

Per eliminare la funzione Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **`confirm`** nel campo di immissione del testo, quindi scegli Elimina.

Come eliminare il ruolo di esecuzione

1. Aprire la pagina Ruoli della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

Per eliminare l'API

1. Apri la APIs pagina della console API Gateway.
2. Selezionare l'API creata.
3. Scegliere Actions (Operazioni), Delete (Elimina).
4. Scegliere Delete (Elimina).

Per eliminare la tabella DynamoDB

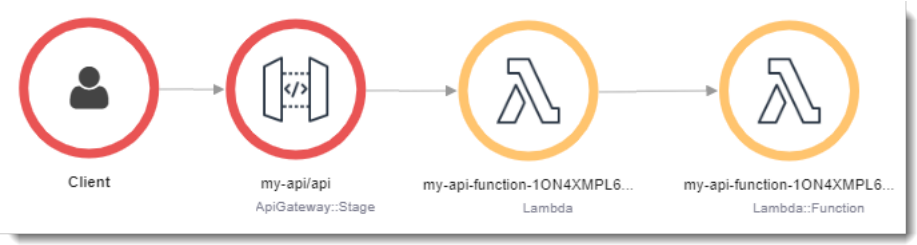
1. Aprire la pagina Tables (Tabelle) della console DynamoDB.
2. Selezionare la tabella creata.
3. Scegliere Delete (Elimina).
4. Immettere **`delete`** nella casella di testo.
5. Seleziona Delete Table (Elimina tabella).

Gestione degli errori con un'API di API Gateway

API Gateway considera tutti gli errori di invocazione e di funzione come errori interni. Se l'API Lambda rifiuta la richiesta di invocazione, API Gateway restituisce un codice di errore 500. Se la funzione viene eseguita ma restituisce un errore o una risposta nel formato errato, API Gateway restituisce il codice 502. In entrambi i casi, il corpo della risposta da API Gateway è `{"message": "Internal server error"}`.

> **Note**
>
> API Gateway non riprova le invocazioni Lambda. Se Lambda restituisce un errore, API Gateway restituisce una risposta di errore al client.

Nell'esempio seguente viene illustrata una mappa di tracciamento X-Ray per una richiesta che ha generato un errore di funzione e un codice 502 da API Gateway. Il client riceve il messaggio di errore generico.

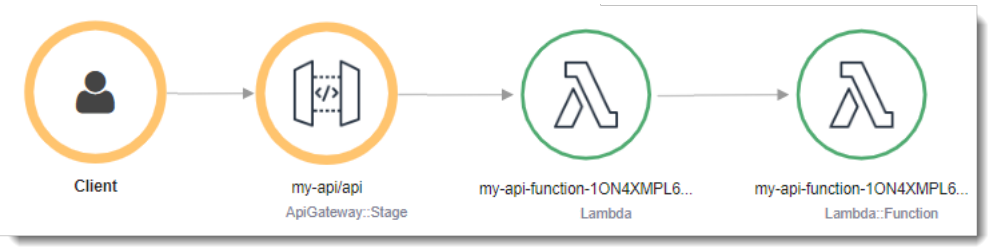


Per personalizzare la risposta di errore, è necessario rilevare gli errori nel codice e formattare una risposta nel formato richiesto.

Example index.mjs: errore di formattazione

```
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 converte questa risposta in un errore HTTP con un codice di stato personalizzato e un corpo. Nella mappa di tracciamento, il nodo della funzione è verde perché ha gestito l'errore.



Selezionare un metodo per richiamare la funzione Lambda tramite una richiesta HTTP

Numerosi casi d'uso comuni per Lambda implicano l'invocazione della funzione tramite una richiesta HTTP. Ad esempio, potrebbe essere preferibile che un'applicazione Web invochi la funzione tramite una richiesta del browser. Le funzioni Lambda possono essere utilizzate anche per creare REST

completo APIs, gestire le interazioni degli utenti da app mobili, elaborare dati da servizi esterni tramite chiamate HTTP o creare webhook personalizzati.

Le sezioni seguenti spiegano quali sono le tue scelte per richiamare Lambda tramite HTTP e forniscono informazioni per aiutarti a prendere la decisione giusta per il tuo caso d'uso particolare.

Quali sono le tue scelte quando selezioni un metodo di invocazione HTTP?

Lambda offre due metodi principali per richiamare una funzione utilizzando una richiesta HTTP: funzione e API URLs Gateway. Le differenze tra queste due opzioni sono le seguenti:

- La funzione Lambda URLs fornisce un endpoint HTTP semplice e diretto per una funzione Lambda. Sono ottimizzati per semplicità ed economicità e forniscono il percorso più veloce per esporre una funzione Lambda tramite HTTP.
- API Gateway è un servizio più avanzato per la creazione di funzionalità complete APIs. API Gateway è ottimizzato per la creazione e la gestione di produzioni APIs su larga scala e fornisce strumenti completi per la sicurezza, il monitoraggio e la gestione del traffico.

Suggerimenti se conosci già le tue esigenze

Se hai già le idee chiare sulle tue esigenze, ecco i nostri suggerimenti di base:

Consigliamo la funzionalità URLs per applicazioni semplici o per la prototipazione in cui sono necessari solo metodi di autenticazione e request/response gestione di base e dove si desidera ridurre al minimo costi e complessità.

API Gateway è la scelta migliore per le applicazioni di produzione su larga scala o per i casi in cui sono necessarie funzionalità più avanzate come il supporto OpenAPI Description, una scelta di opzioni di autenticazione, nomi di dominio personalizzati o una request/response gestione avanzata che include throttling, caching e trasformazione. request/response

Cosa considerare quando si seleziona un metodo per richiamare la funzione Lambda

Nella scelta tra funzione URLs e API Gateway, è necessario considerare i seguenti fattori:

- Le tue esigenze di autenticazione, ad esempio se richiedi OAuth o Amazon Cognito per autenticare gli utenti
- I tuoi requisiti di scalabilità e la complessità dell'API che desideri implementare

- Se hai bisogno di funzionalità avanzate come la convalida e la formattazione delle richieste request/response
- I tuoi requisiti di monitoraggio
- I tuoi obiettivi di costo

Comprendendo questi fattori, è possibile selezionare l'opzione che meglio bilancia i requisiti di sicurezza, complessità e costi.

Le informazioni seguente riepilogano le differenze principali tra le due opzioni.

Autenticazione

- La funzione URLs fornisce opzioni di autenticazione di base tramite AWS Identity and Access Management (IAM). Puoi configurare gli endpoint in modo che siano pubblici (senza autenticazione) o che richiedano l'autenticazione IAM. Con l'autenticazione IAM, puoi utilizzare AWS credenziali standard o ruoli IAM per controllare l'accesso. Sebbene sia semplice da configurare, questo approccio offre opzioni limitate rispetto ad altri metodi di autenticazione.
- API Gateway fornisce l'accesso a una gamma più completa di opzioni di autenticazione. Oltre all'autenticazione IAM, puoi utilizzare gli autorizzatori Lambda (logica di autenticazione personalizzata), i pool di utenti di Amazon Cognito e i flussi .0. OAuth2 Questa flessibilità consente di implementare schemi di autenticazione complessi, inclusi provider di autenticazione di terze parti, autenticazione basata su token e autenticazione a più fattori.

Gestione di richieste/risposte

- La funzione URLs fornisce la gestione di base delle richieste e delle risposte HTTP. Supportano i metodi HTTP standard e includono il supporto integrato per CORS (cross-origin resource sharing). Sebbene siano in grado di gestire i payload JSON e i parametri di query in modo naturale, non offrono funzionalità di trasformazione o convalida delle richieste. La gestione delle risposte è altrettanto semplice: il client riceve la risposta dalla funzione Lambda esattamente come la restituisce Lambda.
- API Gateway offre sofisticate funzionalità di gestione delle richieste e delle risposte. È possibile definire validatori di richiesta, trasformare richieste e risposte utilizzando modelli di mappatura, impostare request/response intestazioni e implementare la memorizzazione nella cache delle risposte. API Gateway supporta anche payload binari e nomi di dominio personalizzati e può modificare le risposte prima che raggiungano il client. È possibile configurare modelli per la request/response convalida e la trasformazione utilizzando JSON Schema.

Dimensionamento

- Le funzioni si URLs adattano direttamente ai limiti di concorrenza della funzione Lambda e gestisci i picchi di traffico scalando la funzione fino al limite di concorrenza massimo configurato. Una volta raggiunto tale limite, Lambda risponde alle richieste aggiuntive con risposte HTTP 429. Non esiste un meccanismo di accodamento integrato, pertanto la gestione della scalabilità dipende interamente dalla configurazione della funzione Lambda. Per impostazione predefinita, le funzioni Lambda hanno un limite di 1.000 esecuzioni simultanee per. Regione AWS
- API Gateway offre funzionalità di scalabilità aggiuntive oltre alla scalabilità propria di Lambda. Include controlli integrati per l'accodamento e la limitazione delle richieste che consentono di gestire i picchi di traffico con maggiore efficienza. Per impostazione predefinita, API Gateway è in grado di gestire fino a 10.000 richieste al secondo per regione, con una capacità di espansione di 5.000 richieste al secondo. Fornisce inoltre strumenti per limitare le richieste a diversi livelli (API, fase o metodo) per proteggere il backend.

Monitoraggio

- La funzione URLs offre un monitoraggio di base tramite i CloudWatch parametri di Amazon, tra cui il numero di richieste, la latenza e i tassi di errore. Puoi accedere a parametri e log Lambda standard, che mostrano le richieste non elaborate che arrivano alla tua funzione. Sebbene ciò fornisca una visibilità operativa essenziale, i parametri si concentrano principalmente sull'esecuzione delle funzioni.
- API Gateway offre funzionalità di monitoraggio complete, tra cui parametri dettagliati, opzioni di registrazione e tracciamento. Puoi monitorare le chiamate API, la latenza, i tassi di errore e i tassi di cache hit/miss tramite. CloudWatch API Gateway si integra anche con AWS X-Ray il tracciamento distribuito e fornisce formati di registrazione personalizzabili.

Costo

- Le funzioni URLs seguono il modello di prezzo standard Lambda: paghi solo per le chiamate di funzione e il tempo di calcolo. Non sono previsti costi aggiuntivi per l'endpoint URL stesso. Ciò lo rende una scelta conveniente per applicazioni semplici APIs o a basso traffico se non sono necessarie le funzionalità aggiuntive di API Gateway.
- API Gateway offre un piano gratuito che include un milione di chiamate API ricevute per REST APIs e un milione di chiamate API ricevute per HTTP APIs. Successivamente, API Gateway addebita i costi per le chiamate API, il trasferimento dei dati e la memorizzazione nella cache (se

abilitata). Consulta la pagina dei prezzi di API Gateway per conoscere i costi relativi al tuo caso d'uso.

Altre funzionalità

- URLsLe funzioni sono progettate per la semplicità e l'integrazione diretta con Lambda. Supportano endpoint HTTP e HTTPS, offrono supporto CORS integrato e forniscono endpoint dual-stack (e). IPv4 IPv6 Sebbene non dispongano di funzionalità avanzate, eccellono negli scenari in cui è necessario un modo rapido e diretto per esporre le funzioni Lambda tramite HTTP.
- API Gateway include numerose funzionalità aggiuntive come il controllo delle versioni delle API, la gestione delle fasi, le chiavi API per i piani di utilizzo, la documentazione delle API tramite Swagger/OpenAPI, l'accesso WebSocket APIs privato all'interno di APIs un VPC e l'integrazione WAF per una maggiore sicurezza. Supporta anche implementazioni Canary, integrazioni fittizie per i test e l'integrazione con altri sistemi oltre a Lambda. Servizi AWS

Selezionare un metodo per richiamare la funzione Lambda

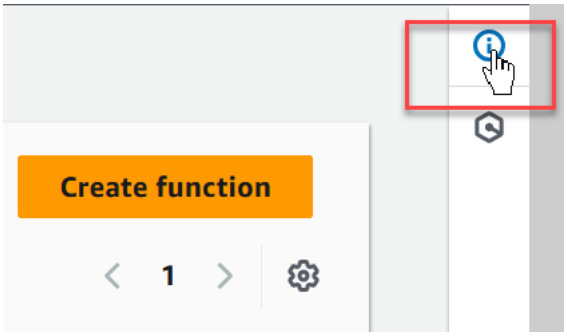
Ora che hai letto i criteri per la selezione tra la funzione Lambda URLs e l'API Gateway e le principali differenze tra di essi, puoi selezionare l'opzione più adatta alle tue esigenze e utilizzare le seguenti risorse per iniziare a utilizzarla.

Function URLs

Inizia a usare la funzione URLs con le seguenti risorse

- Segui il tutorial Creazione di una funzione Lambda con la funzione URL
- Scopri di più sulla funzione URLs nel the section called “Funzione URLs” capitolo di questa guida
- Prova il tutorial guidato dalla console Creare una semplice app Web effettuando le seguenti operazioni:

1. Apri la pagina Funzioni della console Lambda.
2. Apri il pannello di aiuto scegliendo l'icona nell'angolo in alto a destra della schermata.



3. Seleziona Tutorial.
4. In Crea una semplice app Web, scegli Avvia tutorial.

API Gateway

Iniziare a usare Lambda e API Gateway con le seguenti risorse

- Segui il tutorial Utilizzo di Lambda con API Gateway per creare una REST API integrata con una funzione Lambda di backend.
- Scopri di più sui diversi tipi di API offerti da API Gateway nelle seguenti sezioni della Guida per gli sviluppatori di Gateway Amazon API:
  - API Gateway REST APIs
  - Gateway API HTTP APIs
  - API Gateway WebSocket APIs
- Prova uno o più esempi nella sezione Tutorial e workshop della Guida per gli sviluppatori di Gateway Amazon API.

Utilizzo AWS Lambda con AWS Infrastructure Composer

AWS Infrastructure Composer è un generatore visivo per la progettazione di applicazioni moderne su. AWS Progetta l'architettura della tua applicazione trascinandola, raggruppandola e connettendola in un'area di disegno visiva. Servizi AWS Infrastructure Composer crea modelli di infrastructure as code (IaC) a partire dal tuo progetto che puoi implementare utilizzando AWS SAM oppure CloudFormation.

Esportazione di una funzione Lambda in Infrastructure Composer

Per iniziare a utilizzare Infrastructure Composer, crea un nuovo progetto basato sulla configurazione di una funzione Lambda esistente utilizzando la console Lambda. Per esportare la configurazione e il codice della funzione in Infrastructure Composer per creare un nuovo progetto, procedi come segue:

1. Aprire la pagina Funzioni della console Lambda.
2. Seleziona la funzione che desideri utilizzare come base per il tuo progetto Infrastructure Composer.
3. Nel riquadro Panoramica della funzione, scegli Esporta in Infrastructure Composer.

   Per esportare la configurazione e il codice della funzione in Infrastructure Composer, Lambda crea un bucket Amazon S3 nel tuo account in cui archiviare temporaneamente questi dati.
4. Nella finestra di dialogo, scegli Conferma e crea progetto per accettare il nome predefinito per questo bucket ed esportare la configurazione e il codice della funzione in Infrastructure Composer.
5. (Facoltativo) Per scegliere un altro nome per il bucket Amazon S3 creato da Lambda, immetti un nuovo nome e scegli Conferma e crea progetto. I nomi dei bucket Amazon S3 devono essere univoci a livello globale e seguire le regole di denominazione dei bucket.
6. Per salvare i file di progetto e funzione in Infrastructure Composer, attiva la modalità di sincronizzazione locale.

> **Note**
>
> Se hai già utilizzato la funzionalità Esporta in Strumento per la creazione di applicazioni e hai creato un bucket Amazon S3 utilizzando il nome predefinito, Lambda può riutilizzare questo bucket, se esiste ancora. Accetta il nome predefinito del bucket nella finestra di dialogo per riutilizzare il bucket esistente.

Configurazione del bucket di trasferimento Amazon S3

Il bucket Amazon S3 creato da Lambda per trasferire la configurazione della funzione crittografa automaticamente gli oggetti utilizzando lo standard di crittografia AES 256. Lambda configura inoltre il bucket in modo che utilizzi la condizione di proprietario del bucket per garantire che solo il tuo Account AWS sia in grado di aggiungere oggetti al bucket.

Lambda configura il bucket per eliminare automaticamente gli oggetti 10 giorni dopo il caricamento. Tuttavia, Lambda non elimina automaticamente il bucket stesso. Per eliminare il bucket dal tuo Account AWS, segui le istruzioni in Eliminare un bucket. Il nome predefinito del bucket utilizza il prefisso`lambdasam`, una stringa alfanumerica di 10 cifre, e la funzione in cui hai creato la funzione in: Regione AWS

```
lambdasam-06f22da95b-us-east-1
```

Per evitare costi aggiuntivi Account AWS, ti consigliamo di eliminare il bucket Amazon S3 non appena hai finito di esportare la funzione in Infrastructure Composer.

Si applicano i prezzi standard di Amazon S3.

Autorizzazioni richieste

Per utilizzare l'integrazione Lambda con la funzionalità Infrastructure Composer, sono necessarie determinate autorizzazioni per scaricare un AWS SAM modello e scrivere la configurazione della funzione su Amazon S3.

Per scaricare un AWS SAM modello, devi disporre dell'autorizzazione per utilizzare le seguenti azioni API:

- GetPolicy
- sono: GetPolicyVersion
- sono: GetRole
- sono: GetRolePolicy
- sono: ListAttachedRolePolicies
- sono: ListRolePolicies
- sono: ListRoles

Puoi concedere l'autorizzazione a utilizzare tutte queste azioni aggiungendo la policy `AWSLambda_ReadOnlyAccess` AWS gestita al tuo ruolo utente IAM.

Affinché Lambda scriva la configurazione della tua funzione su Amazon S3, devi disporre dell'autorizzazione a utilizzare le seguenti operazioni dell'API:

- S3: PutObject

- S3: CreateBucket
- S3: PutBucketEncryption
- S3: PutBucketLifecycleConfiguration

Se non riesci a esportare la configurazione della tua funzione in Infrastructure Composer, verifica che il tuo account disponga delle autorizzazioni necessarie per queste operazioni. Se disponi delle autorizzazioni richieste ma non riesci comunque a esportare la configurazione della funzione, verifica se ne sono presenti policy basate sulle risorse che potrebbero limitare l'accesso ad Amazon S3.

Altre risorse

Per un tutorial più dettagliato su come progettare un'applicazione serverless in Infrastructure Composer basata su una funzione Lambda esistente, consulta the section called “Infrastructure as code (IaC)”.

Per utilizzare Infrastructure Composer e AWS SAM progettare e distribuire un'applicazione serverless completa utilizzando Lambda, puoi anche seguire il AWS Infrastructure Composer tutorial nel Serverless Patterns Workshop.AWS

Uso di AWS Lambda con CloudFormation

In un modello AWS CloudFormation, puoi specificare una funzione Lambda come target di una risorsa personalizzata. Utilizza risorse personalizzate per elaborare parametri, recuperare valori di configurazioni o chiamare altri Servizi AWS durante gli eventi del ciclo di vita dello stack.

L'esempio seguente invoca una funzione definita in un altro punto del modello.

Example – Definizione di risorse personalizzate

```
Resources:
  primerinvoke:
    Type: AWS::CloudFormation::CustomResource
    Version: "1.0"
    Properties:
      ServiceToken: !GetAtt primer.Arn
      FunctionName: !Ref randomerror
```

Il token del servizio è l'Amazon Resource Name (ARN) della funzione che CloudFormation invoca quando crei, aggiorni o elimini lo stack. Puoi anche includere proprietà aggiuntive come `FunctionName`, che CloudFormation passa alla funzione senza modifiche.

CloudFormation richiama la funzione Lambda in modo asincrono con un evento che include un URL di callback.

Example –CloudFormation Evento messaggio di

```
{
    "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"
    }
}
```

Da questa funzione dipende la risposta all'URL di callback che indica un esito positivo o negativo. Per la sintassi di risposta completa, consulta Oggetti di risposta delle risorse personalizzate.

Example –CloudFormation Risposta della risorsa personalizzata

```
{
    "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 offre una libreria chiamata `cfn-response` che gestisce l'invio della risposta. Se definisci la funzione all'interno di un modello, puoi richiamare la libreria per nome. CloudFormation quindi aggiunge la libreria al pacchetto di distribuzione che crea per la funzione.

Se alla funzione utilizzata da una risorsa personalizzata è collegata un'interfaccia di rete elastica, aggiungi le seguenti risorse alla policy VPC, dove **`region`** è la regione in cui si trova la funzione senza i trattini. Ad esempio, `us-east-1` è `useast1`. Ciò consentirà alla risorsa personalizzata di rispondere all'URL di callback che restituisce un segnale allo stack CloudFormation.

```
arn:aws:s3:::cloudformation-custom-resource-response-region",
"arn:aws:s3:::cloudformation-custom-resource-response-region/*",
```

La seguente funzione di esempio invoca una seconda funzione. Se la chiamata ha esito positivo, la funzione invia una risposta positiva a CloudFormation e l'aggiornamento dello stack prosegue. Il modello utilizza il tipo di risorsa AWS::Serverless::Function fornito da AWS Serverless Application Model.

Example : funzione relativa alle risorse personalizzate

```
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
```

Se la funzione che la risorsa personalizzata invoca non è definita in un modello, è possibile ottenere il codice sorgente di `cfn-response` dal modulo cfn-response nella Guida per l'utente di AWS CloudFormation.

Per ulteriori informazioni sulle risorse personalizzate, consulta Risorse personalizzate nella Guida per l'utente di AWS CloudFormation.

Elaborare gli eventi Amazon DocumentDB con Lambda

Per elaborare gli eventi in un flusso di modifica di Amazon DocumentDB (compatibile con MongoDB), puoi utilizzare una funzione Lambda configurando un cluster Amazon DocumentDB come origine degli eventi. Successivamente puoi automatizzare i carichi di lavoro basati sugli eventi invocando la funzione Lambda con il cluster Amazon DocumentDB ogni volta che i dati cambiano.

> **Note**
>
> Lambda supporta solo le versioni 4.0 e 5.0 di Amazon DocumentDB. Lambda non supporta la versione 3.6.
> Inoltre, per gli strumenti di mappatura dell'origine degli eventi, Lambda supporta solo cluster basati su istanze e cluster regionali. Lambda non supporta cluster elastici o cluster globali. Questa limitazione non si applica quando si utilizza Lambda come client per connettersi ad Amazon DocumentDB. Lambda può connettersi a tutti i tipi di cluster per eseguire operazioni CRUD.

Lambda elabora gli eventi di Amazon DocumentDB nei flussi di modifica in sequenza secondo l'ordine in cui arrivano. Per questo motivo, la tua funzione può gestire solo una invocazione simultanea da Amazon DocumentDB alla volta. Per monitorare la tua funzione, puoi tenere traccia dei relativi parametri di simultaneità.

> **Warning**
>
> Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center. AWS

Argomenti

- Esempio di evento Amazon DocumentDB
- Prerequisiti e autorizzazioni
- Configurare la sicurezza della rete

- Creazione di una mappatura dell'origine degli eventi Amazon DocumentDB (console)
- Creazione di una mappatura dell'origine degli eventi Amazon DocumentDB (SDK o CLI)
- Posizioni di partenza di polling e flussi
- Monitoraggio dell'origine degli eventi di Amazon DocumentDB
- Tutorial: Utilizzo AWS Lambda con Amazon DocumentDB Streams

Esempio di evento 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"
}
```

Per ulteriori informazioni sugli eventi in questo esempio e sulle loro forme, consulta la sezione Eventi di modifica sul sito Web della documentazione di MongoDB.

Prerequisiti e autorizzazioni

Prima di utilizzare Amazon DocumentDB come origine degli eventi della funzione Lambda, è necessario tenere a mente i seguenti prerequisiti. Devi:

- Disponi di un cluster Amazon DocumentDB esistente nella stessa Account AWS Regione AWS funzione. Se non hai un cluster esistente, puoi crearlo seguendo i passaggi riportati nella sezione Nozioni di base su Amazon DocumentDB nella Guida per gli sviluppatori di Amazon DocumentDB. In alternativa, la prima serie di passaggi riportati in Tutorial: Utilizzo AWS Lambda con Amazon DocumentDB Streams ti guideranno nella creazione di un cluster Amazon DocumentDB con tutti i prerequisiti necessari.
- Concedere a Lambda l'accesso alle risorse di Amazon Virtual Private Cloud (Amazon VPC) associate al cluster Amazon DocumentDB. Per ulteriori informazioni, consulta Configurare la sicurezza della rete.
- Abilitare TLS sul cluster Amazon DocumentDB. Questa è l'impostazione predefinita. Se TLS è disabilitato, Lambda non è in grado di comunicare con il cluster.
- È necessario attivare i flussi di modifica sul cluster Amazon DocumentDB. Per ulteriori informazioni, consulta la sezione Utilizzo dei flussi di modifica di Amazon DocumentDB nella Guida per gli sviluppatori di Amazon DocumentDB.
- Fornire a Lambda le credenziali per accedere al cluster Amazon DocumentDB. Quando configuri l'origine degli eventi, fornisci la chiave Gestione dei segreti AWS che contiene i dettagli di autenticazione (nome utente e password) necessari per accedere al cluster. Per fornire questa chiave durante la configurazione, esegui una delle seguenti operazioni:
  - Se per la configurazione stai utilizzando la console Lambda, fornisci la chiave nel campo Chiave di Secrets manager.
  - Se utilizzi AWS Command Line Interface (AWS CLI) per la configurazione, fornisci questa chiave nell'`source-access-configuration`opzione. È possibile includere questa opzione sia con il comando `create-event-source-mapping` sia con il comando `update-event-source-mapping`. Esempio:

```
aws lambda create-event-source-mapping \
    ...
    --source-access-configurations
 '[{"Type":"BASIC_AUTH","URI":"arn:aws:secretsmanager:us-
west-2:123456789012:secret:DocDBSecret-AbC4E6"}]' \
    ...
```

- Concedere a Lambda le autorizzazioni per gestire le risorse correlate al flusso di Amazon DocumentDB. Aggiungi le seguenti autorizzazioni al ruolo di esecuzione della tua funzione:
  - RDS: Descrivi DBClusters
  - RDS: descrivere i parametri DBCluster
  - RDS: Descrivi DBSubnet i gruppi
  - ec2: CreateNetworkInterface
  - ec2: DescribeNetworkInterfaces
  - ec2: DescribeVpcs
  - ec2: DeleteNetworkInterface
  - ec2: DescribeSubnets
  - ec2: DescribeSecurityGroups
  - kms:Decrypt
  - gestore dei segreti: GetSecretValue
- È necessario mantenere la dimensione degli eventi del flusso di modifica di Amazon DocumentDB che invii a Lambda inferiore a 6 MB. Lambda supporta payload di dimensioni massime di 6 MB. Se il flusso di modifica tenta di inviare a Lambda un evento di dimensioni superiori a 6 MB, Lambda tralascia il messaggio ed emette il parametro `OversizedRecordCount`. Lambda emette tutte i parametri sulla base del miglior tentativo.

> **Note**
>
> Sebbene le funzioni Lambda abbiano di solito un timeout massimo di 15 minuti, le mappature dell'origine degli eventi per Amazon MSK, Apache Kafka autogestito, Amazon DocumentDB e Amazon MQ per ActiveMQ e RabbitMQ supportano soltanto funzioni con timeout massimi di 14 minuti. Questa limitazione garantisce che lo strumento di mappatura dell'origine degli eventi possa gestire correttamente errori di funzioni e nuovi tentativi.

Configurare la sicurezza della rete

Per concedere a Lambda l'accesso completo ad Amazon DocumentDB tramite lo strumento di mappatura dell'origine degli eventi, il cluster deve utilizzare un endpoint pubblico (indirizzo IP pubblico) oppure devi fornire l'accesso all'Amazon VPC in cui hai creato il cluster.

Quando usi Amazon DocumentDB con Lambda, crea endpoint VPC AWS PrivateLink che forniscono alla funzione l'accesso alle risorse del tuo Amazon VPC.

> **Note**
>
> AWS PrivateLink Gli endpoint VPC sono necessari per le funzioni con mappature delle sorgenti degli eventi che utilizzano la modalità predefinita (su richiesta) per i poller degli eventi. Se la mappatura delle sorgenti degli eventi utilizza la modalità provisioning, non è necessario configurare gli endpoint VPC AWS PrivateLink .

Crea un endpoint per fornire l'accesso alle seguenti risorse:

- Lambda: crea un endpoint per il principale del servizio Lambda.
- AWS STS — Crea un endpoint per consentire AWS STS a un responsabile del servizio di assumere un ruolo per tuo conto.
- Secrets Manager: se il tuo cluster utilizza Secrets Manager per archiviare le credenziali, crea un endpoint per Secrets Manager.

In alternativa, configura un gateway NAT su ogni sottorete pubblica in Amazon VPC. Per ulteriori informazioni, consulta the section called “Accesso Internet per funzioni del VPC”.

Quando crei una mappatura delle sorgenti di eventi per Amazon DocumentDB, Lambda verifica se Elastic Network Interfaces ENIs () sono già presenti per le sottoreti e i gruppi di sicurezza configurati per il tuo Amazon VPC. Se Lambda rileva che esistono ENIs, tenta di riutilizzarli. Altrimenti, Lambda ne crea di nuovi ENIs per connettersi all'origine dell'evento e richiamare la funzione.

> **Note**
>
> Le funzioni Lambda vengono sempre eseguite all'interno del servizio Lambda di VPCs proprietà. La configurazione VPC della funzione non influisce sullo strumento di mappatura

dell'origine degli eventi. Solo la configurazione di rete dell'origine dell'evento determina il modo in cui Lambda si connette all'origine dell'evento.

Configura i gruppi di sicurezza per l'Amazon VPC contenente il tuo cluster. Per impostazione predefinita, Amazon DocumentDB utilizza le seguenti porte: `27017`.

- Regole in ingresso: consenti tutto il traffico sulla porta del broker predefinita per il gruppo di sicurezza associato all'origine eventi. In alternativa, puoi utilizzare una regola del gruppo di sicurezza autoreferenziante per consentire l'accesso da istanze all'interno dello stesso gruppo di sicurezza.
- Regole in uscita: consentono tutto il traffico sulla porta `443` per destinazioni esterne se la funzione deve comunicare con i servizi. AWS In alternativa, puoi anche utilizzare una regola del gruppo di sicurezza autoreferenziale per limitare l'accesso al broker se non hai bisogno di comunicare con altri servizi. AWS
- Regole di ingresso degli endpoint Amazon VPC: se utilizzi un endpoint Amazon VPC, il gruppo di sicurezza associato all'endpoint Amazon VPC deve consentire il traffico in entrata sulla porta `443` dal gruppo di sicurezza del cluster.

Se il cluster utilizza l'autenticazione, puoi anche limitare la policy degli endpoint per l'endpoint Secrets Manager. Per chiamare l'API Secrets Manager, Lambda utilizza il ruolo della funzione, non il principale del servizio Lambda.

Example Policy dell'endpoint VPC: endpoint 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"
          }
      ]
```

```
  }
```

Quando utilizzi gli endpoint Amazon VPC, AWS indirizza le chiamate API per richiamare la tua funzione utilizzando l'Elastic Network Interface (ENI) dell'endpoint. Il responsabile del servizio Lambda deve `lambda:InvokeFunction` richiamare tutti i ruoli e le funzioni che li utilizzano. ENIs

Per impostazione predefinita, gli endpoint Amazon VPC dispongono di policy IAM aperte che consentono un ampio accesso alle risorse. La best practice consiste nel limitare queste policy per eseguire le azioni necessarie utilizzando quell'endpoint. Per garantire che lo strumento di mappatura dell'origine degli eventi sia in grado di invocare la funzione Lambda, la policy degli endpoint VPC deve consentire al principale del servizio Lambda di chiamare `sts:AssumeRole` e `lambda:InvokeFunction`. Limitare le policy degli endpoint VPC per consentire solo le chiamate API provenienti dall'organizzazione impedisce il corretto funzionamento dello strumento di mappatura dell'origine degli eventi, pertanto `"Resource": "*"` è richiesto in queste policy.

Il seguente esempio di policy degli endpoint VPC mostra come concedere l'accesso richiesto al principale del servizio Lambda per gli endpoint AWS STS e Lambda.

Example Policy VPC Endpoint — endpoint AWS STS

```
{
      "Statement": [
          {
              "Action": "sts:AssumeRole",
              "Effect": "Allow",
              "Principal": {
                  "Service": [
                      "lambda.amazonaws.com"
                  ]
              },
              "Resource": "*"
          }
      ]
    }
```

Example Policy dell'endpoint VPC: endpoint Lambda

```
{
      "Statement": [
          {
```

```
            "Action": "lambda:InvokeFunction",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            },
            "Resource": "*"
        }
    ]
}
```

Creazione di una mappatura dell'origine degli eventi Amazon DocumentDB (console)

Per configurare una funzione Lambda per la lettura dal flusso di modifica di un cluster Amazon DocumentDB, crea una mappatura dell'origine degli eventi. In questa sezione viene descritto come eseguire questa operazione dalla console Lambda. Per AWS SDK e AWS CLI istruzioni, consulta. the section called “Creazione di una mappatura dell'origine degli eventi Amazon DocumentDB (SDK o CLI)”

Creazione di una mappatura dell'origine degli eventi per Amazon DocumentDB (console)

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. In Function overview (Panoramica delle funzioni), scegliere Add trigger (Aggiungi trigger).
4. In Configurazione del trigger, nell'elenco a discesa, scegli DocumentDB.
5. Configurare le opzioni richieste, quindi scegliere Add (Aggiungi).

Lambda supporta le seguenti opzioni per le origini degli eventi Amazon DocumentDB:

- Cluster DocumentDB: seleziona un cluster Amazon DocumentDB.
- Attiva il trigger: seleziona l'opzione se vuoi attivare il trigger immediatamente. Se selezioni questa casella, la funzione inizia immediatamente a ricevere traffico dal flusso di modifica specificato di Amazon DocumentDB al momento della creazione della mappatura dell'origine degli eventi. Ai fini del test, è preferibile deselezionare la casella in modo da creare una mappatura dell'origine degli eventi non attiva. Dopo la creazione, puoi attivare lo strumento di mappatura dell'origine degli eventi in qualsiasi momento.

- Nome database: immetti il nome di un database da utilizzare nel cluster.
- (Facoltativo) Nome della raccolta: immetti il nome di una raccolta da utilizzare nel database. Se non specifichi una raccolta, Lambda ascolta tutti gli eventi di ciascuna raccolta del database.
- Dimensioni batch: imposta, fino a 10.000, il numero massimo di messaggi da recuperare in un singolo batch. La dimensione predefinita del batch è pari a 100.
- Posizione iniziale: scegli la posizione del flusso da cui iniziare a leggere i record.
  - Ultimi: elabora solo i nuovi record aggiunti al flusso. La funzione inizia a elaborare i record solo dopo che Lambda ha terminato la creazione dell'origine degli eventi. Ciò significa che alcuni record potrebbero essere tralasciati fino a quando l'origine degli eventi non viene creata correttamente.
  - Trim Horizon (Orizzonte di taglio): elabora tutti i record contenuti nel flusso. Lambda utilizza la durata di conservazione dei log del tuo cluster per stabilire da dove iniziare a leggere gli eventi. In particolare Lambda inizia a leggere da `current_time - log_retention_duration`. Il flusso di modifica deve essere già attivo prima di questo timestamp affinché Lambda legga tutti gli eventi correttamente.
  - At timestamp (Al timestamp): elaborare record a partire da una determinata ora. Il flusso di modifica deve essere già attivo prima del timestamp specificato affinché Lambda legga tutti gli eventi correttamente.
- .Autenticazione: scegli il metodo di autenticazione per l'accesso dei broker al cluster.
  - BASIC_AUTH: con l'autenticazione di base è necessario fornire la chiave Secrets Manager che contiene le credenziali per accedere al cluster.
- Chiave Secrets Manager: scegli la chiave Secrets Manager che contiene i dettagli di autenticazione (nome utente e password) necessari per accedere al cluster Amazon DocumentDB.
- (Facoltativo) Finestra batch: specifica il tempo massimo, espresso in secondi fino a un massimo di 300, per la raccolta dei record prima che la funzione venga richiamata.
- (Facoltativo) Configurazione completa del documento: per le operazioni di aggiornamento del documento, scegli che cosa inviare al flusso. Il valore predefinito è `Default`, il che significa che Amazon DocumentDB invia solo un delta che descrive le modifiche apportate per ogni evento del flusso di modifica. Per ulteriori informazioni su questo campo, consulta la documentazione dell'FullDocumentAPI Javadoc di MongoDB.
  - Impostazione predefinita: Lambda invia solo un documento parziale che descrive le modifiche apportate.
  - UpdateLookup— Lambda invia un delta che descrive le modifiche, insieme a una copia dell'intero documento.

Creazione di una mappatura dell'origine degli eventi Amazon DocumentDB (SDK o CLI)

Per creare o gestire una mappatura dell'origine degli eventi Amazon DocumentDB tramite un SDK AWS, puoi utilizzare le seguenti operazioni dell'API:

- CreateEventSourceMapping
- ListEventSourceMappings
- GetEventSourceMapping
- UpdateEventSourceMapping
- DeleteEventSourceMapping

Per creare la mappatura della sorgente degli eventi con AWS CLI, usa il comando. `create-event-source-mapping` L'esempio seguente utilizza questo comando per mappare una funzione denominata `my-function` a un flusso di modifica Amazon DocumentDB. L'origine degli eventi è indicata da un nome della risorsa Amazon (ARN), con una dimensione batch di 500, a partire dal timestamp in formato Unix. Il comando specifica anche la chiave Secrets Manager che Lambda utilizza per connettersi ad Amazon DocumentDB. Inoltre, include `document-db-event-source-config` parametri che specificano il database e la raccolta da cui leggere.

```
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"}' \
```

L'output visualizzato dovrebbe essere di questo tipo:

```
{
    "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"
}
```

Dopo la creazione, puoi utilizzare il comando `update-event-source-mapping` per modificare le impostazioni di aggiornamento relative all'origine degli eventi di Amazon DocumentDB. Nell'esempio seguente, la dimensione del batch viene aggiornata a 1.000 e la finestra batch a 10 secondi. Per questo comando è necessario l'UUID della mappatura dell'origine degli eventi, recuperabile utilizzando il comando `list-event-source-mapping` o dalla console Lambda.

```
aws lambda update-event-source-mapping --function-name my-function \
    --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
    --batch-size 1000 \
    --batch-window 10
```

L'output visualizzato dovrebbe essere di questo tipo:

```
{
    "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 aggiorna le impostazioni in modo asincrono, pertanto potresti non essere in grado di visualizzare queste modifiche nell'output fino al completamento del processo. Per visualizzare le impostazioni correnti della mappatura dell'origine degli eventi, utilizza il comando `get-event-source-mapping`.

```
aws lambda get-event-source-mapping --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b
```

L'output visualizzato dovrebbe essere di questo tipo:

```
{
    "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"
}
```

Per eliminare la mappatura dell'origine degli eventi di Amazon DocumentDB, utilizza il comando `delete-event-source-mapping`:

```
aws lambda delete-event-source-mapping \
    --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b11284
```

Posizioni di partenza di polling e flussi

Tieni presente che il polling dei flussi durante la creazione e gli aggiornamenti dello strumento di mappatura dell’origine degli eventi alla fine è coerente.

- Durante la creazione dello strumento di mappatura dell'origine degli eventi, potrebbero essere necessari alcuni minuti per l'avvio degli eventi di polling dal flusso.
- Durante gli aggiornamenti dello strumento di mappatura dell'origine degli eventi, potrebbero essere necessari alcuni minuti per l'avvio degli eventi di polling dal flusso.

Questo comportamento implica che se specifichi `LATEST` come posizione iniziale del flusso, lo strumento di mappatura dell'origine degli eventi potrebbe perdere eventi durante la creazione o gli aggiornamenti. Per non perdere alcun evento, specifica la posizione iniziale del flusso come `TRIM_HORIZON` o `AT_TIMESTAMP`.

Monitoraggio dell'origine degli eventi di Amazon DocumentDB

Per aiutarti a monitorare l'origine degli eventi Amazon DocumentDB, Lambda emette il parametro `IteratorAge` quando la funzione termina l'elaborazione di un batch di record. L'età dell'iteratore è la differenza tra il timestamp dell'evento più recente e il timestamp corrente. Sostanzialmente, il parametro `IteratorAge` indica l'età dell'ultimo record elaborato nel batch. Se la funzione continua a elaborare nuovi eventi, puoi utilizzare la cronologia di iterazione per stimare la latenza tra il momento in cui un record viene aggiunto e il momento in cui viene elaborato dalla tua funzione. Una tendenza in aumento in `IteratorAge` può indicare problemi con la funzione. Per ulteriori informazioni, consulta Utilizzo dei parametri di CloudWatch Logs con Lambda.

I flussi di modifiche di Amazon DocumentDB non sono ottimizzati per gestire ampi intervalli di tempo tra gli eventi. Se l'origine eventi Amazon DocumentDB non riceve alcun evento per un periodo di tempo prolungato, Lambda può disabilitare lo strumento di mappatura dell'origine degli eventi. La durata di questo periodo di tempo può variare da alcune settimane a qualche mese a seconda delle dimensioni del cluster e di altri carichi di lavoro.

Lambda supporta payload di dimensioni massime di 6 MB. Tuttavia, gli eventi del flusso di modifica di Amazon DocumentDB possono avere dimensioni fino a 16 MB. Se il flusso di modifica tenta di inviare a Lambda un evento di flusso di modifica di dimensioni superiori a 6 MB, Lambda tralascia il messaggio ed emette il parametro `OversizedRecordCount`. Lambda emette tutte i parametri sulla base del miglior tentativo.

Tutorial: Utilizzo AWS Lambda con Amazon DocumentDB Streams

In questo tutorial creerai una funzione Lambda di base che utilizza gli eventi provenienti da un flusso di modifica Amazon DocumentDB (compatibile con MongoDB). Per completare questo tutorial, saranno completate le seguenti fasi:

- Configura il tuo cluster Amazon DocumentDB, connettiti a esso e attiva i flussi di modifica su di esso.
- Crea la funzione Lambda e configura il cluster Amazon DocumentDB come origine degli eventi Amazon DocumentDB come origine degli eventi per la funzione.
- Verifica la configurazione inserendo elementi nel tuo database Amazon DocumentDB.

Creare il cluster Amazon DocumentDB

1. Apri la console Amazon DocumentDB. In Cluster, scegli Crea.
2. Crea un cluster con la seguente configurazione:
   - Per il tipo di cluster, scegli Cluster basato su istanze. Questa è l'opzione predefinita.
   - In Configurazione del cluster, assicuratevi che sia selezionata la versione 5.0.0 del motore. Questa è l'opzione predefinita.
   - In Configurazione dell'istanza:
     - Per la classe di istanza DB, seleziona Classi ottimizzate per la memoria. Questa è l'opzione predefinita.
     - Per Numero di istanze di replica regolari, scegli 1.
     - Per la classe Instance, usa la selezione predefinita.
   - In Autenticazione, inserisci un nome utente per l'utente principale, quindi scegli Autogestito. Inserisci una password, quindi confermala.
   - Mantieni tutte le altre impostazioni predefinite.
3. Scegli Crea cluster.

Creazione del segreto di Gestione dei segreti

Mentre Amazon DocumentDB crea il tuo cluster, crea un Gestione dei segreti AWS segreto per archiviare le credenziali del database. Fornirai questo segreto quando creerai la mappatura delle sorgenti degli eventi Lambda in un passaggio successivo.

Creazione del segreto in Gestione dei segreti

1. Apri la console Gestione dei segreti e scegli Archivia un nuovo segreto.
2. In Scegli il tipo di segreto, scegli una delle seguenti opzioni:
   - In Dettagli di base:

- Tipo di segreto: credenziali per il database Amazon DocumentDB
- In Credenziali, inserisci lo stesso nome utente e la stessa password che hai usato per creare il tuo cluster Amazon DocumentDB.
- Database: scegli il tuo cluster Amazon DocumentDB.
- Scegli Next (Successivo).

3. Per Configura segreto, scegli tra le seguenti opzioni:
   - Nome segreto: `DocumentDBSecret`
   - Scegli Next (Successivo).
4. Scegli Next (Successivo).
5. Scegli Store.
6. Aggiorna la console per verificare di aver archiviato correttamente il segreto `DocumentDBSecret`.

Nota l'ARN segreto. Lo utilizzerai in un passaggio successivo.

Connessione al cluster

Connettiti al tuo cluster Amazon DocumentDB tramite AWS CloudShell

1. Nella console di gestione di Amazon DocumentDB, in Clusters, individua il cluster che hai creato. Scegli il tuo cluster facendo clic sulla casella di controllo accanto ad esso.
2. Scegli Connect to cluster. Viene visualizzata la schermata di comando CloudShell Esegui.
3. Nel campo Nuovo nome dell'ambiente, inserisci un nome univoco, ad esempio «test» e scegli Crea ed esegui.
4. Specifica la password, quando richiesto. Quando viene richiesto`rs0 [direct: primary] <env-name>>`, la connessione al cluster Amazon DocumentDB è avvenuta correttamente.

Attivazione dei flussi di modifica

Per questo tutorial, monitorerai le modifiche alla raccolta di `products` del database `docdbdemo` nel cluster Amazon DocumentDB. Puoi farlo attivando i flussi di modifica.

Creazione di un nuovo database all'interno del cluster

1. Esegui il seguente comando per creare un nuovo database chiamato: `docdbdemo`

```
use docdbdemo
```

2. Nella finestra del terminale, utilizzate il seguente comando per inserire un record in`docdbdemo`:

```
db.products.insertOne({"hello":"world"})
```

Dovresti vedere un output come questo:

```
{
  acknowledged: true,
  insertedId: ObjectId('67f85066ca526410fd531d59')
}
```

3. Quindi, attiva i flussi di modifica sulla raccolta `products` del database `docdbdemo` utilizzando il comando seguente:

```
db.adminCommand({modifyChangeStreams: 1,
    database: "docdbdemo",
    collection: "products",
    enable: true});
```

L'output visualizzato dovrebbe essere di questo tipo:

```
{ "ok" : 1, "operationTime" : Timestamp(1680126165, 1) }
```

Creazione di endpoint VPC dell'interfaccia

Successivamente, crea gli endpoint VPC di interfaccia per garantire che Lambda e Gestione dei segreti (utilizzati in seguito per archiviare le credenziali di accesso al cluster) possano connettersi al tuo VPC predefinito.

Creazione di endpoint VPC dell'interfaccia

1. Apri la console VPC. Nel menu a sinistra, in Cloud privato virtuale, scegli Endpoint.
2. Seleziona Crea endpoint. Crea un endpoint con la seguente configurazione:
   - Per Nome tag, inserisci `lambda-default-vpc`.
   - Per la categoria di servizi, scegli AWS servizi.

- Per Servizi, digita `lambda` nella casella di ricerca. Scegli il servizio con il formato `com.amazonaws.<region>.lambda`.
- Per VPC, scegli il VPC in cui si trova il tuo cluster Amazon DocumentDB. Si tratta in genere del VPC predefinito.
- Per Sottoreti, seleziona le caselle accanto a ciascuna zona di disponibilità. Scegli l'ID della sottorete corretta per ogni zona di disponibilità.
- Per il tipo di indirizzo IP, selezionare IPv4.
- Per i gruppi di sicurezza, scegli il gruppo di sicurezza utilizzato dal tuo cluster Amazon DocumentDB. Questo è in genere il gruppo `default` di sicurezza.
- Mantieni tutte le altre impostazioni predefinite.
- Seleziona Crea endpoint.

3. Seleziona di nuovo Crea endpoint. Crea un endpoint con la seguente configurazione:

- Per Nome tag, inserisci `secretsmanager-default-vpc`.
- Per la categoria di servizi, scegli AWS servizi.
- Per Servizi, digita `secretsmanager` nella casella di ricerca. Scegli il servizio con il formato `com.amazonaws.<region>.secretsmanager`.
- Per VPC, scegli il VPC in cui si trova il tuo cluster Amazon DocumentDB. Si tratta in genere del VPC predefinito.
- Per Sottoreti, seleziona le caselle accanto a ciascuna zona di disponibilità. Scegli l'ID della sottorete corretta per ogni zona di disponibilità.
- Per il tipo di indirizzo IP, selezionare IPv4.
- Per i gruppi di sicurezza, scegli il gruppo di sicurezza utilizzato dal tuo cluster Amazon DocumentDB. Questo è in genere il gruppo `default` di sicurezza.
- Mantieni tutte le altre impostazioni predefinite.
- Seleziona Crea endpoint.

Questo completa la sezione del tutorial dedicata alla configurazione del cluster.

Creazione del ruolo di esecuzione

Nella serie di passaggi successiva, creerai la funzione Lambda. Innanzitutto, devi creare il ruolo di esecuzione che fornisce alla funzione l'autorizzazione per accedere al cluster. Per farlo, creerai prima una policy IAM, dopodiché la collegherai a un ruolo IAM.

Creazione di una policy IAM

1. Nella console IAM, apri la pagina Policy, quindi scegli Crea policy.
2. Scegli la scheda JSON. Nella policy seguente, sostituisci l'ARN della risorsa Gestione dei segreti nell'ultima riga dell'istruzione con l'ARN del segreto utilizzato in precedenza e copia la policy nell'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. Scegli Successivo: Tag, quindi Successivo: Verifica.
4. In Nome, inserisci `AWSDocumentDBLambdaPolicy`.
5. Scegli Create Policy (Crea policy).

Per creare il ruolo IAM

1. Nella console IAM, apri la pagina Ruoli, quindi scegli Crea ruolo.
2. Per Seleziona un'entità attendibile, scegli le seguenti opzioni:
   - Tipo di entità affidabile: AWS servizio
   - Servizio o caso d'uso: Lambda
   - Scegli Next (Successivo).
3. Per Aggiungi autorizzazioni, scegli la `AWSDocumentDBLambdaPolicy` policy che hai appena creato e `AWSLambdaBasicExecutionRole` autorizza la funzione a scrivere su Amazon CloudWatch Logs.
4. Scegli Next (Successivo).
5. Per Nome ruolo, inserisci `AWSDocumentDBLambdaExecutionRole`.
6. Scegli Crea ruolo.

Creazione della funzione Lambda

Questo tutorial utilizza il runtime di Python 3.14, ma abbiamo anche fornito file di codice di esempio per altri runtime. Per visualizzare il codice per il runtime che ti interessa, seleziona la scheda corrispondente nella casella seguente.

Il codice riceve l'input di un evento Amazon DocumentDB ed elabora il messaggio in esso contenuto.

Creazione della funzione Lambda

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli Crea funzione.

3. Scegli Crea da zero.
4. In Basic information (Informazioni di base) eseguire queste operazioni:
   a. Nel campo Nome funzione, inserisci `ProcessDocumentDBRecords`.
   b. Per Runtime, scegli Python 3.14.
   c. In Architecture (Architettura), scegli x86_64.
5. Nella scheda Modifica ruolo di esecuzione predefinito, effettua le seguenti operazioni:
   a. Espandi la scheda, quindi scegli Utilizza un ruolo esistente.
   b. Seleziona il `AWSDocumentDBLambdaExecutionRole` che hai creato in precedenza.
6. Scegli Crea funzione.

Implementazione del codice della funzione

1. Scegli la scheda Python nella casella seguente e copia il codice.

   .NET

   SDK per .NET

   > **Note**
   >
   > C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

   Utilizzo di un evento Amazon DocumentDB con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento Amazon DocumentDB con Lambda utilizzando. 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));
};
```

Consumo di un evento Amazon DocumentDB con Lambda utilizzando 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite Ruby.

```
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(())

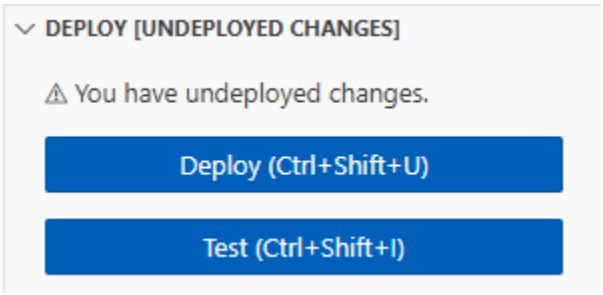
}

#[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. Nel riquadro Codice sorgente della console Lambda, incolla il codice nell'editor di codice, sostituendo il codice creato da Lambda.
3. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:



Creazione della mappatura dell'origine degli eventi Lambda

Crea lo strumento di mappatura dell'origine degli eventi che associa il flusso di modifica di Amazon DocumentDB alla funzione Lambda. Dopo aver creato questa mappatura delle sorgenti degli eventi, inizia AWS Lambda immediatamente il polling dello stream.

Creazione di una mappatura dell'origine degli eventi

1. Apri la pagina Funzioni della console Lambda.
2. Scegli la funzione `ProcessDocumentDBRecords` creata in precedenza.
3. Scegli la scheda Configurazione, quindi scegli Triggers nel menu a sinistra.
4. Selezionare Add trigger (Aggiungi trigger).
5. In Configurazione del trigger, per l'origine seleziona Amazon DocumentDB.
6. Crea la mappatura dell'origine degli eventi con la seguente configurazione:
   - Cluster Amazon DocumentDB: scegli il cluster creato in precedenza.
   - Nome del database: docdbdemo
   - Nome della collezione: prodotti
   - Dimensione del batch: 1
   - Posizione di partenza: più recente
   - Autenticazione: BASIC_AUTH
   - Chiave Secrets Manager: scegli il segreto per il tuo cluster Amazon DocumentDB. Si chiamerà qualcosa del genere`rds!cluster-12345678-a6f0-52c0-b290-db4aga89274f`.
   - Finestra Batch: 1
   - Configurazione completa del documento: UpdateLookup
7. Scegliere Aggiungi. La creazione della mappatura dell'origine degli eventi può richiedere alcuni minuti.

Test della funzione

Attendi che la mappatura della fonte dell'evento raggiunga lo stato Abilitato. Questo processo può richiedere diversi minuti. Quindi, verifica la end-to-end configurazione inserendo, aggiornando ed eliminando i record del database. Prima di iniziare:

1. Riconnettiti al cluster Amazon DocumentDB nel CloudShell tuo ambiente.
2. Esegui il comando seguente per assicurarti di utilizzare il `docdbdemo` database:

```
use docdbdemo
```

Inserisci un record

Inserisci un record nella raccolta `products` del database `docdbdemo`:

```
db.products.insertOne({"name":"Pencil", "price": 1.00})
```

Verifica che la tua funzione abbia elaborato correttamente questo evento controllando CloudWatch Logs. Dovresti vedere una voce di registro come questa:

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	}

Aggiornare un record

Aggiorna il record che hai appena inserito con il seguente comando:

```
db.products.updateOne(
    { "name": "Pencil" },
    { $set: { "price": 0.50 }}
)
```

Verifica che la tua funzione abbia elaborato correttamente questo evento controllando CloudWatch Logs. Dovresti vedere una voce di registro come questa:

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	}

Eliminazione di un registro

Eliminare il record appena aggiornato con il seguente comando:

```
db.products.deleteOne( { "name": "Pencil" } )
```

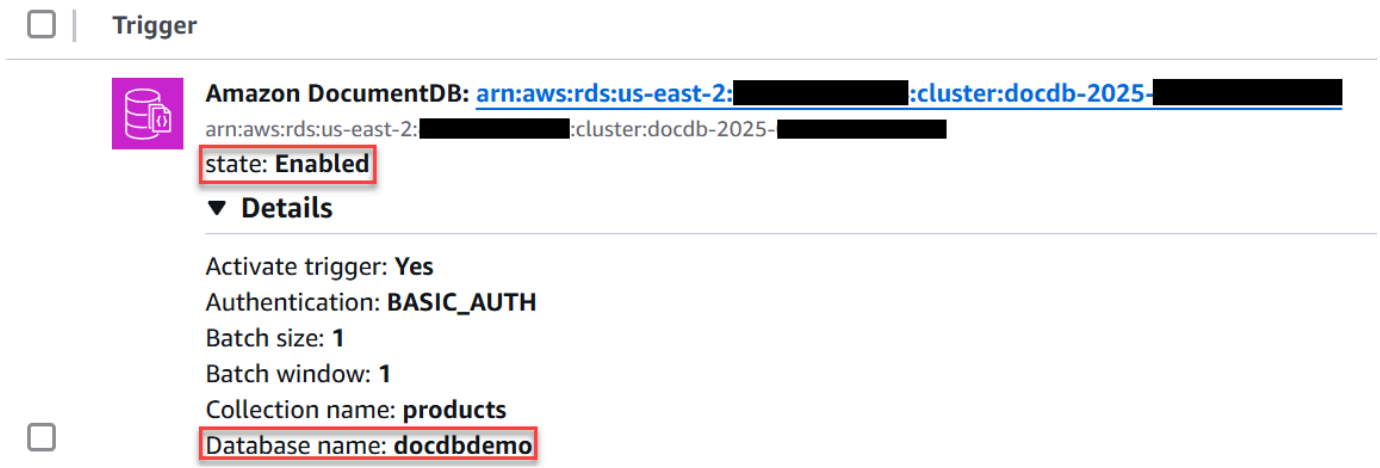
Verifica che la tua funzione abbia elaborato correttamente questo evento controllando CloudWatch Logs. Dovresti vedere una voce di registro come questa:

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: {}

risoluzione dei problemi

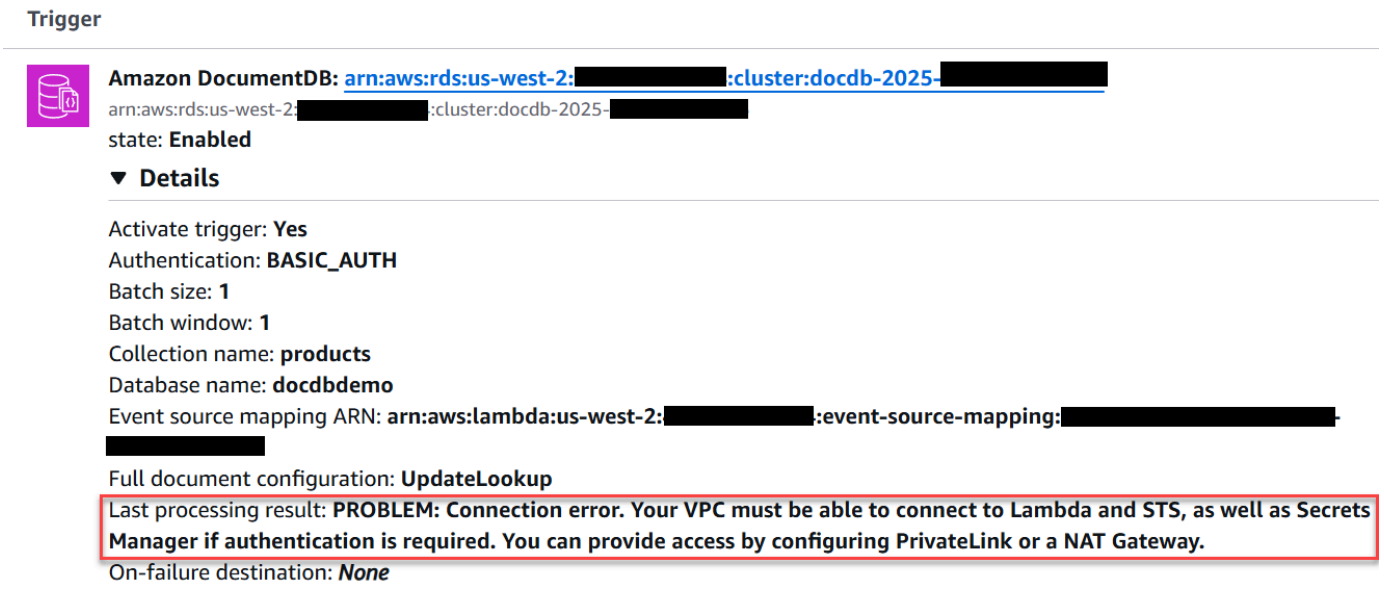
Se non vedi alcun evento del database nei CloudWatch log della tua funzione, controlla quanto segue:

- Assicurati che la mappatura dell'origine degli eventi Lambda (nota anche come trigger) sia nello stato Abilitato. La creazione delle mappature delle sorgenti degli eventi può richiedere diversi minuti.
- Se la mappatura delle sorgenti degli eventi è abilitata ma non riesci ancora a visualizzare gli eventi del database in: CloudWatch
  - Assicurati che il nome del database nella mappatura delle sorgenti degli eventi sia impostato su. `docdbdemo`



  - Controlla la mappatura della sorgente dell'evento nel campo Ultimo risultato di elaborazione il seguente messaggio «PROBLEMA: errore di connessione. Il tuo VPC deve essere in grado di connettersi a Lambda e STS, oltre a Secrets Manager se è richiesta l'autenticazione.» Se vedi questo errore, assicurati di aver creato gli endpoint dell'interfaccia VPC Lambda and Secrets

Manager e che gli endpoint utilizzino lo stesso VPC e le stesse sottoreti utilizzate dal cluster Amazon DocumentDB.



Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando le risorse AWS che non si utilizzano più, è possibile evitare addebiti superflui sul proprio account Account AWS.

Per eliminare la funzione Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **`confirm`** nel campo di immissione del testo, quindi scegli Elimina.

Come eliminare il ruolo di esecuzione

1. Aprire la pagina Ruoli della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Elimina.

Eliminazione degli endpoint VPC

1. Apri la console VPC. Nel menu a sinistra, in Cloud privato virtuale, scegli Endpoint.
2. Seleziona gli endpoint creati.
3. Seleziona Actions (Operazioni), Delete VPC endpoints (Eliminazione di endpoint VPC).
4. Inserisci **`delete`** nel campo di immissione del testo.
5. Scegli Elimina.

Eliminazione del cluster Amazon DocumentDB

1. Apri la console Amazon DocumentDB.
2. Scegli il cluster Amazon DocumentDB che hai creato per questo tutorial e disabilita la protezione dall'eliminazione.
3. Nella pagina principale Cluster, scegli nuovamente il tuo cluster Amazon DocumentDB.
4. Scegli Operazioni > Elimina.
5. Per Crea snapshot finale del cluster, seleziona No.
6. Inserisci **`delete`** nel campo di immissione del testo.
7. Scegli Elimina.

Eliminazione del segreto in Gestione dei segreti

1. Apri la console Secrets Manager.
2. Scegli il segreto creato per questo tutorial.
3. Scegli Operazioni, Elimina segreto.
4. Scegliere Schedule deletion (Pianifica eliminazione).

Uso di AWS Lambda con Amazon DynamoDB

> **Note**
>
> Se desideri inviare dati a una destinazione diversa da una funzione Lambda o arricchire i dati prima di inviarli, consulta Pipe di Amazon EventBridge.

È possibile utilizzare una AWS Lambda funzione per elaborare i record in un flusso Amazon DynamoDB. Con DynamoDB Streams, è possibile attivare una funzione Lambda per eseguire lavoro aggiuntivo ogni volta che una tabella DynamoDB viene aggiornata.

Durante l'elaborazione di flussi DynamoDB, è necessario implementare una logica di risposta batch parziale per evitare che i record elaborati correttamente vengano ritentati quando alcuni record di un batch falliscono. L'utilità Batch Processor di Powertools for AWS Lambda è disponibile in Python, TypeScript, .NET e Java e semplifica questa implementazione gestendo automaticamente la logica di risposta batch parziale, riducendo i tempi di sviluppo e migliorando l'affidabilità.

Argomenti

- Flussi di polling e batching
- Posizioni di partenza di polling e flussi
- Lettori simultanei di uno shard in DynamoDB Streams
- Esempio di evento
- Elaborare i record DynamoDB con Lambda
- Configurazione della risposta batch parziale con DynamoDB e Lambda
- Conservare i record scartati per un'origine eventi DynamoDB in Lambda
- Implementazione dell'elaborazione del flusso DynamoDB stateful in Lambda
- Parametri Lambda per gli strumenti di mappatura dell'origine degli eventi di Amazon DynamoDB
- Utilizzo del filtro eventi con un'origine eventi DynamoDB
- Tutorial: Utilizzo AWS Lambda con flussi Amazon DynamoDB

Flussi di polling e batching

Lambda esegue il polling delle partizioni presenti nel proprio flusso DynamoDB ricercando i record a una velocità di base di 4 volte al secondo. Quando sono disponibili dei record, Lambda invoca la

funzione e attende il risultato. Se l'elaborazione ha esito positivo, Lambda riprende il polling fino a quando non riceve più record.

Per impostazione predefinita, Lambda richiama la funzione non appena i record sono disponibili. Se il batch che Lambda legge dall'origine eventi contiene un solo record, Lambda invia solo un record alla funzione. Per evitare di richiamare la funzione con pochi record è possibile, configurando un periodo di batch, chiedere all'origine eventi di memorizzare nel buffer i registri per un massimo di 5 minuti. Prima di richiamare la funzione, Lambda continua a leggere i registri dall'origine eventi fino a quando non ha raccolto un batch completo, fino alla scadenza del periodo di batch o fino a quando il batch non ha raggiunto il limite del payload di 6 MB. Per ulteriori informazioni, consulta Comportamento di batching.

> **Warning**
>
> Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center AWS.

Lambda non attende il completamento di alcuna estensione prima di inviare il batch successivo per l'elaborazione. In altre parole, le estensioni possono continuare a funzionare mentre Lambda elabora il successivo batch di record. Ciò può causare problemi di limitazione in caso di violazione delle impostazioni o dei limiti di simultaneità. Per rilevare se si tratta di un potenziale problema, monitora le tue funzioni e verifica se i parametri di simultaneità per lo strumento di mappatura dell'origine degli eventi sono superiori al previsto. A causa degli intervalli ridotti tra le invocazioni, Lambda potrebbe segnalare brevemente un utilizzo della simultaneità maggiore rispetto al numero di partizioni. Ciò può essere vero anche per le funzioni Lambda senza estensioni.

Configura l'impostazione ParallelizationFactor per elaborare uno shard di un flusso DynamoDB con più invocazioni Lambda contemporaneamente. È possibile specificare il numero di batch simultanei di cui Lambda esegue il polling da uno shard da un fattore di parallelizzazione compreso tra da 1 (predefinito) e 10. Ad esempio, impostando `ParallelizationFactor` su 2, puoi disporre al massimo di 200 invocazioni Lambda simultanee per elaborare 100 shard di flusso DynamoDB (anche se nella pratica potresti vedere valori differenti per il parametro `ConcurrentExecutions`). Ciò permette di aumentare il throughput di elaborazione quando il volume dei dati è volatile e IteratorAge

è alto. Se si aumenta il numero di batch simultanei per shard, Lambda garantisce comunque l'ordine di elaborazione a livello di elemento (partizione e chiave di ordinamento).

Posizioni di partenza di polling e flussi

Tieni presente che il polling dei flussi durante la creazione e gli aggiornamenti dello strumento di mappatura dell'origine degli eventi alla fine è coerente.

- Durante la creazione dello strumento di mappatura dell'origine degli eventi, potrebbero essere necessari alcuni minuti per l'avvio degli eventi di polling dal flusso.
- Durante gli aggiornamenti dello strumento di mappatura dell'origine degli eventi, potrebbero essere necessari alcuni minuti per l'avvio degli eventi di polling dal flusso.

Questo comportamento implica che se specifichi `LATEST` come posizione iniziale del flusso, lo strumento di mappatura dell'origine degli eventi potrebbe perdere eventi durante la creazione o gli aggiornamenti. Per garantire che nessun evento venga perso, specifica la posizione di inizio del flusso come `TRIM_HORIZON`.

Lettori simultanei di uno shard in DynamoDB Streams

Per le tabelle di una regione singola che non sono tabelle globali, è possibile progettare contemporaneamente fino a due funzioni Lambda per leggere dallo stesso shard di DynamoDB Streams nello stesso momento. Il superamento di questo limite comporta una limitazione delle richieste. Per le tabelle globali consigliamo di limitare il numero di lettori simultanei a uno per evitare richieste di limitazione della larghezza di banda della rete.

Esempio di evento

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"
    }
  ]}
```

Elaborare i record DynamoDB con Lambda

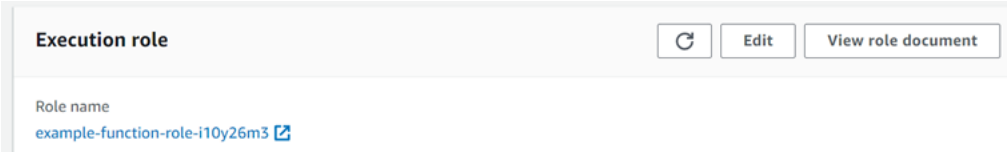
Crea una mappatura dell'origine eventi per indicare a Lambda di inviare i record dal proprio flusso a una funzione Lambda. È possibile creare più mappature dell'origine eventi per elaborare gli stessi dati con più funzioni Lambda o per elaborare elementi da più flussi con una singola funzione.

È possibile configurare gli strumenti di mappatura dell'origine degli eventi per elaborare i record da un flusso in un altro Account AWS. Per ulteriori informazioni, consulta the section called “Mappature tra account”.

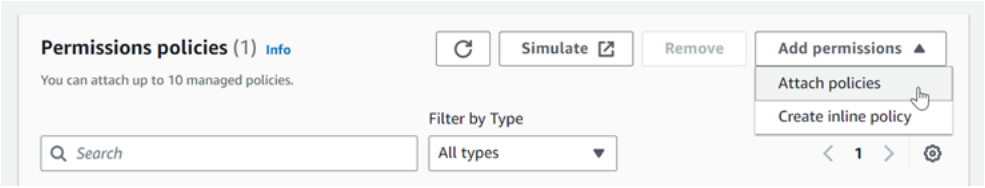
Per configurare la tua funzione per la lettura da DynamoDB Streams, collega la policy gestita di DBExecutionDynamo Role AWSLambda al tuo AWS ruolo di esecuzione e quindi crea un trigger DynamoDB.

Per aggiungere le autorizzazioni e creare un trigger

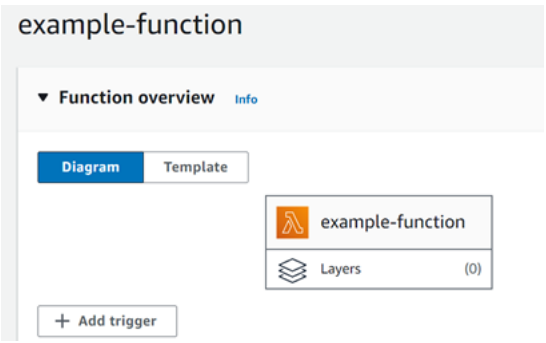
1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Quindi, seleziona la scheda Configuration (Configurazione) e poi Permissions (Autorizzazioni).
4. In Nome del ruolo, scegli il link al tuo ruolo di esecuzione. Questo ruolo si apre nella console IAM.



5. Seleziona Aggiungi autorizzazioni, quindi seleziona Collega policy.



6. Inserisci `AWSLambdaDynamoDBExecutionRole` nel campo di ricerca. Aggiungi questa policy al tuo ruolo di esecuzione. Si tratta di una policy AWS gestita che contiene le autorizzazioni che la funzione deve leggere dal flusso DynamoDB. Per ulteriori informazioni su questa politica, consulta AWSLambdaDynamo DBExecution Role nel Managed Policy Reference.AWS
7. Torna alla funzione nella console Lambda. In Panoramica delle funzioni, scegliere Aggiungi trigger.



8. Scegliere un tipo di trigger.
9. Configurare le opzioni richieste, quindi scegliere Add (Aggiungi).

Lambda supporta le seguenti opzioni per le origini eventi DynamoDB:

Opzioni di origine eventi

- DynamoDB table (Tabella DynamoDB): la tabella DynamoDB da cui leggere i record.
- Batch size (Dimensione batch): il numero di record da inviare alla funzione in ogni batch, fino a 10.000. Lambda passa tutti i record del batch alla funzione in una singola chiamata, purché la dimensione totale degli eventi non superi il limite di payload per l'invocazione sincrona (6 MB).
- Batch window (Periodo di batch): specifica il tempo massimo in secondi per la raccolta dei record prima di richiamare la funzione.
- Starting position (Posizione iniziale): elabora solo i nuovi record o tutti i record esistenti.
  - Latest (Ultimi): consente di elaborare i nuovi record aggiunti al flusso.
  - Trim Horizon (Orizzonte di taglio): elabora tutti i record contenuti nel flusso.

Dopo l'elaborazione di qualsiasi record esistente, la funzione è aggiornata e continua a elaborare nuovi record.

- Destinazione in caso di errore: una coda SQS standard o un argomento SNS standard per i record che non è possibile elaborare. Quando Lambda scarta un batch di record perché è troppo datato o ha esaurito tutti i tentativi, invia i dettagli sul batch alla coda o all'argomento.
- Retry attempts (Nuovi tentativi): il numero massimo di tentativi che Lambda effettua quando la funzione restituisce un errore. Non si applica agli errori di servizio o alle limitazioni in cui il batch non ha raggiunto la funzione.
- Maximum age of record (Età massima del record): l'età massima di un record inviato da Lambda alla funzione.
- Split batch on error (Dividi batch in caso di errore): quando la funzione restituisce un errore, divide il batch in due prima di un nuovo tentativo. L'impostazione originale delle dimensioni del batch rimane invariata.
- Batch simultanei per shard: elabora più batch dallo stesso shard simultaneamente.
- Enabled (Abilitato): impostare su true per abilitare la mappatura dell'origine eventi. Impostare su false per interrompere l'elaborazione dei record. Lambda registra l'ultimo record elaborato e riprende l'elaborazione da quel punto quando la mappatura viene riabilitata.

> **Note**
>
> Non ti vengono addebitati costi per le chiamate GetRecords API richiamate da Lambda come parte dei trigger di DynamoDB.

Per gestire la configurazione dell'origine eventi in un momento successivo, scegliere il trigger nel designer.

Creazione di uno strumento di mappatura dell'origine degli eventi multi-account

Amazon DynamoDB ora supporta policy basate sulle risorse. Con questa funzionalità, puoi elaborare i dati da un flusso DynamoDB in uno Account AWS con una funzione Lambda in un altro account.

Per creare una mappatura delle sorgenti degli eventi per la tua funzione Lambda utilizzando un flusso DynamoDB in un Account AWS altro, devi configurare il flusso utilizzando una policy basata sulle risorse per concedere alla funzione Lambda il permesso di leggere i record. Per informazioni su come

configurare lo stream per l'accesso tra account diversi, consulta Condividi l'accesso con le funzioni Lambda tra account nella Amazon DynamoDB Developer Guide.

Dopo aver configurato lo stream con una policy basata sulle risorse che fornisce alla funzione Lambda le autorizzazioni richieste, crea la mappatura delle sorgenti degli eventi con il tuo ARN del flusso tra account. Puoi trovare l'ARN dello stream nella scheda Esportazioni e flussi della tabella nella console DynamoDB per più account.

Quando usi la console Lambda, incolla l'ARN dello stream direttamente nel campo di input della tabella DynamoDB nella pagina di creazione della mappatura delle sorgenti degli eventi.

Nota: i trigger interregionali non sono supportati.

Configurazione della risposta batch parziale con DynamoDB e Lambda

Quando si consumano ed elaborano i dati di streaming da un'origine eventi, per impostazione predefinita i Lambda imposta i checkpoint al numero di sequenza più alto di un batch solo quando il batch è riuscito completamente. Lambda tratta tutti gli altri risultati come un fallimento completo e riprova a elaborare il batch fino al limite di tentativi. Per consentire i successi parziali durante l'elaborazione di batch da un flusso, attivare `ReportBatchItemFailures`. Consentire successi parziali può contribuire a ridurre il numero di tentativi su un record, anche se non impedisce del tutto la possibilità di tentativi in un record riuscito.

Per attivare `ReportBatchItemFailures`, includere il valore enum**`ReportBatchItemFailures`** nell'FunctionResponseTypeselenco. Questo elenco indica quali tipi di risposta sono abilitati per la funzione. È possibile configurare questo elenco quando si crea o si aggiorna uno strumento di mappatura dell'origine degli eventi.

> **Note**
>
> Anche quando il codice della funzione restituisce risposte parziali di errore in batch, queste risposte non verranno elaborate da Lambda a meno che la `ReportBatchItemFailures` funzionalità non sia esplicitamente attivata per la mappatura delle sorgenti degli eventi.

Sintassi di report

Quando si configura la creazione di report sugli errori degli elementi batch, la `StreamsEventResponse` classe viene restituita con un elenco di errori degli articoli batch. È possibile utilizzare un `StreamsEventResponse` oggetto per restituire il numero di sequenza

del primo record non riuscito nel batch. È inoltre possibile creare la propria classe personalizzata utilizzando la sintassi di risposta corretta. La seguente struttura JSON mostra la sintassi di risposta richiesta:

```
{
  "batchItemFailures": [
        {
            "itemIdentifier": "<SequenceNumber>"
        }
    ]
}
```

> **Note**
>
> Se l'array `batchItemFailures` contiene più elementi, Lambda utilizza il record con il numero di sequenza più basso come checkpoint. Lambda quindi riprova tutti i record a partire da quel checkpoint.

Condizioni di successo e di errore

Lambda considera un batch come un successo completo se si restituisce uno degli elementi seguenti:

- Una `batchItemFailure` lista vuota
- Un `batchItemFailure` elenco nullo
- Un vuoto `EventResponse`
- Un valore nullo `EventResponse`

Lambda considera un batch come un fallimento completo se si restituisce uno degli elementi seguenti:

- Una stringa vuota `itemIdentifier`
- Un valore nullo `itemIdentifier`
- Un `itemIdentifier` con un nome chiave errato

Lambda esegue nuovi tentativi in seguito ai fallimenti secondo la strategia di tentativi impostata.

Bisezione un batch

Se il richiamo fallisce ed `BisectBatchOnFunctionError` è attivato, il batch viene bisecato a prescindere dalle `ReportBatchItemFailures` impostazioni.

Quando si riceve una risposta di successo parziale del batch ed entrambi `BisectBatchOnFunctionError` e `ReportBatchItemFailures` sono attivati, il batch viene bisecato in corrispondenza del numero di sequenza restituito e Lambda ritenta solo i record rimanenti.

Per semplificare l'implementazione della logica di risposta batch parziale, prendi in considerazione l'utilizzo dell'utilità Batch Processor di Powertools for AWS Lambda, che gestisce automaticamente queste complessità per te.

Ecco alcuni esempi di codice di funzione che restituiscono l'elenco dei messaggi non riusciti IDs nel batch:

.NET

SDK per .NET

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione degli errori degli elementi batch di DynamoDB utilizzando 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: [] };
};
```

Segnalazione degli errori degli elementi batch di DynamoDB utilizzando 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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
}
```

Utilizzo di Powertools per il processore AWS Lambda batch

L'utilità per il processore batch di Powertools gestisce AWS Lambda automaticamente la logica di risposta parziale in batch, riducendo la complessità dell'implementazione della segnalazione degli errori dei batch. Ecco alcuni esempi di utilizzo del processore batch:

Python

> **Note**
>
> Per esempi completi e istruzioni di configurazione, consultate la documentazione del processore batch.

Elaborazione dei record AWS Lambda di flusso DynamoDB con processore 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**
>
> Per esempi completi e istruzioni di configurazione, consultate la documentazione del processore batch.

Elaborazione dei record AWS Lambda di flusso DynamoDB con processore 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**
>
> Per esempi completi e istruzioni di configurazione, consultate la documentazione del processore batch.

Elaborazione dei record AWS Lambda di flusso DynamoDB con processore 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**
>
> Per esempi completi e istruzioni di configurazione, consultate la documentazione del processore batch.

Elaborazione dei record AWS Lambda di flusso DynamoDB con processore 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;
    }
}
```

Conservare i record scartati per un'origine eventi DynamoDB in Lambda

La gestione degli errori per gli strumenti di mappatura dell'origine degli eventi DynamoDB dipende dal fatto che l'errore si verifichi prima che la funzione venga richiamata o durante la chiamata della funzione:

- Prima della chiamata: se una mappatura dell'origine degli eventi Lambda non è in grado di richiamare la funzione a causa di limitazioni o altri problemi, riprova finché i record non scadono o superano l'età massima configurata nella mappatura dell'origine dell'evento (). MaximumRecordAgeInSeconds
- Durante la chiamata: se la funzione viene richiamata ma restituisce un errore, Lambda riprova fino alla scadenza dei record, al superamento dell'età massima () o al raggiungimento della quota di tentativi configurata (MaximumRecordAgeInSeconds). MaximumRetryAttempts Per gli errori di funzione, puoi anche configurare, che divide un batch non riuscito in due batch più piccoli BisectBatchOnFunctionError, isolando i record non validi ed evitando i timeout. La divisione dei batch non consuma la quota di tentativi.

Se le misure di gestione degli errori non riescono, Lambda elimina i record e continua l'elaborazione dei batch dal flusso. Con le impostazioni predefinite, ciò significa che un record errato può bloccare l'elaborazione sullo shard interessata per un massimo di un giorno. Per evitare questa situazione, configura la mappatura dell'origine eventi della funzione con un numero ragionevole di tentativi e un'età massima dei record che sia adatta al caso d'uso.

Configurazione delle destinazioni per le chiamate non riuscite

Per mantenere i record delle chiamate non riuscite allo strumento di mappatura dell'origine degli eventi, aggiungi una destinazione allo strumento di mappatura dell'origine degli eventi della funzione. Ogni record inviato alla destinazione è un documento JSON contenente i metadati relativi alla chiamata non riuscita. Per le destinazioni Amazon S3, Lambda invia insieme ai metadati anche l'intero record di invocazione. Puoi configurare qualsiasi argomento Amazon SNS, coda Amazon SQS, bucket Amazon S3 o Kafka come destinazione.

Con le destinazioni Amazon S3, puoi utilizzare la funzionalità Notifiche eventi Amazon S3 per ricevere notifiche quando gli oggetti vengono caricati nel bucket S3 di destinazione. Puoi anche configurare Notifiche eventi S3 per richiamare un'altra funzione Lambda per eseguire l'elaborazione automatica su batch non riusciti.

Il tuo ruolo di esecuzione deve avere le autorizzazioni per la destinazione:

- Per una destinazione SQS: sqs: SendMessage
- Per una destinazione SNS: sns:publish
- Per una destinazione S3: s3: e s3: PutObject ListBucket
- Per una destinazione Kafka: kafka-cluster: WriteData

È possibile configurare un argomento di Kafka come destinazione in caso di errore per le mappature delle sorgenti degli eventi Kafka. Quando Lambda non è in grado di elaborare i record dopo estenuanti tentativi o quando i record superano l'età massima, Lambda invia i record non riusciti all'argomento Kafka specificato per un'elaborazione successiva. Fai riferimento a the section called “Destinazione Kafka in caso di errore”.

Se hai abilitato la crittografia con la tua chiave KMS per una destinazione S3, anche il ruolo di esecuzione della funzione deve avere l'autorizzazione a chiamare kms:. GenerateDataKey Se la chiave KMS e la destinazione del bucket S3 si trovano in un account diverso dalla funzione Lambda e dal ruolo di esecuzione, configura la chiave KMS in modo che consideri attendibile il ruolo di esecuzione da consentire. kms: GenerateDataKey

Per configurare una destinazione in caso di errore tramite la console, completa i seguenti passaggi:

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.

3. In Function overview (Panoramica delle funzioni), scegliere Add destination (Aggiungi destinazione).
4. Per Origine, scegli Chiamata allo strumento di mappatura dell'origine degli eventi.
5. Per Strumento di mappatura dell'origine degli eventi, scegli un'origine dell'evento configurata per questa funzione.
6. Per Condizione, seleziona In caso di errore. Per le chiamate allo strumento di mappatura dell'origine degli eventi, questa è l'unica condizione accettata.
7. Per Tipo di destinazione, scegli il tipo di destinazione a cui Lambda deve inviare i record di chiamata.
8. Per Destination (Destinazione), scegliere una risorsa.
9. Scegli Save (Salva).

Puoi anche configurare una destinazione in caso di errore utilizzando (). AWS Command Line Interface AWS CLI Ad esempio, il create-event-source-mappingcomando seguente aggiunge una mappatura dell'origine degli eventi con una destinazione SQS in caso di errore a: `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"}}'
```

Il update-event-source-mappingcomando seguente aggiorna una mappatura dell'origine degli eventi per inviare i record di chiamata non riuscita a una destinazione SNS dopo due tentativi o se i record risalgono a più di un'ora.

```
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"}}'
```

Le impostazioni aggiornate sono applicate in modo asincrono e non sono riflesse nell'output fino al completamento del processo. Utilizza il comando get-event-source-mapping per visualizzare lo stato corrente.

Per rimuovere una destinazione, fornisci una stringa vuota come argomento del parametro `destination-config`:

```
aws lambda update-event-source-mapping \
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

Best practice di sicurezza per destinazioni Amazon S3

L'eliminazione di un bucket S3 configurato come destinazione senza rimuovere la destinazione dalla configurazione della funzione può creare un rischio per la sicurezza. Se un altro utente conosce il nome del bucket di destinazione, può ricreare il bucket nel proprio Account AWS. I record delle invocazioni non riuscite verranno inviati al relativo bucket, esponendo potenzialmente i dati della tua funzione.

> **Warning**
>
> Per garantire che i record di invocazione della tua funzione non possano essere inviati a un bucket S3 in un altro Account AWS, aggiungi una condizione al ruolo di esecuzione della funzione che limiti le autorizzazioni ai bucket del tuo account. `s3:PutObject`

Di seguito viene illustrato un esempio di policy IAM che limita le autorizzazioni `s3:PutObject` della funzione ai bucket presenti nell'account. Questa policy fornisce inoltre a Lambda l'autorizzazione `s3:ListBucket` necessaria per utilizzare un bucket S3 come destinazione.

```
{
    "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"
                }
            }
        }
    ]
}
```

Per aggiungere una politica di autorizzazioni al ruolo di esecuzione della tua funzione utilizzando Console di gestione AWS o AWS CLI, consulta le istruzioni nelle seguenti procedure:

Console

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Seleziona la funzione Lambda di cui si desidera modificare il ruolo di esecuzione.
3. Nella scheda Configurazione scegli Autorizzazioni.
4. Nella scheda Ruolo di esecuzione, seleziona il nome del ruolo della funzione per aprire la pagina della console IAM del ruolo.
5. Aggiungi una policy di autorizzazioni di base al ruolo completando le seguenti operazioni:
    a. Nel riquadro Policy di autorizzazioni, scegli Aggiungi autorizzazioni, poi Crea policy in linea.
    b. Nell'editor delle policy, seleziona JSON.
    c. Incolla la policy che desideri aggiungere nell'editor (sostituendo il codice JSON esistente), quindi scegli Avanti.
    d. In Dettagli della policy, specifica un nome per la policy.
    e. Scegli Crea policy.

AWS CLI

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (CLI)

1. Crea un documento di policy JSON con le autorizzazioni richieste e salvalo in una directory locale.
2. Utilizza il comando della CLI `put-role-policy` di IAM per aggiungere le autorizzazioni per il ruolo di esecuzione di una funzione. Esegui il comando seguente dalla directory in cui hai

salvato il documento di policy JSON e sostituisci il nome del ruolo, il nome della policy e il documento di policy con i tuoi valori.

```
aws iam put-role-policy \
--role-name my_lambda_role \
--policy-name LambdaS3DestinationPolicy \
--policy-document file://my_policy.json
```

Record di invocazione Amazon SNS e Amazon SQS di esempio

L'esempio seguente mostra un record di chiamata che Lambda invia a una destinazione SQS o SNS per un flusso 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"
    }
}
```

È possibile utilizzare queste informazioni per recuperare i record interessati dal flusso per la risoluzione dei problemi. I record effettivi non sono inclusi, pertanto è necessario elaborare questo record e recuperarli dal flusso prima che scadano e vadano persi.

Record di invocazione Amazon S3 di esempio

L'esempio seguente mostra un record di invocazione che Lambda invia a un bucket S3 per un flusso DynamoDB. Oltre a tutti i campi dell'esempio precedente per le destinazioni SQS e SNS, il campo `payload` contiene il record di chiamata originale come stringa JSON con escape.

```
{
    "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
}
```

L'oggetto S3 contenente il record di invocazione utilizza la seguente convenzione di denominazione:

```
aws/lambda/<ESM-UUID>/<shardID>/YYYY/MM/DD/YYYY-MM-DDTHH.MM.SS-<Random UUID>
```

Implementazione dell'elaborazione del flusso DynamoDB stateful in Lambda

Le funzioni Lambda possono eseguire applicazioni di elaborazione di flussi continui. Un flusso rappresenta dati illimitati che scorrono continuamente attraverso l'applicazione. Per analizzare le informazioni da questo input di aggiornamento continuo, è possibile associare i record inclusi utilizzando una finestra definita in termini di tempo.

Le finestre a cascata sono finestre temporali distinte che si aprono e si chiudono a intervalli regolari. Per impostazione predefinita, le invocazioni Lambda sono senza stato: non è possibile utilizzarle per l'elaborazione dei dati tra più invocazioni continue senza un database esterno. Tuttavia, con la finestra a cascata, è possibile mantenere il proprio stato tra le invocazioni. Questo stato contiene il risultato aggregato dei messaggi precedentemente elaborati per la finestra corrente. Lo stato può essere un massimo di 1 MB per shard. Se supera quella dimensione, Lambda termina la finestra in anticipo.

Ogni record in un flusso appartiene a una finestra specifica. Lambda elaborerà ogni record almeno una volta, ma non garantisce che ogni record venga elaborato una sola volta. In rari casi, come nel caso della gestione degli errori, alcuni record potrebbero essere elaborati più di una volta. La prima volta i record vengono sempre elaborati in ordine. Se i record vengono elaborati più di una volta, possono essere elaborati fuori ordine.

Aggregazione ed elaborazione

La funzione gestita dall'utente viene richiamata sia per l'aggregazione che per l'elaborazione dei risultati finali di tale aggregazione. Lambda aggrega tutti i record ricevuti nella finestra. È possibile ricevere questi record in più batch, ciascuno come richiamo separato. Ogni richiamo riceve uno stato. Pertanto, quando si utilizzano finestre a cascata, la risposta della funzione Lambda deve contenere una proprietà di `state`. Se la risposta non contiene una proprietà di `state`, Lambda la considera un'invocazione non riuscita. Per soddisfare questa condizione, la funzione può restituire un oggetto `TimeWindowEventResponse`, che presenta la seguente forma JSON:

Example **`TimeWindowEventResponse`** valori

```
{
    "state": {
        "1": 282,
        "2": 715
    },
    "batchItemFailures": []
}
```

> **Note**
>
> Per le funzioni Java, si consiglia di utilizzare una `Map<String, String>` per rappresentare lo stato.

Alla fine della finestra, il flag `isFinalInvokeForWindow` è impostato `true` per indicare che questo è lo stato finale e che è pronto per l'elaborazione. Dopo l'elaborazione, la finestra viene completata e il richiamo finale viene completato e quindi lo stato viene eliminato.

Al termine della finestra, Lambda utilizza l'elaborazione finale per le operazioni sui risultati dell'aggregazione. L'elaborazione finale viene richiamata in modo sincrono. Dopo il richiamo riuscito, la funzione controlla il numero di sequenza e l'elaborazione del flusso continua. Se il richiamo non ha esito positivo, la funzione Lambda sospende l'ulteriore elaborazione fino a quando non viene eseguito correttamente il richiamo.

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
}
```

Configurazione

È possibile configurare le finestre a cascata quando si crea o si aggiorna un mapping di origini di eventi. Per configurare una finestra a cascata, specifica la finestra in secondi (TumblingWindowInSeconds). Il comando di esempio seguente AWS Command Line Interface (AWS CLI) crea un mapping di origini di eventi di streaming con una finestra a cascata di 120 secondi. La

funzione Lambda definita per l'aggregazione e l'elaborazione è denominata `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 determina i limiti delle finestre a cascata in base al momento in cui i record sono stati inseriti nel flusso. Tutti i record dispongono di un timestamp approssimativo che Lambda utilizza nelle determinazioni dei limiti.

Le aggregazioni delle finestre a cascata non supportano il risharding. Quando lo shard termina, Lambda considera la finestra chiusa e gli shard secondari iniziano la propria finestra in uno stato nuovo.

Le finestre a cascata supportano completamente le policy di ripetizione dei tentativi esistenti `maxRetryAttempts` e `maxRecordAge`.

Example Handler.py: aggregazione ed elaborazione

La seguente funzione Python mostra come aggregare e quindi elaborare lo stato finale:

```
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}
```

Parametri Lambda per gli strumenti di mappatura dell'origine degli eventi di Amazon DynamoDB

Tutti i tipi di origine eventi Lambda condividono le stesse operazioni API CreateEventSourceMapping e UpdateEventSourceMapping. Tuttavia, solo alcuni dei parametri si applicano a DynamoDB Streams.

Parametro	Obbligatorio	Predefinito	Note
BatchSize	N	100	Massimo: 10.000.
BisectBatchOnFunctionError	N	false	nessuno
DestinationConfig	N	N/D	Una destinazione della coda standard di Amazon SQS o di un argomento standard di Amazon SNS per i record scartati.
Abilitato	N	true	nessuno
EventSourceArn	Y	N/D	L'ARN del flusso dei dati o di un consumer di flusso.
FilterCriteria	N	N/D	Controllare gli eventi che Lambda invia alla funzione
FunctionName	Y	N/D	nessuno

Parametro	Obbligatorio	Predefinito	Note
FunctionResponseTypes	N	N/D	Per consentire alla funzione di segnalare errori specifici in un batch, includi il valore `ReportBatchItemFailures` in `FunctionResponseTypes` . Per ulteriori informazioni, consulta Configurazione della risposta batch parziale con DynamoDB e Lambda.
MaximumBatchingWindowInSeconds	N	0	nessuno
MaximumRecordAgeInSeconds	N	-1	-1 sta per infinito: i record non riusciti vengono ripetuti fino alla scadenza del record. Il limite di conservazione dei dati per i flussi DynamoDB è di 24 ore. Minimo: -1 Massimo: 604.800

Parametro	Obbligatorio	Predefinito	Note
MaximumRetryAttempts	N	-1	-1 sta per infinito: i record non riusciti vengono ripetuti fino alla scadenza del record Minimo: 0 Massimo: 10.000.
ParallelizationFactor	N	1	Maximum: 10
StartingPosition	Y	N/D	TRIM_HORIZON o LATEST
TumblingWindowInSeconds	N	N/D	Minimo: 0 Massimo: 900

Utilizzo del filtro eventi con un'origine eventi DynamoDB

Puoi utilizzare il filtraggio degli eventi per controllare quali record di un flusso o di una coda Lambda invia alla funzione. Per informazioni generali sul funzionamento del filtraggio eventi, consulta the section called “Filtro eventi”.

In questa sezione viene descritto il filtraggio degli eventi per le origini eventi DynamoDB.

> **Note**
>
> Le mappature delle sorgenti degli eventi DynamoDB supportano solo il filtraggio sulla chiave. `dynamodb`

Argomenti

- Evento DynamoDB
- Filtraggio con attributi di tabella

- Filtraggio con espressioni booleane
- Utilizzo dell'operatore Exists
- Formato JSON per il filtro DynamoDB

Evento DynamoDB

Supponiamo di avere una tabella DynamoDB con la chiave primaria `CustomerName` e gli attributi `AccountManager` e `PaymentTerms`. Di seguito è riportato un esempio di record dal flusso della tabella DynamoDB.

```
{
      "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"
      }
  }
```

Per filtrare in base ai valori della chiave e degli attributi nella tabella DynamoDB, utilizza la chiave `dynamodb` nel record. Le seguenti sezioni forniscono esempi per diversi tipi di filtri.

Filtraggio con chiavi di tabella

Supponiamo che tu voglia che la tua funzione elabori solo i record in cui la chiave primaria `CustomerName` è "AnyCompany Industries". L'oggetto `FilterCriteria` dovrebbe avere la struttura seguente.

```
{
     "Filters": [
          {
              "Pattern": "{ \"dynamodb\" : { \"Keys\" : { \"CustomerName\" : { \"S\" :
 [ \"AnyCompany Industries\" ] } } } }"
          }
      ]
 }
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
     "dynamodb": {
          "Keys": {
              "CustomerName": {
                  "S": [ "AnyCompany Industries" ]
                  }
              }
          }
 }
```

Puoi aggiungere il filtro utilizzando la console, la AWS CLI o un modello AWS SAM.

Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in Collegamento dei criteri di filtro a una mappatura dell'origine evento (console) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "dynamodb" : { "Keys" : { "CustomerName" : { "S" : [ "AnyCompany
 Industries" ] } } } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtraggio utilizzando l'AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
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\" ] } } } }"}]}'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"dynamodb\" : { \"Keys\" :
 { \"CustomerName\" : { \"S\" : [ \"AnyCompany Industries\" ] } } } }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML dell'origine degli eventi.

```
FilterCriteria:
   Filters:
     - Pattern: '{ "dynamodb" : { "Keys" : { "CustomerName" : { "S" : [ "AnyCompany
 Industries" ] } } } }'
```

Filtraggio con attributi di tabella

Con DynamoDB, puoi anche utilizzare le chiavi `NewImage` e `OldImage` per filtrare i valori degli attributi. Supponiamo di voler filtrare i record in cui l'attributo `AccountManager` nell'ultima immagine della tabella è "Pat Candella" o "Shirley Rodriguez". L'oggetto `FilterCriteria` dovrebbe avere la struttura seguente.

```
{
    "Filters": [
        {
            "Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S
\" : [ \"Pat Candella\", \"Shirley Rodriguez\" ] } } } }"
        }
    ]
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
    "dynamodb": {
        "NewImage": {
            "AccountManager": {
                "S": [ "Pat Candella", "Shirley Rodriguez" ]
            }
        }
    }
}
```

Puoi aggiungere il filtro utilizzando la console, la AWS CLI o un modello AWS SAM.

Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in Collegamento dei criteri di filtro a una mappatura dell'origine evento (console) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat Candella",
 "Shirley Rodriguez" ] } } } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtraggio utilizzando l'AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
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
\" ] } } } }"}]}'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
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

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML dell'origine degli eventi.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
 Candella", "Shirley Rodriguez" ] } } } }'
```

Filtraggio con espressioni booleane

È inoltre possibile creare filtri utilizzando espressioni booleane AND. Queste espressioni possono includere sia i parametri chiave sia quelli degli attributi della tabella. Supponiamo che tu voglia filtrare i record in cui il valore di `NewImage` di `AccountManager` è "Pat Candella" e il valore di `OldImage` è "Terry Whitlock". L'oggetto `FilterCriteria` dovrebbe avere la struttura seguente.

```
{
    "Filters": [
        {
            "Pattern": "{ \"dynamodb\" : { \"NewImage\" : { \"AccountManager\" : { \"S
\" : [ \"Pat Candella\" ] } } } , \"dynamodb\" : { \"OldImage\" : { \"AccountManager
\" : { \"S\" : [ \"Terry Whitlock\" ] } } } }"
        }
    ]
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
    "dynamodb" : {
        "NewImage" : {
            "AccountManager" : {
                "S" : [
                    "Pat Candella"
                ]
            }
```

```
        }
    },
    "dynamodb": {
        "OldImage": {
            "AccountManager": {
                "S": [
                    "Terry Whitlock"
                ]
            }
        }
    }
}
```

Puoi aggiungere il filtro utilizzando la console, la AWS CLI o un modello AWS SAM.

Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in Collegamento dei criteri di filtro a una mappatura dell'origine evento (console) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
 Candella" ] } } } , "dynamodb" : { "OldImage" : { "AccountManager" : { "S" :
 [ "Terry Whitlock" ] } } } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtraggio utilizzando l'AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
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\" ] } } } }
 "}]}'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
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

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML dell'origine degli eventi.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "dynamodb" : { "NewImage" : { "AccountManager" : { "S" : [ "Pat
 Candella" ] } } } , "dynamodb" : { "OldImage" : { "AccountManager" : { "S" :
 [ "Terry Whitlock" ] } } } }'
```

> **Note**
>
> Il filtraggio degli eventi DynamoDB non supporta l'uso di operatori numerici (uguali numerici e intervallo numerico). Anche se gli elementi della tabella sono memorizzati come numeri, questi parametri vengono convertiti in stringhe nell'oggetto record JSON.

Utilizzo dell'operatore Exists

A causa del modo in cui sono strutturati gli oggetti evento JSON di DynamoDB, l'utilizzo dell'operatore Exists richiede particolare attenzione. L'operatore Exists funziona solo sui nodi foglia nel JSON dell'evento, quindi se il modello di filtro utilizza Exists per testare un nodo intermedio, non funzionerà. Considera il seguente elemento della tabella DynamoDB:

```
{
  "UserID": {"S": "12345"},
  "Name": {"S": "John Doe"},
  "Organizations": {"L": [
      {"S":"Sales"},
      {"S":"Marketing"},
      {"S":"Support"}
    ]
  }
```

```
}
```

Potresti voler creare uno schema di filtro come il seguente per verificare la presenza di eventi contenenti `"Organizations"`:

```
{ "dynamodb" : { "NewImage" : { "Organizations" : [ { "exists": true } ] } } }
```

Tuttavia, questo modello di filtro non restituirebbe mai una corrispondenza perché `"Organizations"` non è un nodo foglia. L'esempio seguente mostra come utilizzare correttamente l'operatore Exists per creare lo schema di filtro desiderato:

```
{ "dynamodb" : { "NewImage" : {"Organizations": {"L": {"S": [ {"exists":
 true } ] } } } } }
```

Formato JSON per il filtro DynamoDB

Per filtrare correttamente gli eventi da origini DynamoDB, sia il campo dati sia i criteri di filtraggio per il campo dati (`dynamodb`) devono essere in un formato JSON valido. Se uno dei due campi non è in un formato JSON valido, Lambda rilascia il messaggio o genera un'eccezione. La tabella seguente riepiloga il comportamento specifico:

Formato dei dati in entrata	Formato del modello di filtro per le proprietà di dati	Operazione risultante
JSON valido	JSON valido	Filtri Lambda in base ai criteri di filtro.
JSON valido	Nessun modello di filtro per le proprietà dei dati	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.
JSON valido	Non-JSON	Lambda genera un'eccezione al momento della creazione o dell'aggiornamento della mappatura dell'origine evento. Il modello di filtro per le proprietà dei dati deve essere in un formato JSON valido.

Formato dei dati in entrata	Formato del modello di filtro per le proprietà di dati	Operazione risultante
Non-JSON	JSON valido	Lambda rilascia il registro.
Non-JSON	Nessun modello di filtro per le proprietà dei dati	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.
Non-JSON	Non-JSON	Lambda genera un'eccezione al momento della creazione o dell'aggiornamento della mappatura dell'origine evento. Il modello di filtro per le proprietà dei dati deve essere in un formato JSON valido.

Tutorial: Utilizzo AWS Lambda con flussi Amazon DynamoDB

In questo tutorial creerai una funzione Lambda per consumare eventi da un flusso Amazon DynamoDB.

Prerequisiti

Installa il AWS Command Line Interface

Se non l'hai ancora installato AWS Command Line Interface, segui i passaggi indicati in Installazione o aggiornamento della versione più recente di AWS CLI per installarlo.

Per eseguire i comandi nel tutorial, sono necessari un terminale a riga di comando o una shell (interprete di comandi). In Linux e macOS, utilizza la shell (interprete di comandi) e il gestore pacchetti preferiti.

> **Note**
>
> Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere

una versione integrata su Windows di Ubuntu e Bash, installa il sottosistema Windows per Linux.

Creazione del ruolo di esecuzione

Crea il ruolo di esecuzione che autorizza la funzione ad accedere alle AWS risorse.

Per creare un ruolo di esecuzione

1. Apri la pagina Ruoli nella console IAM.
2. Scegliere Crea ruolo.
3. Creare un ruolo con le seguenti proprietà.
   - Entità attendibile – Lambda.
   - Autorizzazioni — AWSLambdaDBExecutionDynamo Role.
   - Nome ruolo – **`lambda-dynamodb-role`**.

Il DBExecutionruolo AWSLambda Dynamo dispone delle autorizzazioni necessarie alla funzione per leggere gli elementi da DynamoDB e scrivere i log nei log. CloudWatch

Creazione della funzione

Crea una funzione Lambda che elabora gli eventi DynamoDB. Il codice della funzione scrive alcuni dei dati degli eventi in entrata in Logs. CloudWatch

.NET

SDK per .NET

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento DynamoDB con Lambda utilizzando. 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)}`);
};
```

Consumo di un evento DynamoDB con Lambda utilizzando. 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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(())

}
```

Per creare la funzione

1. Copiare il codice di esempio in un file denominato `example.js`.
2. Crea un pacchetto di implementazione.

```
zip function.zip example.js
```

3. Creare una funzione Lambda con il comando `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
```

Test della funzione Lambda

In questo passaggio, si richiama manualmente la funzione Lambda utilizzando il comando `invoke` AWS Lambda CLI e il seguente evento DynamoDB di esempio. Copia quanto riportato di seguito in un file denominato `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"
        }
    ]
}
```

Eseguire il seguente comando `invoke`.

```
aws lambda invoke --function-name ProcessDynamoDBRecords \
    --cli-binary-format raw-in-base64-out \
    --payload file://input.txt outputfile.txt
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la versione 2. AWS CLI Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

La funzione restituisce la stringa `message` nel corpo della risposta.

Verificare l'output nel file `outputfile.txt`.

Creazione di una tabella DynamoDB con un flusso abilitato

Crea una tabella Amazon DynamoDB con un flusso abilitato.

Per creare una tabella DynamoDB

1. Aprire la console DynamoDB.
2. Scegliere Create table (Crea tabella).
3. Creare una tabella con le impostazioni seguenti.
   - Table name (Nome tabella) – **`lambda-dynamodb-stream`**
   - Primary key (Chiave primaria) – **`id`** (string)
4. Scegliere Create (Crea).

Per abilitare i flussi

1. Aprire la console DynamoDB.
2. Scegliere Tables (Tabelle).
3. Seleziona la tabella lambda-dynamodb-stream.
4. In Exports and streams (Esportazioni e flussi), scegliere DynamDB stream details (Dettagli del flusso di Dynamo DB).
5. Scegliere Turn On (Attiva).
6. Per Tipo di visualizzazione, scegli Solo attributi chiave.
7. Scegli Attiva il flusso.

Prendi nota dell'ARN del flusso. Questa operazione è necessaria nella fase successiva quando si associa il flusso alla funzione Lambda. Per ulteriori informazioni sull'attivazione dei flussi, consultare Acquisizione dell'attività sulla tabella tramite DynamoDB Streams.

Aggiungi una fonte di eventi in AWS Lambda

Creare una mappatura dell'origine eventi in AWS Lambda. Questa mappatura dell'origine eventi associa il flusso DynamoDB alla funzione Lambda. Dopo aver creato questa mappatura delle sorgenti degli eventi, AWS Lambda inizia il polling dello stream.

Eseguire il seguente comando AWS CLI `create-event-source-mapping`. Dopo l'esecuzione del comando, annotare l'UUID. Questo UUID è necessario per fare riferimento alla mappatura delle origini eventi nei comandi, ad esempio quando si elimina tale mappatura.

```
aws lambda create-event-source-mapping --function-name ProcessDynamoDBRecords \
    --batch-size 100 --starting-position LATEST --event-source DynamoDB-stream-arn
```

Questa procedura crea una mappatura tra il flusso DynamoDB specificato e la funzione Lambda. È possibile associare un flusso DynamoDB a più funzioni Lambda e associare la stessa funzione Lambda a più flussi. Tuttavia, le funzioni Lambda condivideranno il throughput di lettura per il flusso che condividono.

Mediante l'esecuzione del comando riportato di seguito è possibile ottenere l'elenco di mappature delle origini eventi.

```
aws lambda list-event-source-mappings
```

L'elenco restituisce tutte le mappature delle origini eventi create e per ciascuna mappatura mostra, tra l'altro, il valore `LastProcessingResult`. Questo campo è utilizzato per fornire un messaggio informativo nel caso in cui vengano riscontrati problemi. Valori come `No records processed` (indica che non AWS Lambda è stato avviato il polling o che non ci sono record nello stream) e `OK` (indica che i record dallo stream sono AWS Lambda stati letti correttamente e che è stata richiamata la funzione Lambda) indicano che non ci sono problemi. Se vengono riscontrati problemi, si riceve un messaggio di errore.

Se disponi di una grande quantità di mappature delle origini eventi, utilizza il parametro funzione-nome per restringere i risultati.

```
aws lambda list-event-source-mappings --function-name ProcessDynamoDBRecords
```

Eseguire il test della configurazione

Metti alla prova l'esperienza. end-to-end Quando vengono eseguiti gli aggiornamenti delle tabelle, DynamoDB scrive record di eventi nel flusso. Quando AWS Lambda esegue il polling del flusso, rileva nuovi record nel flusso e richiama la funzione Lambda per proprio conto passando eventi alla funzione.

1. Nella console di DynamoDB, aggiungi, aggiorna ed elimina elementi dalla tabella. DynamoDB scrive record di queste operazioni nel flusso.
2. AWS Lambda esegue il polling dello stream e quando rileva aggiornamenti allo stream richiama la funzione Lambda passando i dati degli eventi che trova nello stream.
3. La tua funzione viene eseguita e crea registri in Amazon CloudWatch. Puoi verificare i log riportati nella CloudWatch console Amazon.

Fasi successive

Questo tutorial ti ha mostrato le basi dell'elaborazione degli eventi di flusso DynamoDB con Lambda. Per i carichi di lavoro di produzione, prendi in considerazione l'implementazione della logica di risposta parziale in batch per gestire gli errori dei singoli record in modo più efficiente. L'utilità di elaborazione batch di Powertools for AWS Lambda è disponibile in Python TypeScript, .NET e Java e fornisce una soluzione robusta per questo, gestendo automaticamente la complessità delle risposte parziali in batch e riducendo il numero di tentativi per i record elaborati correttamente.

Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili ai tuoi. Account AWS

Per eliminare la funzione Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **`confirm`** nel campo di immissione del testo, quindi scegli Elimina.

Come eliminare il ruolo di esecuzione

1. Aprire la pagina Ruoli della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

Per eliminare la tabella DynamoDB

1. Aprire la pagina Tables (Tabelle) della console DynamoDB.
2. Selezionare la tabella creata.
3. Scegliere Delete (Elimina).
4. Immettere **`delete`** nella casella di testo.
5. Seleziona Delete Table (Elimina tabella).

Elaborare gli eventi del ciclo di vita di Amazon EC2 con una funzione Lambda

Puoi utilizzare AWS Lambda per elaborare gli eventi del ciclo di vita da Amazon Elastic Compute Cloud e gestire le risorse Amazon EC2. Amazon EC2 invia gli eventi ad Amazon EventBridge (Eventi CloudWatch) per eventi del ciclo di vita, ad esempio quando un'istanza cambia stato, quando viene completata una snapshot del volume di Amazon Elastic Block Store o quando un'istanza spot è pianificata per essere terminata. È possibile configurare EventBridge (CloudWatch Events) per inoltrare tali eventi a una funzione Lambda per l'elaborazione.

EventBridge (CloudWatch Events) richiama la funzione Lambda in modo asincrono con il documento di evento da Amazon EC2.

Example Evento del ciclo di vita dell'istanza

```
{
    "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"
    }
}
```

Per informazioni dettagliate sulla configurazione degli eventi, consulta Richiamo di una funzione Lambda su una pianificazione. Per una funzione di esempio che elabora le notifiche di snapshot Amazon EBS, consulta Pianificatore EventBridge per Amazon EBS.

È inoltre possibile utilizzare l'SDK AWS per gestire istanze e altre risorse con l'API Amazon EC2.

Concessione di autorizzazioni a EventBridge (Eventi CloudWatch)

Per elaborare gli eventi del ciclo di vita da Amazon EC2, EventBridge (CloudWatch Events) deve disporre dell'autorizzazione per richiamare la funzione. Questa autorizzazione proviene dalla policy basata sulle risorse della funzione. Se si utilizza la console EventBridge (CloudWatch Events) per configurare un trigger di evento, la console aggiorna la policy basata sulle risorse per tuo conto. In caso contrario, aggiungere un'istruzione come la seguente:

Example Dichiarazione della policy basata sulle risorse per le notifiche del ciclo di vita di 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/*"
    }
  }
}
```

Per aggiungere un'istruzione, utilizzare il comando `add-permission` della AWS CLI.

```
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/*'
```

Se la funzione utilizza l'SDK AWS per gestire le risorse Amazon EC2, aggiungere le autorizzazioni Amazon EC2 al ruolo di esecuzione della funzione.

Elaborare le richieste di Application Load Balancer con Lambda

È possibile utilizzare una funzione Lambda per elaborare le richieste di un Application Load Balancer. Elastic Load Balancing supporta le funzioni Lambda come target per un Application Load Balancer. Utilizza le regole per il sistema di bilanciamento del carico per indirizzare le richieste HTTP a una funzione in base ai valori del percorso o dell'intestazione. Elabora la richiesta e restituisce una risposta HTTP dalla funzione Lambda.

Elastic Load Balancing richiama la funzione Lambda in modo sincrono con un evento contenente il corpo della richiesta e i metadati.

Example Evento di richiesta 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
}
```

La funzione elabora l'evento e restituisce un documento di risposta al sistema di bilanciamento del carico in JSON. Elastic Load Balancing converte il documento in una risposta HTTP di esito positivo o negativo e la restituisce all'utente.

Example Formato del documento di risposta

```
{
    "statusCode": 200,
    "statusDescription": "200 OK",
    "isBase64Encoded": False,
    "headers": {
        "Content-Type": "text/html"
    },
    "body": "<h1>Hello from Lambda!</h1>"
}
```

Per configurare un Application Load Balancer come trigger di funzione, concedi l'autorizzazione Elastic Load Balancing per eseguire la funzione, crea un gruppo di destinazioni che instradi le richieste alla funzione e aggiungi una regola al sistema di bilanciamento del carico che invia le richieste al gruppo di destinazioni.

Utilizza il comando `add-permission` per aggiungere un'istruzione di autorizzazione alla policy basata sulle risorse della funzione.

```
aws lambda add-permission --function-name alb-function \
--statement-id load-balancer --action "lambda:InvokeFunction" \
--principal elasticloadbalancing.amazonaws.com
```

Verrà visualizzato l'output seguente:

```
{
    "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\"}"
}
```

Per istruzioni su come configurare il gruppo di destinazione e il listener di Application Load Balancer, consulta funzioni Lambda come target nella Guida per l'utente dei sistemi Application Load Balancer.

Gestore di eventi di Powertools per Lambda AWS

Il gestore di eventi del toolkit Powertools for AWS Lambda fornisce routing, middleware, configurazione CORS, generazione di specifiche OpenAPI, convalida delle richieste, gestione degli errori e altre funzioni utili per la scrittura di funzioni Lambda richiamate da un Application Load Balancer. L'utilità Event Handler è disponibile per Python. Per ulteriori informazioni, consulta l'API REST di Event Handler nella documentazione di 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)
```

Richiamo di una funzione Lambda su una pianificazione

Pianificatore Amazon EventBridge è un pianificatore serverless che consente di creare, eseguire e gestire attività da un unico servizio gestito centralizzato. Con il Pianificatore EventBridge puoi creare pianificazioni utilizzando le espressioni cron e rate per schemi ricorrenti o configurare invocazioni una tantum. Puoi configurare finestre temporali flessibili per la consegna, definire limiti per nuovi tentativi e impostare il tempo massimo di conservazione per eventi non elaborati.

Quando configuri il pianificatore EventBridge con Lambda, questo richiama la funzione Lambda in modo asincrono. Questa pagina descrive l'utilizzo del Pianificatore EventBridge per richiamare una funzione Lambda su una pianificazione.

Configurare il ruolo di esecuzione

Quando crei una nuova pianificazione, il PIanificatore EventBridge deve disporre dell'autorizzazione per richiamare automaticamente l'operazione dell'API di destinazione. Concedi queste autorizzazioni al Pianificatore EventBridge utilizzando un ruolo di esecuzione. La policy di autorizzazione collegata al ruolo di esecuzione della pianificazione definisce le autorizzazioni necessarie. Tali autorizzazioni dipendono dall'API di destinazione che deve essere richiamata dal Pianificatore EventBridge.

Quando utilizzi la console del Pianificatore EventBridge per creare una pianificazione, come nella procedura seguente, il Pianificatore EventBridge configura automaticamente un ruolo di esecuzione in base alla destinazione selezionata. Per creare una pianificazione utilizzando uno degli SDK del Pianificatore EventBridge, AWS CLI o CloudFormation, devi disporre di un ruolo di esecuzione esistente che conceda le autorizzazioni richieste dal Pianificatore EventBridge per richiamare una destinazione. Per ulteriori informazioni sull'impostazione manuale di un ruolo di esecuzione per la pianificazione, consulta Configurazione di un ruolo di esecuzione nella Guida per l'utente di Pianificatore EventBridge.

Creare una pianificazione.

Per creare una pianificazione utilizzando la console

1. Apri la console del Pianificatore Amazon EventBridge all'indirizzo https://console.aws.amazon.com/scheduler/home.
2. Nella pagina Pianificazioni, scegli Crea pianificazione.
3. Nella pagina Specifica i dettagli della pianificazione, nella sezione Nome e descrizione della pianificazione, effettua le seguenti operazioni:

a. Per Nome pianificazione, inserisci un nome per la pianificazione. Ad esempio, **`MyTestSchedule`**.

b. (Facoltativo) Per Descrizione, inserisci una descrizione per la pianificazione. Ad esempio, **`My first schedule`**.

c. Per Gruppo di pianificazioni, scegli un gruppo di pianificazioni dall'elenco a discesa. Se non hai un gruppo, scegli predefinito. Per creare un gruppo di pianificazioni, scegli crea la tua pianificazione.

   I gruppi di pianificazione vengono utilizzati per aggiungere tag a gruppi di pianificazioni.

4. • Scegli le opzioni di pianificazione.

Ricorrenza	Esegui questa operazione…	
Pianificazione una tantum Una pianificazione unica richiama una destinazione solo una volta alla data e all'ora specificate.	Per Data e ora, effettua le seguenti operazioni: • Inserisci una data valida in formato `YYYY/MM/DD` . • Inserisci un timestamp in formato `hh:mm` 24 ore. • Per Fuso orario, scegli il fuso orario.	
Pianificazione ricorrente Una pianificazione ricorrente richiama una destinazione con una frequenza specificata utilizzando un'espressione cron o un'espressione rate.	a. Per Tipo di pianificazione, esegui una delle seguenti operazioni: • Per utilizzare un'espressione Cron per definire la pianificazione, scegli Pianificazione basata su cron e immetti l'espressione Cron.	

Ricorrenza	Esegui questa operazione…	
	• Per utilizzare un'espressione di frequenza per definire la pianificazione, scegli Pianificazione basata su frequenza e inserisci l'espressione di frequenza. Per ulteriori informazioni sulle espressioni Cron e rate, consulta Tipi di pianificazione nel Pianificatore EventBridge nella Guida per l'utente di Pianificatore Amazon EventBridge. b. Per Finestra temporale flessibile, scegli Disattivata per disattivare l'opzione o scegli una delle finestre temporali predefinite. Ad esempio, se scegli 15 minuti e imposti una pianificazione ricorrente per il richiamo della destinazione ogni ora, la pianificazione viene eseguita entro 15 minuti dall'inizio di ogni ora.	

5. (Facoltativo) Se hai scelto Pianificazione ricorrente nel passaggio precedente, nella sezione Intervallo di tempo effettua le seguenti operazioni:

   a. Per Fuso orario, scegli un fuso orario.

   b. Per Data e ora di inizio, inserisci una data valida in formato `YYYY/MM/DD`, quindi specifica un timestamp in formato `hh:mm` 24 ore.

   c. Per Data e ora di fine, inserisci una data valida in formato `YYYY/MM/DD`, quindi specifica un timestamp in formato `hh:mm` 24 ore.

6. Scegli Next (Successivo).

7. Nella pagina Seleziona destinazione, scegli l'operazione dell'API AWS richiamata dal Pianificatore EventBridge:

   a. Scegli Richiama AWS Lambda.

   b. Nella sezione Richiama, seleziona una funzione o scegli Crea una nuova funzione Lambda.

   c. (Facoltativo) Inserisci un payload JSON. Se non inserisci un payload, il Pianificatore EventBridge utilizza un evento vuoto per richiamare la funzione.

8. Scegli Next (Successivo).

9. Nella pagina Settings (Impostazioni), eseguire le operazioni descritte di seguito.

   a. Per attivare la pianificazione, in Stato della pianificazione, attiva Abilita pianificazione.

   b. Per configurare una policy di ripetizione per la tua pianificazione, in Policy di ripetizione e coda DLQ (Dead-Letter Queue) effettua le seguenti operazioni:

      - Attiva/disattiva Riprova.
      - Per Età massima dell'evento, inserisci il numero massimo di ore e minuti per cui il Pianificatore EventBridge deve conservare un evento non elaborato.
      - La durata massima è 24 ore.
      - Per Numero massimo di tentativi, inserisci il numero massimo di volte che il Pianificatore EventBridge ritenta la pianificazione se la destinazione restituisce un errore.

        Il valore massimo è 185 tentativi.

      Con le policy per nuovi tentativi, se una pianificazione non riesce a richiamare la sua destinazione, il Pianificatore EventBridge esegue nuovamente la pianificazione. Se

configurato, è necessario impostare il tempo di conservazione massimo e i nuovi tentativi per la pianificazione.

c. Scegli la posizione in cui il Pianificatore EventBridge deve archiviare gli eventi non consegnati.

Opzione Dead-letter queue (DLQ)	Esegui questa operazione…
Non conservare	Scegliere None (Nessuno).
Memorizza l'evento nello stesso Account AWS in cui crei la pianificazione	a. Scegli Seleziona una coda Amazon SQS nel mio Account AWS come DLQ. b. Scegli il nome della risorsa Amazon (ARN) della coda di Amazon SQS.
Memorizza l'evento in un Account AWS diverso da quello in cui crei la pianificazione	a. Scegli Specifica una coda Amazon SQS in un altro Account AWS come DLQ. b. Inserisci il nome della risorsa Amazon (ARN) della coda di Amazon SQS.

d. Per utilizzare una chiave gestita dal cliente per crittografare l'input di destinazione, in Crittografia scegli Personalizza le impostazioni di crittografia (avanzate).

   Se scegli questa opzione, inserisci l'ARN di una chiave KMS esistente scegli Crea una AWS KMS key per accedere alla console AWS KMS. Per ulteriori informazioni sulla modalità con cui il Pianificatore EventBridge esegue la crittografia dei dati inattivi, consulta Crittografia a riposo nella Guida per l'utente di Pianificatore Amazon EventBridge.

e. Affinché il Pianificatore EventBridge crei automaticamente un nuovo ruolo di esecuzione, scegli Crea nuovo ruolo per questa pianificazione. Inserisci, quindi, un nome per Nome

ruolo. Se scegli questa opzione, il Pianificatore EventBridge collega al ruolo le autorizzazioni necessarie per la destinazione basata sul modello.

10. Scegli Next (Successivo).
11. Nella pagina Rivedi e crea pianificazione, rivedi i dettagli della pianificazione. In ogni sezione, scegli Modifica per tornare a tale passaggio e modificarne i dettagli.
12. Scegli Crea pianificazione.

    Puoi visualizzare un elenco delle pianificazioni nuove ed esistenti nella pagina Pianificazioni. Nella colonna Stato, accertati che la nuova pianificazione sia Abilitata.

Per accertarti che il Pianificatore EventBridge abbia richiamato la funzione, controlla i Amazon CloudWatch Logs della funzione.

Risorse correlate

Per ulteriori informazioni sul Pianificatore EventBridge, consulta:

- Guida per l'utente di Pianificatore EventBridge
- Riferimento all'API del Pianificatore EventBridge
- Prezzi del Pianificatore EventBridge

Uso di AWS Lambda con AWS IoT

AWS IoT fornisce una comunicazione sicura tra i dispositivi connessi a Internet (come i sensori) e AWS Cloud. Questo consente raccogliere, archiviare e analizzare i dati di telemetria da più dispositivi.

È possibile creare regole AWS IoT affinché i dispositivi interagiscano con i Servizi AWS. AWS IoT Rules Engine fornisce un linguaggio basato su SQL, per selezionare i dati dai payload dei messaggi e inviare i dati ad altri servizi, ad esempio Amazon S3, Amazon DynamoDB e AWS Lambda. Si definisce una regola per richiamare una funzione Lambda quando si desidera richiamare un altro servizio AWS o un servizio di terze parti.

Quando un messaggio IoT in ingresso attiva la regola, AWS IoTrichiama la funzione Lambda in modo asincrono e passa i dati dal messaggio IoT alla funzione.

L'esempio seguente mostra una lettura dell'umidità da un sensore serra. I valori di riga e pos identificano la posizione del sensore. Questo evento di esempio si basa sul tipo serra in Tutorial sulle regole AWS IoT.

Example AWS IoTEvento messaggio di

```
{
    "row" : "10",
    "pos" : "23",
    "moisture" : "75"
}
```

Per le chiamate asincrone, Lambda inserisce in una coda i messaggi e i tentativi se la funzione restituisce un errore. Configura la tua funzione con una destination per mantenere gli eventi che la tua funzione non è in grado di elaborare.

È necessario concedere l'autorizzazione al servizio AWS IoT per richiamare la funzione Lambda. Utilizza il comando `add-permission` per aggiungere un'istruzione di autorizzazione alla policy basata sulle risorse della funzione.

```
aws lambda add-permission --function-name my-function \
--statement-id iot-events --action "lambda:InvokeFunction" --principal
 iot.amazonaws.com
```

Verrà visualizzato l'output seguente:

```
{
    "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\"}"
}
```

Per ulteriori informazioni su come utilizzare Lambda con AWS IoT, consulta Creazione di una regola AWS Lambda.

Scopri come Lambda elabora i record di Flusso di dati Amazon Kinesis.

È possibile usare una funzione Lambda per elaborare i record in un flusso di dati Amazon Kinesis. È possibile mappare una funzione Lambda a un consumer a throughput condiviso (iteratore standard) del flusso di dati Kinesis o a un consumer di throughput dedicato con fan-out avanzato. Per gli iteratori standard, Lambda interroga ogni shard nel flusso Kinesis alla ricerca di record utilizzando il protocollo HTTP. La mappatura dell'origine eventi condivide il throughput di lettura con altri utenti dello shard.

Per informazioni dettagliate sui flussi di dati Kinesis, consulta Lettura dei dati da Flusso di dati Amazon Kinesis.

> **Note**
>
> Kinesis addebita dei costi per ogni shard, per il fan-out avanzato e per i dati letti dal flusso. Per i dettagli sui prezzi, consulta Prezzi di Amazon Kinesis.

Flussi di polling e batching

Lambda legge i record dal flusso di dati e richiama la funzione in modo sincrono con un evento che contiene record di flusso. Lambda legge i record in batch e richiama la funzione per elaborare i record dal batch. Ogni batch contiene registri da una singolo shard/flusso dei dati.

La funzione Lambda è un'applicazione consumer per il flusso di dati. Elabora un batch di record alla volta da ciascuno shard. È possibile mappare una funzione Lambda a un consumer a throughput condiviso (iteratore standard) o a un consumer di throughput dedicato con fan-out avanzato.

- Iteratori standard: Lambda esegue il polling per ogni shard nel flusso Kinesis per i record a una velocità di base di una volta al secondo. Quando sono disponibili più record, Lambda continua l'elaborazione dei batch fino a quando la funzione raggiunge il flusso. La mappatura dell'origine eventi condivide il throughput di lettura con altri utenti dello shard.
- Fan-out avanzato: per ridurre al minimo la latenza e massimizzare il throughput di lettura, è necessario creare un consumer del flusso di dati con fan-out avanzato. I consumatori del fan-out avanzato ottengono una connessione dedicata a ciascuno shard che non ha conseguenze su altre applicazioni che leggono dal flusso. I consumatori del flusso utilizzano HTTP/2 per

ridurre la latenza spingendo record da Lambda a long-lived su una connessione di lunga durata e comprimendo le intestazioni della richiesta. È possibile creare un consumatore di flusso con l'API RegisterStreamConsumer Kinesis.

```
aws kinesis register-stream-consumer \
--consumer-name con1 \
--stream-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream
```

Verrà visualizzato l'output seguente:

```
{
    "Consumer": {
        "ConsumerName": "con1",
        "ConsumerARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream/
consumer/con1:1540591608",
        "ConsumerStatus": "CREATING",
        "ConsumerCreationTimestamp": 1540591608.0
    }
}
```

Per aumentare la velocità con cui la funzione elabora i record, aggiungere gli shard al flusso di dati. Lambda elabora i record in ogni shard in ordine. Interrompe l'elaborazione di record aggiuntivi in uno shard se la funzione restituisce un errore. Con più partizioni, vengono elaborati più batch contemporaneamente e l'impatto di errori in simultanea viene ridotto.

Se la funzione non è in grado di aumentare le dimensioni fino a gestire il numero totale di batch simultanei, richiedere un aumento della quota o riservare la simultaneità per la funzione.

Per impostazione predefinita, Lambda richiama la funzione non appena i record sono disponibili. Se il batch che Lambda legge dall'origine eventi contiene un solo record, Lambda invia solo un record alla funzione. Per evitare di richiamare la funzione con pochi record è possibile, configurando un periodo di batch, chiedere all'origine eventi di memorizzare nel buffer i registri per un massimo di 5 minuti. Prima di richiamare la funzione, Lambda continua a leggere i registri dall'origine eventi fino a quando non ha raccolto un batch completo, fino alla scadenza del periodo di batch o fino a quando il batch non ha raggiunto il limite del payload di 6 MB. Per ulteriori informazioni, consulta Comportamento di batching.

> **Warning**
>
> Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center AWS.

Lambda non attende il completamento di alcuna estensione prima di inviare il batch successivo per l'elaborazione. In altre parole, le estensioni possono continuare a funzionare mentre Lambda elabora il successivo batch di record. Ciò può causare problemi di limitazione in caso di violazione delle impostazioni o dei limiti di simultaneità. Per rilevare se si tratta di un potenziale problema, monitora le tue funzioni e verifica se i parametri di simultaneità per lo strumento di mappatura dell'origine degli eventi sono superiori al previsto. A causa degli intervalli ridotti tra le invocazioni, Lambda potrebbe segnalare brevemente un utilizzo della simultaneità maggiore rispetto al numero di partizioni. Ciò può essere vero anche per le funzioni Lambda senza estensioni.

Configura l'impostazione ParallelizationFactor per elaborare uno shard di un flusso di dati Kinesis con più invocazioni Lambda contemporaneamente. È possibile specificare il numero di batch simultanei di cui Lambda esegue il polling da uno shard da un fattore di parallelizzazione compreso tra da 1 (predefinito) e 10. Ad esempio, impostando `ParallelizationFactor` su 2, puoi disporre al massimo di 200 invocazioni Lambda simultanee per elaborare 100 partizioni di dati Kinesis (anche se nella pratica potresti vedere valori differenti per il parametro `ConcurrentExecutions`). Ciò permette di dimensionare verso l'alto il throughput di elaborazione quando il volume dei dati è volatile e l'`IteratorAge` è alta. Se si aumenta il numero di batch simultanei per shard, Lambda garantisce comunque l'ordine di elaborazione a livello di partizione-chiave.

Puoi anche utilizzare `ParallelizationFactor` con l'aggregazione Kinesis. Il comportamento dello strumento di mappatura dell'origine degli eventi dipende dall'utilizzo o meno di un fan-out avanzato:

- Senza fan-out avanzato: tutti gli eventi all'interno di un evento aggregato devono avere la stessa chiave di partizione. La chiave di partizione deve inoltre corrispondere a quella dell'evento aggregato. Se gli eventi all'interno dell'evento aggregato hanno chiavi di partizione diverse, Lambda non può garantire l'elaborazione ordinata degli eventi per chiave di partizione.
- Con fan-out migliorato: innanzitutto, Lambda decodifica l'evento aggregato nei suoi singoli eventi. L'evento aggregato può avere una chiave di partizione diversa dagli eventi che contiene. Tuttavia,

gli eventi che non corrispondono alla chiave di partizione vengono eliminati e persi. Lambda non elabora questi eventi e non li invia a una destinazione di errore configurata.

Esempio di evento

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"
        }
    ]
}
```

Elaborare i record del flusso di dati Amazon Kinesis con Lambda

Per elaborare i record di Amazon Kinesis Data Streams con Lambda, crea una mappatura delle sorgenti degli eventi Lambda. È possibile mappare una funzione Lambda a un iteratore standard o a un consumatore fan-out avanzato. Per ulteriori informazioni, consulta Flussi di polling e batching.

Creare una mappatura delle sorgenti degli eventi Kinesis

Per richiamare la funzione Lambda con i record dal flusso di dati, crea uno strumento di mappatura dell'origine degli eventi. È possibile creare più mappature delle origini eventi per elaborare gli stessi dati con più funzioni Lambda o per elaborare elementi da più flussi di dati con una singola funzione. Quando si elaborano elementi da più flussi, ogni batch conterrà i record di un solo shard o di un solo flusso.

È possibile configurare gli strumenti di mappatura dell'origine degli eventi per elaborare i record da un flusso in un altro Account AWS. Per ulteriori informazioni, consulta the section called “Mappature tra account”.

Prima di creare uno strumento di mappatura dell'origine degli eventi, devi autorizzare la funzione Lambda a leggere da un flusso di dati Kinesis. Lambda richiede le seguenti autorizzazioni per gestire le risorse correlate al flusso di dati Kinesis:

- kinesis: DescribeStream
- cinesi: DescribeStreamSummary
- cinesi: GetRecords
- cinesi: GetShardIterator
- cinesi: ListShards
- cinesi: SubscribeToShard

La politica AWS gestita AWSLambdaKinesisExecutionRoleinclude queste autorizzazioni. Aggiungi questa policy gestita alla funzione come descritto nella seguente procedura.

> **Note**
>
> - Non è necessaria l'`kinesis:ListStreams`autorizzazione per creare e gestire le mappature delle sorgenti degli eventi per Kinesis. Tuttavia, se crei una mappatura dell'origine degli eventi nella console e non disponi di questa autorizzazione, non sarai in grado di selezionare uno stream Kinesis da un elenco a discesa e la console visualizzerà un errore. Per creare la mappatura delle sorgenti degli eventi, devi inserire manualmente l'Amazon Resource Name (ARN) del tuo stream.
> - Lambda effettua `kinesis:GetRecords` chiamate `kinesis:GetShardIterator` API ogni volta che tenta di ripetere le chiamate non riuscite.

Console di gestione AWS

Per aggiungere le autorizzazioni Kinesis alla tua funzione

1. Apri la pagina Funzioni della console Lambda e scegli la tua funzione.
2. Nella scheda Configurazione scegli Autorizzazioni.
3. Nel riquadro Ruolo di esecuzione, ub Nome del ruolo, scegli il link al ruolo di esecuzione della tua funzione. Questo link apre la pagina del ruolo nella console IAM.
4. Nel riquadro Policy di autorizzazioni, seleziona Aggiungi autorizzazioni, quindi Collega policy.
5. Inserisci **`AWSLambdaKinesisExecutionRole`** nel campo di ricerca.
6. Seleziona la casella di controllo accanto al nome della policy, quindi scegli Aggiungi autorizzazione.

AWS CLI

Per aggiungere le autorizzazioni Kinesis alla tua funzione

- Per aggiungere la policy `AWSLambdaKinesisExecutionRole` al ruolo di esecuzione della funzione, eseguire il comando della CLI sotto riportato:

```
aws iam attach-role-policy \
--role-name MyFunctionRole \
```

```
--policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaKinesisExecutionRole
```

AWS SAM

Per aggiungere le autorizzazioni Kinesis alla tua funzione

- Nella definizione della funzione, aggiungi la proprietà `Policies` come mostrato nell'esempio seguente:

```
Resources:
  MyFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./my-function/
      Handler: index.handler
      Runtime: nodejs24.x
      Policies:
        - AWSLambdaKinesisExecutionRole
```

Dopo aver configurato le autorizzazioni richieste, crea lo strumento di mappatura dell'origine degli eventi.

Console di gestione AWS

Per creare uno strumento di mappatura dell'origine degli eventi Kinesis

1. Apri la pagina Funzioni della console Lambda e scegli la tua funzione.
2. Nel riquadro Panoramica della funzione, scegli Aggiungi trigger.
3. In Configurazione del trigger, per l'origine seleziona Kinesis.
4. Seleziona il flusso Kinesis per il quale desideri creare lo strumento di mappatura dell'origine degli eventi e, facoltativamente, un consumer del tuo flusso.
5. (Facoltativo) Modifica la dimensione del batch, la posizione iniziale e la finestra batch per lo strumento di mappatura dell'origine degli eventi.
6. Scegliere Aggiungi.

Quando crei la mappatura delle sorgenti degli eventi dalla console, il tuo ruolo IAM deve disporre delle autorizzazioni kinesis: e kinesis:. ListStreams ListStreamConsumers

AWS CLI

Per creare uno strumento di mappatura dell'origine degli eventi Kinesis

- Per creare uno strumento di mappatura dell'origine degli eventi Kinesis, eseguire il comando della CLI sotto riportato. Scegli la dimensione del batch e la posizione iniziale in base al tuo caso d'uso.

```
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
```

Per specificare una finestra di batch, aggiungi l'opzione `--maximum-batching-window-in-seconds`. Per ulteriori informazioni sull'utilizzo di questo e di altri parametri, consulta create-event-source-mappingla sezione Command Reference.AWS CLI

AWS SAM

Per creare uno strumento di mappatura dell'origine degli eventi Kinesis

- Nella definizione della funzione, aggiungi la proprietà `KinesisEvent` come mostrato nell'esempio seguente:

```
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
```

Per ulteriori informazioni sulla creazione di una mappatura delle sorgenti di eventi per Kinesis Data Streams in AWS SAM, consulta Kinesis nella Developer Guide.AWS Serverless Application Model

Posizioni di partenza di polling e flussi

Tieni presente che il polling dei flussi durante la creazione e gli aggiornamenti dello strumento di mappatura dell'origine degli eventi alla fine è coerente.

- Durante la creazione dello strumento di mappatura dell'origine degli eventi, potrebbero essere necessari alcuni minuti per l'avvio degli eventi di polling dal flusso.
- Durante gli aggiornamenti dello strumento di mappatura dell'origine degli eventi, potrebbero essere necessari alcuni minuti per l'avvio degli eventi di polling dal flusso.

Questo comportamento implica che se specifichi `LATEST` come posizione iniziale del flusso, lo strumento di mappatura dell'origine degli eventi potrebbe perdere eventi durante la creazione o gli aggiornamenti. Per non perdere alcun evento, specifica la posizione iniziale del flusso come `TRIM_HORIZON` o `AT_TIMESTAMP`.

Creazione di uno strumento di mappatura dell'origine degli eventi multi-account

Flusso di dati Amazon Kinesis supporta le policy basate sulle risorse. Per questo motivo, puoi elaborare i dati inseriti in un flusso in uno Account AWS con una funzione Lambda in un altro account.

Per creare una mappatura dell'origine degli eventi per la tua funzione Lambda utilizzando un flusso Kinesis in un Account AWS altro, devi configurare il flusso utilizzando una policy basata sulle risorse per autorizzare la funzione Lambda a leggere gli elementi. Per informazioni su come configurare lo stream per consentire l'accesso su più account, consulta Condivisione dell'accesso con AWS Lambda funzioni su più account nella guida per sviluppatori di Amazon Kinesis Streams.

Dopo aver configurato il flusso con una policy basata sulle risorse che fornisce alla funzione Lambda le autorizzazioni richieste, crea lo strumento di mappatura dell'origine degli eventi utilizzando uno dei metodi descritti nella sezione precedente.

Se scegli di creare lo strumento di mappatura dell'origine degli eventi utilizzando la console Lambda, incolla l'ARN del tuo flusso direttamente nel campo di input. Se desideri specificare un consumer per il tuo flusso, incollando l'ARN del consumer viene compilato automaticamente il campo del flusso.

Configurazione della risposta batch parziale con il flusso di dati Kinesis e Lambda

Quando si consumano ed elaborano i dati di streaming da un'origine eventi, per impostazione predefinita i Lambda imposta i checkpoint al numero di sequenza più alto di un batch solo quando il batch è riuscito completamente. Lambda tratta tutti gli altri risultati come un fallimento completo e riprova a elaborare il batch fino al limite di tentativi. Per consentire i successi parziali durante l'elaborazione di batch da un flusso, attivare `ReportBatchItemFailures`. Consentire successi parziali può contribuire a ridurre il numero di tentativi su un record, anche se non impedisce del tutto la possibilità di tentativi in un record riuscito.

Per attivare `ReportBatchItemFailures`, includere il valore enum**`ReportBatchItemFailures`** nell'FunctionResponseTypeselenco. Questo elenco indica quali tipi di risposta sono abilitati per la funzione. È possibile configurare questo elenco quando si crea o si aggiorna uno strumento di mappatura dell'origine degli eventi.

> **Note**
>
> Anche quando il codice della funzione restituisce risposte parziali di errore in batch, queste risposte non verranno elaborate da Lambda a meno che la `ReportBatchItemFailures` funzionalità non sia esplicitamente attivata per la mappatura dell'origine degli eventi.

Sintassi di report

Quando si configura la creazione di report sugli errori degli elementi batch, la `StreamsEventResponse` classe viene restituita con un elenco di errori degli articoli batch. È possibile utilizzare un `StreamsEventResponse` oggetto per restituire il numero di sequenza del primo record non riuscito nel batch. È inoltre possibile creare la propria classe personalizzata utilizzando la sintassi di risposta corretta. La seguente struttura JSON mostra la sintassi di risposta richiesta:

```
{
  "batchItemFailures": [
        {
            "itemIdentifier": "<SequenceNumber>"
        }
    ]
}
```

> **Note**
>
> Se l'array `batchItemFailures` contiene più elementi, Lambda utilizza il record con il numero di sequenza più basso come checkpoint. Lambda quindi riprova tutti i record a partire da quel checkpoint.

Condizioni di successo e di errore

Lambda considera un batch come un successo completo se si restituisce uno degli elementi seguenti:

- Una `batchItemFailure` lista vuota
- Un `batchItemFailure` elenco nullo
- Un vuoto `EventResponse`
- Un valore nullo `EventResponse`

Lambda considera un batch come un fallimento completo se si restituisce uno degli elementi seguenti:

- Una stringa vuota `itemIdentifier`
- Un valore nullo `itemIdentifier`
- Un `itemIdentifier` con un nome chiave errato

Lambda esegue nuovi tentativi in seguito ai fallimenti secondo la strategia di tentativi impostata.

Bisezione un batch

Se il richiamo fallisce ed `BisectBatchOnFunctionError` è attivato, il batch viene bisecato a prescindere dalle `ReportBatchItemFailures` impostazioni.

Quando si riceve una risposta di successo parziale del batch ed entrambi `BisectBatchOnFunctionError` e `ReportBatchItemFailures` sono attivati, il batch viene bisecato in corrispondenza del numero di sequenza restituito e Lambda ritenta solo i record rimanenti.

Per semplificare l'implementazione della logica di risposta batch parziale, prendi in considerazione l'utilizzo dell'utilità Batch Processor di Powertools for AWS Lambda, che gestisce automaticamente queste complessità per te.

Ecco alcuni esempi di codice di funzione che restituiscono l'elenco dei messaggi non riusciti IDs nel batch:

.NET

SDK per .NET

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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;
}
```

Segnalazione degli errori degli elementi batch di Kinesis utilizzando 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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
}
```

Utilizzo di Powertools per il processore AWS Lambda batch

L'utilità per il processore batch di Powertools gestisce AWS Lambda automaticamente la logica di risposta parziale in batch, riducendo la complessità dell'implementazione della segnalazione degli errori dei batch. Ecco alcuni esempi di utilizzo del processore batch:

Python

> **Note**
>
> Per esempi completi e istruzioni di configurazione, consultate la documentazione del processore batch.

Elaborazione dei AWS Lambda record di flusso Kinesis Data Streams con processore 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 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**
>
> Per esempi completi e istruzioni di configurazione, consulta la documentazione del processore batch.

Elaborazione dei AWS Lambda record di flusso Kinesis Data Streams con processore batch.

```
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**
>
> Per esempi completi e istruzioni di configurazione, consulta la documentazione del processore batch.

Elaborazione dei AWS Lambda record di flusso Kinesis Data Streams con processore batch.

```
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**
>
> Per esempi completi e istruzioni di configurazione, consulta la documentazione del processore batch.

Elaborazione dei AWS Lambda record di flusso Kinesis Data Streams con processore batch.

```
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;
    }
}
```

Conservare i record batch scartati per un'origine di eventi del flusso di dati Kinesis in Lambda

La gestione degli errori per gli strumenti di mappatura dell'origine degli eventi Kinesis dipende dal fatto che l'errore si verifichi prima che la funzione venga richiamata o durante la chiamata della funzione:

- Prima della chiamata: se una mappatura dell'origine degli eventi Lambda non è in grado di richiamare la funzione a causa di limitazioni o altri problemi, riprova finché i record non scadono o superano l'età massima configurata nella mappatura dell'origine dell'evento (). MaximumRecordAgeInSeconds
- Durante la chiamata: se la funzione viene richiamata ma restituisce un errore, Lambda riprova fino alla scadenza dei record, al superamento dell'età massima () o al raggiungimento della quota di tentativi configurata (MaximumRecordAgeInSeconds). MaximumRetryAttempts Per gli errori di funzione, puoi anche configurare, che divide un batch non riuscito in due batch più piccoli BisectBatchOnFunctionError, isolando i record non validi ed evitando i timeout. La divisione dei batch non consuma la quota di tentativi.

Se le misure di gestione degli errori non riescono, Lambda elimina i record e continua l'elaborazione dei batch dal flusso. Con le impostazioni predefinite, ciò significa che un record errato può bloccare l'elaborazione sullo shard interessata per un massimo di una settimana. Per evitare questa situazione, configura la mappatura dell'origine eventi della funzione con un numero ragionevole di tentativi e un'età massima dei record che sia adatta al caso d'uso.

Configurazione delle destinazioni per le chiamate non riuscite

Per mantenere i record delle chiamate non riuscite allo strumento di mappatura dell'origine degli eventi, aggiungi una destinazione allo strumento di mappatura dell'origine degli eventi della funzione.

Ogni record inviato alla destinazione è un documento JSON contenente i metadati relativi alla chiamata non riuscita. Per le destinazioni Amazon S3, Lambda invia insieme ai metadati anche l'intero record di invocazione. Puoi configurare qualsiasi argomento Amazon SNS, coda Amazon SQS, bucket Amazon S3 o Kafka come destinazione.

Con le destinazioni Amazon S3, puoi utilizzare la funzionalità Notifiche eventi Amazon S3 per ricevere notifiche quando gli oggetti vengono caricati nel bucket S3 di destinazione. Puoi anche configurare Notifiche eventi S3 per richiamare un'altra funzione Lambda per eseguire l'elaborazione automatica su batch non riusciti.

Il tuo ruolo di esecuzione deve avere le autorizzazioni per la destinazione:

- Per una destinazione SQS: sqs: SendMessage
- Per una destinazione SNS: sns:publish
- Per una destinazione S3: s3: e s3: PutObject ListBucket
- Per una destinazione Kafka: kafka-cluster: WriteData

È possibile configurare un argomento di Kafka come destinazione in caso di errore per le mappature delle sorgenti degli eventi Kafka. Quando Lambda non è in grado di elaborare i record dopo estenuanti tentativi o quando i record superano l'età massima, Lambda invia i record non riusciti all'argomento Kafka specificato per un'elaborazione successiva. Fai riferimento a the section called “Destinazione Kafka in caso di errore”.

Se hai abilitato la crittografia con la tua chiave KMS per una destinazione S3, anche il ruolo di esecuzione della funzione deve avere l'autorizzazione a chiamare kms:. GenerateDataKey Se la chiave KMS e la destinazione del bucket S3 si trovano in un account diverso dalla funzione Lambda e dal ruolo di esecuzione, configura la chiave KMS in modo che consideri attendibile il ruolo di esecuzione da consentire. kms: GenerateDataKey

Per configurare una destinazione in caso di errore tramite la console, completa i seguenti passaggi:

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. In Function overview (Panoramica delle funzioni), scegliere Add destination (Aggiungi destinazione).
4. Per Origine, scegli Chiamata allo strumento di mappatura dell'origine degli eventi.

5. Per Strumento di mappatura dell'origine degli eventi, scegli un'origine dell'evento configurata per questa funzione.
6. Per Condizione, seleziona In caso di errore. Per le chiamate allo strumento di mappatura dell'origine degli eventi, questa è l'unica condizione accettata.
7. Per Tipo di destinazione, scegli il tipo di destinazione a cui Lambda deve inviare i record di chiamata.
8. Per Destination (Destinazione), scegliere una risorsa.
9. Scegli Save (Salva).

Puoi anche configurare una destinazione in caso di errore utilizzando (). AWS Command Line Interface AWS CLI Ad esempio, il create-event-source-mappingcomando seguente aggiunge una mappatura dell'origine degli eventi con una destinazione SQS in caso di errore a: `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"}}'
```

Il update-event-source-mappingcomando seguente aggiorna una mappatura dell'origine degli eventi per inviare i record di chiamata non riuscita a una destinazione SNS dopo due tentativi o se i record risalgono a più di un'ora.

```
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"}}'
```

Le impostazioni aggiornate sono applicate in modo asincrono e non sono riflesse nell'output fino al completamento del processo. Utilizza il comando get-event-source-mapping per visualizzare lo stato corrente.

Per rimuovere una destinazione, fornisci una stringa vuota come argomento del parametro `destination-config`:

```
aws lambda update-event-source-mapping \
```

```
--uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b \
--destination-config '{"OnFailure": {"Destination": ""}}'
```

Best practice di sicurezza per destinazioni Amazon S3

L'eliminazione di un bucket S3 configurato come destinazione senza rimuovere la destinazione dalla configurazione della funzione può creare un rischio per la sicurezza. Se un altro utente conosce il nome del bucket di destinazione, può ricreare il bucket nel proprio Account AWS. I record delle invocazioni non riuscite verranno inviati al relativo bucket, esponendo potenzialmente i dati della tua funzione.

> **Warning**
>
> Per garantire che i record di invocazione della tua funzione non possano essere inviati a un bucket S3 in un altro Account AWS, aggiungi una condizione al ruolo di esecuzione della funzione che limiti le autorizzazioni ai bucket del tuo account. `s3:PutObject`

Di seguito viene illustrato un esempio di policy IAM che limita le autorizzazioni `s3:PutObject` della funzione ai bucket presenti nell'account. Questa policy fornisce inoltre a Lambda l'autorizzazione `s3:ListBucket` necessaria per utilizzare un bucket S3 come destinazione.

```
{
    "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"
                }
            }
```

```
        }
    ]
}
```

Per aggiungere una politica di autorizzazioni al ruolo di esecuzione della tua funzione utilizzando Console di gestione AWS o AWS CLI, consulta le istruzioni nelle seguenti procedure:

Console

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Seleziona la funzione Lambda di cui si desidera modificare il ruolo di esecuzione.
3. Nella scheda Configurazione scegli Autorizzazioni.
4. Nella scheda Ruolo di esecuzione, seleziona il nome del ruolo della funzione per aprire la pagina della console IAM del ruolo.
5. Aggiungi una policy di autorizzazioni di base al ruolo completando le seguenti operazioni:
    a. Nel riquadro Policy di autorizzazioni, scegli Aggiungi autorizzazioni, poi Crea policy in linea.
    b. Nell'editor delle policy, seleziona JSON.
    c. Incolla la policy che desideri aggiungere nell'editor (sostituendo il codice JSON esistente), quindi scegli Avanti.
    d. In Dettagli della policy, specifica un nome per la policy.
    e. Scegli Crea policy.

AWS CLI

Per aggiungere una policy di autorizzazioni al ruolo di esecuzione di una funzione (CLI)

1. Crea un documento di policy JSON con le autorizzazioni richieste e salvalo in una directory locale.
2. Utilizza il comando della CLI `put-role-policy` di IAM per aggiungere le autorizzazioni per il ruolo di esecuzione di una funzione. Esegui il comando seguente dalla directory in cui hai salvato il documento di policy JSON e sostituisci il nome del ruolo, il nome della policy e il documento di policy con i tuoi valori.

```
aws iam put-role-policy \
--role-name my_lambda_role \
--policy-name LambdaS3DestinationPolicy \
--policy-document file://my_policy.json
```

Record di invocazione Amazon SNS e Amazon SQS di esempio

L'esempio seguente mostra ciò che Lambda invia a un argomento SNS o una coda SQS di destinazione per una chiamata non riuscita all'origine dell'evento Kinesis. Poiché Lambda invia solo i metadati per questi tipi di destinazione, utilizza i campi `streamArn`, `shardId`, `startSequenceNumber` e `endSequenceNumber` per ottenere il record originale completo. Tutti i campi mostrati nella proprietà `KinesisBatchInfo` saranno sempre presenti.

```
{
    "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"
    }
}
```

È possibile utilizzare queste informazioni per recuperare i record interessati dal flusso per la risoluzione dei problemi. I record effettivi non sono inclusi, pertanto è necessario elaborare questo record e recuperarli dal flusso prima che scadano e vadano persi.

Record di invocazione Amazon S3 di esempio

L'esempio seguente mostra ciò che Lambda invia a un bucket Amazon S3 per una chiamata non riuscita all'origine eventi Kinesis. Oltre a tutti i campi dell'esempio precedente per le destinazioni SQS e SNS, il campo `payload` contiene il record di chiamata originale come stringa JSON con escape.

```
{
    "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
}
```

L'oggetto S3 contenente il record di invocazione utilizza la seguente convenzione di denominazione:

```
aws/lambda/<ESM-UUID>/<shardID>/YYYY/MM/DD/YYYY-MM-DDTHH.MM.SS-<Random UUID>
```

Implementazione dell'elaborazione di Flusso di dati Kinesis stateful in Lambda

Le funzioni Lambda possono eseguire applicazioni di elaborazione di flussi continui. Un flusso rappresenta dati illimitati che scorrono continuamente attraverso l'applicazione. Per analizzare le informazioni da questo input di aggiornamento continuo, è possibile associare i record inclusi utilizzando una finestra definita in termini di tempo.

Le finestre a cascata sono finestre temporali distinte che si aprono e si chiudono a intervalli regolari. Per impostazione predefinita, le invocazioni Lambda sono senza stato: non è possibile utilizzarle per l'elaborazione dei dati tra più invocazioni continue senza un database esterno. Tuttavia, con la finestra a cascata, è possibile mantenere il proprio stato tra le invocazioni. Questo stato contiene il risultato aggregato dei messaggi precedentemente elaborati per la finestra corrente. Lo stato può essere un massimo di 1 MB per shard. Se supera quella dimensione, Lambda termina la finestra in anticipo.

Ogni record in un flusso appartiene a una finestra specifica. Lambda elaborerà ogni record almeno una volta, ma non garantisce che ogni record venga elaborato una sola volta. In rari casi, come nel caso della gestione degli errori, alcuni record potrebbero essere elaborati più di una volta. La prima volta i record vengono sempre elaborati in ordine. Se i record vengono elaborati più di una volta, possono essere elaborati fuori ordine.

Aggregazione ed elaborazione

La funzione gestita dall'utente viene richiamata sia per l'aggregazione che per l'elaborazione dei risultati finali di tale aggregazione. Lambda aggrega tutti i record ricevuti nella finestra. È possibile ricevere questi record in più batch, ciascuno come richiamo separato. Ogni richiamo riceve uno stato. Pertanto, quando si utilizzano finestre a cascata, la risposta della funzione Lambda deve contenere una proprietà di `state`. Se la risposta non contiene una proprietà di `state`, Lambda la considera un'invocazione non riuscita. Per soddisfare questa condizione, la funzione può restituire un oggetto `TimeWindowEventResponse`, che presenta la seguente forma JSON:

Example **`TimeWindowEventResponse`** valori

```
{
    "state": {
        "1": 282,
        "2": 715
    },
    "batchItemFailures": []
```

```
}
```

> **Note**
>
> Per le funzioni Java, si consiglia di utilizzare una `Map<String, String>` per rappresentare lo stato.

Alla fine della finestra, il flag `isFinalInvokeForWindow` è impostato `true` per indicare che questo è lo stato finale e che è pronto per l'elaborazione. Dopo l'elaborazione, la finestra viene completata e il richiamo finale viene completato e quindi lo stato viene eliminato.

Al termine della finestra, Lambda utilizza l'elaborazione finale per le operazioni sui risultati dell'aggregazione. L'elaborazione finale viene richiamata in modo sincrono. Dopo il richiamo riuscito, la funzione controlla il numero di sequenza e l'elaborazione del flusso continua. Se il richiamo non ha esito positivo, la funzione Lambda sospende l'ulteriore elaborazione fino a quando non viene eseguito correttamente il richiamo.

Example KineesistimeWindowEvent

```
{
    "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
}
```

Configurazione

È possibile configurare le finestre a cascata quando si crea o si aggiorna un mapping di origini di eventi. Per configurare una finestra a cascata, specifica la finestra in secondi (TumblingWindowInSeconds). Il comando di esempio seguente AWS Command Line Interface (AWS CLI) crea un mapping di origini di eventi di streaming con una finestra a cascata di 120 secondi. La funzione Lambda definita per l'aggregazione e l'elaborazione è denominata `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 determina i limiti delle finestre a cascata in base al momento in cui i record sono stati inseriti nel flusso. Tutti i record dispongono di un timestamp approssimativo che Lambda utilizza nelle determinazioni dei limiti.

Le aggregazioni delle finestre a cascata non supportano il risharding. Quando lo shard viene terminato, Lambda considera la finestra chiusa e gli shard secondari iniziano la propria finestra in uno stato nuovo. Quando non vengono aggiunti nuovi record alla finestra corrente, Lambda attende fino a due minuti prima di presumere che la finestra sia terminata. Ciò aiuta a garantire che la funzione legga tutti i record nella finestra corrente, anche se i record vengono aggiunti a intermittenza.

Le finestre a cascata supportano completamente le policy di ripetizione dei tentativi esistenti `maxRetryAttempts` e `maxRecordAge`.

Example Handler.py: aggregazione ed elaborazione

La seguente funzione Python mostra come aggregare e quindi elaborare lo stato finale:

```
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}
```

Parametri Lambda per gli strumenti di mappatura dell'origine degli eventi di un flusso di dati Amazon Kinesis

Tutti gli strumenti di mappatura dell'origine degli eventi Lambda condividono le stesse operazioni API CreateEventSourceMapping e UpdateEventSourceMapping. Tuttavia, solo alcuni dei parametri si applicano a Kinesis.

Parametro	Obbligatorio	Predefinito	Note
BatchSize	N	100	Massimo: 10.000.

Parametro	Obbligatorio	Predefinito	Note
BisectBatchOnFunctionError	N	false	nessuno
DestinationConfig	N	N/D	Una destinazione coda Amazon SQS o argomento Amazon SNS per i record scartati. Per ulteriori informazioni, consulta Configurazione delle destinazioni per le chiamate non riuscite.
Abilitazione di	N	true	nessuno
EventSourceArn	Y	N/D	L'ARN del flusso dei dati o di un consumer di flusso.
FunctionName	Y	N/D	nessuno
FunctionResponseTypes	N	N/D	Per consentire alla funzione di segnalare errori specifici in un batch, includi il valore `ReportBatchItemFailures` in `FunctionResponseTypes` . Per ulteriori informazioni, consulta Configurazione della risposta batch parziale con il flusso di dati Kinesis e Lambda.

Parametro	Obbligatorio	Predefinito	Note
MaximumBatchingWindowInSeconds	N	0	nessuno
MaximumRecordAgeInSeconds	N	-1	-1 sta per infinito: Lambda non scarta i record (le impostazioni di conservazione dei dati del flusso di dati Kinesis sono ancora valide) Minimo: -1 Massimo: 604.800
MaximumRetryAttempts	N	-1	-1 sta per infinito: i record non riusciti vengono ripetuti fino alla scadenza del record Minimo: -1 Massimo: 10.000.
ParallelizationFactor	N	1	Maximum: 10
StartingPosition	Y	N/D	AT_TIMESTAMP, TRIM_HORIZON o LATEST

Parametro	Obbligatorio	Predefinito	Note
StartingPositionTimestamp	N	N/D	Valido solo se StartingPosition è impostato su AT_TIMESTAMP. Il tempo da cui avviare la lettura, in secondi di tempo Unix.
TumblingWindowInSeconds	N	N/D	Minimo: 0 Massimo: 900

Utilizzo del filtro eventi con un'origine eventi Kinesis

Puoi utilizzare il filtraggio degli eventi per controllare quali record di un flusso o di una coda Lambda invia alla funzione. Per informazioni generali sul funzionamento del filtraggio eventi, consulta the section called “Filtro eventi”.

In questa sezione viene descritto il filtraggio degli eventi per le origini eventi Kinesis.

> **Note**
>
> Le mappature delle sorgenti degli eventi Kinesis supportano solo il filtraggio sulla chiave. `data`

Argomenti

- Nozioni di base sul filtro di eventi Kinesis
- Filtraggio dei record aggregati Kinesis

Nozioni di base sul filtro di eventi Kinesis

Supponiamo che un produttore stia inserendo dati in formato JSON nel flusso di dati Kinesis. Un record di esempio sarebbe simile al seguente, con i dati JSON convertiti in una stringa codificata Base64 nel campo `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"
}
```

Fintantoché i dati che il produttore inserisce nel flusso sono JSON validi, puoi utilizzare il filtraggio degli eventi per filtrare i record utilizzando la chiave `data`. Supponiamo che un produttore stia inserendo dati in formato JSON nel flusso di dati Kinesis.

```
{
    "record": 12345,
    "order": {
        "type": "buy",
        "stock": "ANYCO",
        "quantity": 1000
        }
}
```

Per filtrare solo i record in cui il tipo di ordine è "acquista", l'oggetto `FilterCriteria` dovrebbe avere la struttura seguente.

```
{
    "Filters": [
        {
            "Pattern": "{ \"data\" : { \"order\" : { \"type\" : [ \"buy\" ] } } }"
        }
    ]
```

```
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
    "data": {
        "order": {
            "type": [ "buy" ]
            }
      }
}
```

Puoi aggiungere il filtro utilizzando la console, la AWS CLI o un modello AWS SAM.

Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in Collegamento dei criteri di filtro a una mappatura dell'origine evento (console) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "data" : { "order" : { "type" : [ "buy" ] } } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtraggio utilizzando l'AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
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\" ] } } }"}]}'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"order\" : { \"type
\" : [ \"buy\" ] } } }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML dell'origine degli eventi.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : { "order" : { "type" : [ "buy" ] } } }'
```

Per filtrare correttamente gli eventi da origini Kinesis, sia il campo dati sia i criteri di filtraggio per il campo dati devono essere in un formato JSON valido. Se uno dei due campi non è in un formato JSON valido, Lambda rilascia il messaggio o genera un'eccezione. La tabella seguente riepiloga il comportamento specifico:

Formato dei dati in entrata	Formato del modello di filtro per le proprietà di dati	Operazione risultante
JSON valido	JSON valido	Filtri Lambda in base ai criteri di filtro.
JSON valido	Nessun modello di filtro per le proprietà dei dati	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.
JSON valido	Non-JSON	Lambda genera un'eccezione al momento della creazione o dell'aggiornamento della mappatura dell'origine evento. Il modello di filtro per le proprietà dei dati deve essere in un formato JSON valido.
Non-JSON	JSON valido	Lambda rilascia il registro.
Non-JSON	Nessun modello di filtro per le proprietà dei dati	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.

Formato dei dati in entrata	Formato del modello di filtro per le proprietà di dati	Operazione risultante
Non-JSON	Non-JSON	Lambda genera un'eccezione al momento della creazione o dell'aggiornamento della mappatura dell'origine evento. Il modello di filtro per le proprietà dei dati deve essere in un formato JSON valido.

Filtraggio dei record aggregati Kinesis

Con Kinesis, puoi aggregare più record in un unico record flusso di dati Kinesis per aumentare il throughput dei dati. Lambda può applicare criteri di filtraggio ai record aggregati solo quando si utilizza il fan-out avanzato di Kinesis. Il filtraggio dei record aggregati con Kinesis standard non è supportato. Quando utilizzi il fan-out avanzato, configuri un consumatore Kinesis a throughput dedicato che funga da trigger per la funzione Lambda. Lambda filtra quindi i record aggregati e passa solo i record che soddisfano i criteri di filtraggio.

Per ulteriori informazioni sull'aggregazione dei record Kinesis, consulta la sezione Aggregazione nella pagina Concetti chiave di Kinesis Producer Library (KPL). Per ulteriori informazioni sull'utilizzo di Lambda con il fan-out avanzato di Kinesis, consulta la pagina Aumento delle prestazioni di elaborazione dei flussi in tempo reale con il fan-out avanzato dei Flussi di dati Amazon Kinesis e AWS Lambda nel blog di calcolo AWS.

Tutorial: Utilizzo di Lambda con Flusso di dati Kinesis

In questo tutorial, viene creata una funzione Lambda per utilizzare gli eventi da un flusso di dati Amazon Kinesis.

1. L'app personalizzata scrive record nel flusso.
2. AWS Lambda interroga lo stream e, quando rileva nuovi record nello stream, richiama la funzione Lambda.
3. AWS Lambda esegue la funzione Lambda assumendo il ruolo di esecuzione specificato al momento della creazione della funzione Lambda.

Prerequisiti

Installa AWS Command Line Interface

Se non l'hai ancora installato AWS Command Line Interface, segui i passaggi indicati in Installazione o aggiornamento della versione più recente di AWS CLI per installarlo.

Per eseguire i comandi nel tutorial, sono necessari un terminale a riga di comando o una shell (interprete di comandi). In Linux e macOS, utilizza la shell (interprete di comandi) e il gestore pacchetti preferiti.

> **Note**
>
> Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere una versione integrata su Windows di Ubuntu e Bash, installa il sottosistema Windows per Linux.

Creazione del ruolo di esecuzione

Crea il ruolo di esecuzione che autorizza la funzione ad accedere alle AWS risorse.

Per creare un ruolo di esecuzione

1. Apri la pagina Ruoli nella console IAM.
2. Scegliere Crea ruolo.
3. Creare un ruolo con le seguenti proprietà.
   - Trusted entity (Entità attendibile) – AWS Lambda.
   - Autorizzazioni — AWSLambdaKinesisExecutionRole.
   - Nome ruolo – **`lambda-kinesis-role`**.

La AWSLambdaKinesisExecutionRolepolicy dispone delle autorizzazioni necessarie alla funzione per leggere gli elementi da Kinesis e scrivere i log in CloudWatch Logs.

Creazione della funzione

Crea una funzione Lambda che elabora i messaggi Kinesis. Il codice funzione registra l'ID dell'evento e i dati dell'evento del record CloudWatch Kinesis in Logs.

Questo tutorial utilizza il runtime Node.js 24, ma abbiamo anche fornito codice di esempio in altri linguaggi di runtime. Per visualizzare il codice per il runtime che ti interessa, seleziona la scheda corrispondente nella casella seguente. Il JavaScript codice che utilizzerai in questo passaggio si trova nel primo esempio mostrato nella JavaScriptscheda.

.NET

SDK per .NET

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento Kinesis con Lambda utilizzando. 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;
}
```

Consumo di un evento Kinesis con Lambda utilizzando. 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite 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
}
```

Creazione della funzione

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir kinesis-tutorial
cd kinesis-tutorial
```

2. Copia il JavaScript codice di esempio in un nuovo file denominato`index.js`.
3. Crea un pacchetto di implementazione.

```
zip function.zip index.js
```

4. Creare una funzione Lambda con il comando `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
```

Test della funzione Lambda

Richiama la funzione Lambda manualmente utilizzando il comando `invoke` AWS Lambda CLI e un evento Kinesis di esempio.

Verifica della funzione Lambda

1. Copiare il codice JSON seguente in un file e salvarlo con nome `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. Utilizzare il comando `invoke` per inviare l'evento alla funzione.

```
aws lambda invoke --function-name ProcessKinesisRecords \
```

```
--cli-binary-format raw-in-base64-out \
--payload file://input.txt outputfile.txt
```

L'cli-binary-formatopzione è obbligatoria se utilizzi la versione 2. AWS CLI Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

La risposta viene salvata in `out.txt`.

Creare un flusso Kinesis

Utilizzare il comando `create-stream` per creare un flusso.

```
aws kinesis create-stream --stream-name lambda-stream --shard-count 1
```

Eseguire il seguente comando `describe-stream` per ottenere il flusso ARN.

```
aws kinesis describe-stream --stream-name lambda-stream
```

Verrà visualizzato l'output seguente:

```
{
    "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
    }
}
```

Nella fase successiva utilizzare l'ARN del flusso per associare il flusso alla funzione Lambda.

Aggiungi un'origine eventi in AWS Lambda

Eseguire il seguente comando AWS CLI `add-event-source`.

```
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
```

Annotare l'ID della mappatura per utilizzarlo in futuro. Mediante l'esecuzione del comando `list-event-source-mappings` è possibile ottenere un elenco di mappature delle origini eventi.

```
aws lambda list-event-source-mappings --function-name ProcessKinesisRecords \
--event-source arn:aws:kinesis:us-east-1:111122223333:stream/lambda-stream
```

Nella risposta è possibile verificare che il valore dello stato sia `enabled` (attivato). Le mappature dell'origine eventi possono essere disabilitate per sospendere il polling temporaneamente senza perdere alcuni record.

Eseguire il test della configurazione

Per testare la mappatura dell'origine eventi, aggiungere i record dell'evento al flusso Kinesis. Il valore `--data` è una stringa che il CLI codifica in base64 prima di inviarla a Kinesis. È possibile eseguire lo stesso comando più volte per aggiungere più record al flusso.

```
aws kinesis put-record --stream-name lambda-stream --partition-key 1 \
--data "Hello, this is a test."
```

Lambda usa il ruolo di esecuzione per leggere i record dal flusso. Quindi richiama la funzione Lambda, passando ai batch dei record. La funzione decodifica i dati di ogni record e li registra, inviando l'output a CloudWatch Logs. Visualizza i log nella console CloudWatch .

Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili a tuo carico. Account AWS

Come eliminare il ruolo di esecuzione

1. Aprire la pagina Ruoli della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

Per eliminare la funzione Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Digita **`confirm`** nel campo di immissione testo e scegli Delete (Elimina).

Per eliminare il flusso Kinesis

1. Accedi Console di gestione AWS e apri la console Kinesis all'indirizzo https://console.aws.amazon.com /kinesis.
2. Selezionare il flusso creato.
3. Scegli Operazioni > Elimina.
4. Inserisci **`delete`** nel campo di immissione del testo.
5. Scegli Delete (Elimina).

Utilizzo di Lambda con Kubernetes

Puoi implementare e gestire funzioni Lambda con l'API Kubernetes utilizzando i controller AWS per Kubernetes (ACK) o Crossplane.

Controller AWS per Kubernetes (ACK)

Puoi utilizzare ACK per implementare e gestire risorse AWS dall'API Kubernetes. Tramite ACK, AWS fornisce controller personalizzati open source per servizi AWS, ad esempio Lambda, Amazon Elastic Container Registry (Amazon ECR), Amazon Simple Storage Service (Amazon S3) e Amazon SageMaker AI. Ogni servizio AWS supportato ha un proprio controller personalizzato. Nel cluster Kubernetes installa un controller per ogni servizio AWS da utilizzare. Crea, quindi, una definizione risorsa personalizzata (CRD) per definire le risorse AWS.

È preferibile utilizzare Helm 3.8 o versioni successive per installare i controller ACK. Ogni controller ACK è dotato di un proprio grafico Helm, che installa il controller, le CRD e le regole RBAC di Kubernetes. Per ulteriori informazioni, consulta Installare un ACK Controller nella documentazione di ACK.

Dopo aver creato la risorsa personalizzata ACK, puoi utilizzarla come qualunque altro oggetto Kubernetes integrato. Ad esempio, puoi implementare e gestire funzioni Lambda con le tue toolchain Kubernetes preferite, inclusa kubectl.

Di seguito sono riportati alcuni esempi di casi d'uso per il provisioning di funzioni Lambda tramite ACK:

- La tua organizzazione utilizza il controllo degli accessi basato sui ruoli (RBAC) e ruoli IAM per gli account di servizio per creare limiti delle autorizzazioni. Con ACK puoi riutilizzare questo modello di sicurezza per Lambda senza bisogno di creare nuovi utenti e policy.
- La tua organizzazione utilizza un processo DevOps per implementare le risorse in un cluster Amazon Elastic Kubernetes Service (Amazon EKS) utilizzando manifesti Kubernetes. Con ACK puoi utilizzare un manifesto per eseguire il provisioning delle funzioni Lambda senza creare un'infrastruttura separata come modelli di codice.

Per ulteriori informazioni sull'utilizzo di ACK, consulta il tutorial Lambda nella documentazione di ACK.

Crossplane

Crossplane è un progetto CNCF (Cloud Native Computing Foundation) che utilizza Kubernetes per gestire risorse dell'infrastruttura cloud. Con Crossplane gli sviluppatori possono richiedere l'infrastruttura senza doverne comprendere le complessità. I team della piattaforma mantengono il controllo sulle modalità di provisioning e gestione dell'infrastruttura.

Utilizzando Crossplane puoi implementare e gestire funzioni Lambda con le tue toolchain Kubernetes preferite, ad esempio kubectl, e qualunque pipeline CI/CD in grado di implementare manifesti su Kubernetes. Di seguito sono riportati alcuni esempi di casi d'uso per il provisioning di funzioni Lambda tramite Crossplane:

- La tua organizzazione desidera attenersi alla conformità accertandosi che le funzioni Lambda abbiano tag corretti. I team della piattaforma possono utilizzare composizioni Crossplane per definire questa policy tramite astrazioni API. Gli sviluppatori, quindi, possono utilizzare tali astrazioni per implementare funzioni Lambda con tag.
- Il tuo progetto utilizza GitOps con Kubernetes. In questo modello, Kubernetes riconcilia continuamente il repository git (stato desiderato) con le risorse in esecuzione all'interno del cluster (stato corrente). Se esistono differenze, il processo GitOps apporta automaticamente modifiche al cluster. Puoi utilizzare GitOps con Kubernetes per implementare e gestire funzioni Lambda tramite Crossplane, adoperando strumenti e concetti familiari di Kubernetes, ad esempio CRD e controller.

Per ulteriori informazioni sull'utilizzo di Crossplane con Lambda, consulta:

- Schemi AWS per Crossplane: questo repository include esempi di utilizzo di Crossplane per l'implementazione di risorse AWS, incluse le funzioni Lambda.

  > **Note**
  >
  > Gli schemi AWS per Crossplane sono in fase di sviluppo attivo e non devono essere utilizzati in produzione.

- Implementazione di Lambda con Amazon EKS e Crossplane: questo video mostra un esempio avanzato di implementazione di un'architettura serverless AWS con Crossplane, esplorando la progettazione dal punto di vista dello sviluppatore e della piattaforma.

Utilizzo di Lambda con Amazon MQ

> **Note**
>
> Se desideri inviare dati a una destinazione diversa da una funzione Lambda o arricchire i dati prima di inviarli, consulta Amazon Pipes. EventBridge

Amazon MQ è un servizio gestito di broker dei messaggi per Apache ActiveMQ e RabbitMQ. Un broker di messaggi consente alle applicazioni e ai componenti software di comunicare utilizzando vari linguaggi di programmazione, sistemi operativi e protocolli di messaggistica formali tramite destinazioni eventi di tipo argomento o coda.

Amazon MQ può anche gestire istanze Amazon Elastic Compute Cloud (Amazon EC2) per tuo conto installando broker ActiveMQ o RabbitMQ e fornendo diverse topologie di rete e altre esigenze di infrastruttura.

Puoi utilizzare una funzione Lambda per elaborare i record da un broker di messaggi di Amazon MQ. Lambda richiama la funzione tramite uno strumento di mappatura dell'origine degli eventi, una risorsa Lambda che legge i messaggi dal broker e richiama la funzione in maniera sincrona.

> **Warning**
>
> Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center. AWS

Lo strumento di mappatura dell'origine degli eventi di Amazon MQ presenta le seguenti restrizioni di configurazione:

- Simultaneità: le funzioni Lambda che utilizzano uno strumento di mappatura dell'origine degli eventi Amazon MQ hanno un'impostazione di simultaneità massima predefinita. Per ActiveMQ, il servizio Lambda limita il numero di ambienti di esecuzione simultanei a cinque per lo strumento di mappatura dell'origine degli eventi di Amazon MQ. Per RabbitMQ, il numero di ambienti di esecuzione simultanei è limitato a 1 per lo strumento di mappatura dell'origine degli eventi di

Amazon MQ. Anche se modifichi le impostazioni di simultaneità sottoposta a provisioning o riservata, il servizio Lambda non renderà disponibili altri ambienti di esecuzione. Per richiedere un aumento della concorrenza massima predefinita per una singola mappatura della sorgente di eventi Amazon MQ, contatta Supporto l'UUID di mappatura della sorgente dell'evento e la regione. Poiché gli aumenti vengono applicati al livello dello strumento di mappatura dell'origine degli eventi specifico, non a livello di account o regione, è necessario richiedere manualmente un aumento di scalabilità per ogni strumento di mappatura dell'origine degli eventi.

- Più account: Lambda non supporta l'elaborazione tra più account. Non puoi utilizzare Lambda per elaborare i record da un broker di messaggi di Amazon MQ incluso in un Account AWS diverso.
- Autenticazione: per ActiveMQ, è supportato solo SimpleAuthenticationPluginActiveMQ. Per RabbitMQ è supportato solo il meccanismo di autenticazione PLAIN. Gli utenti devono utilizzare per gestire le proprie Gestione dei segreti AWS credenziali. Per ulteriori informazioni sull'autenticazione di ActiveMQ, consulta Integrazione di broker ActiveMQ con LDAP nella Guida per gli sviluppatori di Amazon MQ.
- Quota di connessione: i broker hanno un numero massimo di connessioni consentite per protocollo a livello di collegamento. Questa quota si basa sul tipo di istanza del broker. Per ulteriori informazioni, consulta la sezione Broker di Quote in Amazon MQ nella Guida per gli sviluppatori di Amazon MQ.
- Connettività: puoi creare broker in un VPC (Virtual Private Cloud) pubblico o privato. Per uso privato VPCs, la funzione Lambda deve accedere al VPC per ricevere messaggi. Per ulteriori informazioni, consulta the section called “Configurare la sicurezza della rete” di seguito in questa sezione.
- Destinazioni eventi: sono supportate solo le destinazioni coda. Tuttavia, puoi utilizzare un argomento virtuale che si comporta internamente come un argomento mentre interagisce con Lambda come una coda. Per ulteriori informazioni, consulta Destinazioni virtuali sul sito web di Apache ActiveMQ e Host virtuali sul sito Web di RabbitMQ.
- Topologia di rete: per ActiveMQ è supportato un solo broker a istanza singola o in standby per ogni strumento di mappatura dell'origine degli eventi. Per RabbitMQ è supportata una sola implementazione di broker o cluster a istanza singola per ogni strumento di mappatura dell'origine degli eventi. I broker a istanza singola richiedono un endpoint di failover. Per ulteriori informazioni su queste modalità di implementazione del broker, consulta Architettura del broker MQ attiva e Architettura del broker MQ di Rabbit nella Guida per gli sviluppatori di Amazon MQ.
- Protocolli: i protocolli supportati dipendono dal tipo di integrazione di Amazon MQ.
  - Per le integrazioni ActiveMQ, Lambda utilizza i messaggi utilizzando OpenWire/Java il protocollo Message Service (JMS). Non sono supportati altri protocolli per l'utilizzo dei messaggi. All'interno

del protocollo JMS, sono supportati solo `TextMessage` e `BytesMessage`. Lambda supporta anche le proprietà JMS personalizzate. Per ulteriori informazioni sul OpenWire protocollo, vedere OpenWireil sito Web di Apache ActiveMQ.

- Per le integrazioni RabbitMQ, Lambda utilizza i messaggi tramite il protocollo AMQP 0-9-1. Non sono supportati altri protocolli per l'utilizzo dei messaggi. Per ulteriori informazioni sull'implementazione del protocollo AMQP 0-9-1 in RabbitMQ, consulta la Guida di riferimento completa di AMQP 0-9-1 sul sito web di RabbitMQ.

Lambda supporta automaticamente le versioni più recenti di ActiveMQ e RabbitMQ supportate da Amazon MQ. Per le ultime versioni supportate, consulta le Note di rilascio di Amazon MQ nella Guida per gli sviluppatori di Amazon MQ.

> **Note**
>
> Per impostazione predefinita, Amazon MQ prevede un periodo di manutenzione settimanale per i broker. Durante quella finestra temporale, i broker non sono disponibili. Per i broker senza standby, Lambda non può elaborare alcun messaggio durante tale finestra.

Argomenti

- Informazioni sul gruppo di consumatori Lambda per Amazon MQ
- Configurazione dell'origine eventi Amazon MQ per Lambda
- Parametri dello strumento di mappatura dell'origine degli eventi
- Filtrare gli eventi da una origine eventi Amazon MQ
- Risolvere i problemi relativi gli errori di mappatura dell'origine eventi di Amazon MQ

Informazioni sul gruppo di consumatori Lambda per Amazon MQ

Per interagire con Amazon MQ, Lambda crea un gruppo di utenti che può leggere dai broker di Amazon MQ. Il gruppo di utenti viene creato con lo stesso ID dell'UUID dello strumento di mappatura dell'origine degli eventi.

Per le origini eventi di Amazon MQ, Lambda crea un batch dei record e li invia alla tua funzione in un singolo payload. Per controllare il comportamento, puoi configurare la finestra batch e le dimensioni del batch. Lambda estrae i messaggi finché elabora la dimensione del payload massima di 6 MB, la

finestra di batch scade o il numero di record raggiunge la dimensione completa del batch. Per ulteriori informazioni, consulta Comportamento di batching.

Il gruppo di utenti recupera i messaggi come BLOB di byte, li codifica in base64 in un singolo payload JSON e richiama la tua funzione. Se la funzione restituisce un errore per uno qualunque dei messaggi in un batch, Lambda ritenta l'intero batch di messaggi fino a quando l'elaborazione riesce o i messaggi scadono.

> **Note**
>
> Anche se le funzioni Lambda generalmente prevedono un timeout massimo di 15 minuti, gli strumenti di mappatura dell'origine degli eventi per Amazon MSK, Apache Kafka autogestito, Amazon DocumentDB e Amazon MQ per ActiveMQ e RabbitMQ supportano solo funzioni con timeout massimi di 14 minuti. Questa limitazione garantisce che lo strumento di mappatura dell'origine degli eventi possa gestire correttamente errori di funzioni e nuovi tentativi.

Puoi monitorare l'utilizzo simultaneo di una determinata funzione utilizzando la `ConcurrentExecutions` metrica in Amazon. CloudWatch Per ulteriori informazioni sulla simultaneità, consulta the section called “Configurazione della simultaneità riservata”.

Example Eventi record di 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**
>
> Nell'esempio di RabbitMQ, `pizzaQueue` è il nome della coda RabbitMQ e `/` è il nome dell'host virtuale. Quando si ricevono messaggi, l'origine eventi elenca i messaggi in `pizzaQueue::/`.

Configurazione dell'origine eventi Amazon MQ per Lambda

Argomenti

- Configurare la sicurezza della rete
- Creare lo strumento di mappatura dell'origine degli eventi

Configurare la sicurezza della rete

Per concedere a Lambda l'accesso completo ad Amazon MQ tramite lo strumento di mappatura dell'origine degli eventi, il broker deve utilizzare un endpoint pubblico (indirizzo IP pubblico) oppure deve fornire l'accesso all'Amazon VPC in cui hai creato il broker.

Quando usi Amazon MQ con Lambda, crea endpoint VPC AWS PrivateLink che forniscono alla funzione l'accesso alle risorse del tuo Amazon VPC.

> **Note**
>
> AWS PrivateLink Gli endpoint VPC sono necessari per le funzioni con mappature delle sorgenti degli eventi che utilizzano la modalità predefinita (su richiesta) per i poller degli eventi. Se la mappatura delle sorgenti degli eventi utilizza la modalità provisioning, non è necessario configurare gli endpoint VPC AWS PrivateLink .

Crea un endpoint per fornire l'accesso alle seguenti risorse:

- Lambda: crea un endpoint per il principale del servizio Lambda.
- AWS STS — Crea un endpoint per consentire AWS STS a un responsabile del servizio di assumere un ruolo per tuo conto.
- Secrets Manager: se il tuo broker utilizza Secrets Manager per archiviare le credenziali, crea un endpoint per Secrets Manager.

In alternativa, configura un gateway NAT su ogni sottorete pubblica in Amazon VPC. Per ulteriori informazioni, consulta the section called “Accesso Internet per funzioni del VPC”.

Quando crei una mappatura dell'origine degli eventi per Amazon MQ, Lambda verifica se Elastic Network Interfaces ENIs () sono già presenti per le sottoreti e i gruppi di sicurezza configurati per il tuo Amazon VPC. Se Lambda rileva che esistono ENIs, tenta di riutilizzarli. Altrimenti, Lambda ne crea di nuovi ENIs per connettersi all'origine dell'evento e richiamare la funzione.

> **Note**
>
> Le funzioni Lambda vengono sempre eseguite all'interno del servizio Lambda di VPCs proprietà. La configurazione VPC della funzione non influisce sullo strumento di mappatura dell'origine degli eventi. Solo la configurazione di rete dell'origine dell'evento determina il modo in cui Lambda si connette all'origine dell'evento.

Configura i gruppi di sicurezza per l'Amazon VPC contenente il tuo broker. Per impostazione predefinita, Amazon MQ utilizza le seguenti porte: `61617` (Amazon MQ per ActiveMQ) e `5671` (Amazon MQ per RabbitMQ).

- Regole in ingresso: consenti tutto il traffico sulla porta del broker predefinita per il gruppo di sicurezza associato all'origine eventi. In alternativa, puoi utilizzare una regola del gruppo di sicurezza autoreferenziante per consentire l'accesso da istanze all'interno dello stesso gruppo di sicurezza.
- Regole in uscita: consentono tutto il traffico sulla porta `443` per destinazioni esterne se la funzione deve comunicare con i servizi. AWS In alternativa, puoi anche utilizzare una regola del gruppo di sicurezza autoreferenziale per limitare l'accesso al broker se non hai bisogno di comunicare con altri servizi. AWS

- Regole di ingresso degli endpoint Amazon VPC: se utilizzi un endpoint Amazon VPC, il gruppo di sicurezza associato all'endpoint Amazon VPC deve consentire il traffico in entrata sulla porta 443 dal gruppo di sicurezza del broker.

Se il broker utilizza l'autenticazione, puoi anche limitare la policy degli endpoint per l'endpoint Secrets Manager. Per chiamare l'API Secrets Manager, Lambda utilizza il ruolo della funzione, non il principale del servizio Lambda.

Example Policy dell'endpoint VPC: endpoint 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"
          }
      ]
  }
```

Quando utilizzi gli endpoint Amazon VPC, AWS indirizza le chiamate API per richiamare la tua funzione utilizzando l'Elastic Network Interface (ENI) dell'endpoint. Il responsabile del servizio Lambda deve `lambda:InvokeFunction` richiamare tutti i ruoli e le funzioni che li utilizzano. ENIs

Per impostazione predefinita, gli endpoint Amazon VPC dispongono di policy IAM aperte che consentono un ampio accesso alle risorse. La best practice consiste nel limitare queste policy per eseguire le azioni necessarie utilizzando quell'endpoint. Per garantire che lo strumento di mappatura dell'origine degli eventi sia in grado di invocare la funzione Lambda, la policy degli endpoint VPC deve consentire al principale del servizio Lambda di chiamare `sts:AssumeRole` e `lambda:InvokeFunction`. Limitare le policy degli endpoint VPC per consentire solo le chiamate API provenienti dall'organizzazione impedisce il corretto funzionamento dello strumento di mappatura dell'origine degli eventi, pertanto `"Resource": "*"` è richiesto in queste policy.

Il seguente esempio di policy degli endpoint VPC mostra come concedere l'accesso richiesto al principale del servizio Lambda per gli endpoint AWS STS e Lambda.

Example Policy VPC Endpoint — endpoint AWS STS

```
{
      "Statement": [
          {
              "Action": "sts:AssumeRole",
              "Effect": "Allow",
              "Principal": {
                  "Service": [
                      "lambda.amazonaws.com"
                  ]
              },
              "Resource": "*"
          }
      ]
    }
```

Example Policy dell'endpoint VPC: endpoint Lambda

```
{
      "Statement": [
          {
              "Action": "lambda:InvokeFunction",
              "Effect": "Allow",
              "Principal": {
                  "Service": [
                      "lambda.amazonaws.com"
                  ]
              },
              "Resource": "*"
          }
      ]
  }
```

Creare lo strumento di mappatura dell'origine degli eventi

Creare una mappatura dell'origine eventi per indicare a Lambda di inviare i record da un broker Amazon MQ a una funzione Lambda. È possibile creare più mappature delle origini di eventi per

elaborare gli stessi dati con più funzioni o per elaborare elementi da più fonti con una singola funzione.

Per configurare la funzione per la lettura da Amazon MQ, aggiungi le autorizzazioni richieste e crea un trigger MQ nella console Lambda.

Per leggere i record da un broker Amazon MQ, la funzione Lambda richiede le seguenti autorizzazioni. Concedi a Lambda l'autorizzazione a interagire con il tuo broker Amazon MQ e le relative risorse sottostanti aggiungendo istruzioni di autorizzazione al tuo ruolo di esecuzione della funzione:

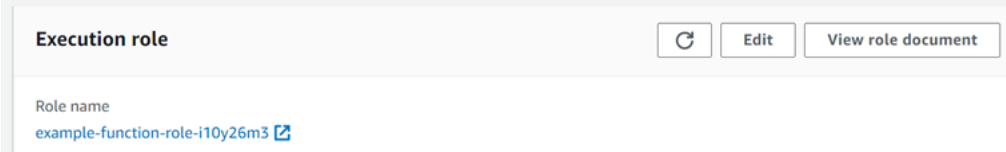
- mq: DescribeBroker
- gestore dei segreti: GetSecretValue
- ec2: CreateNetworkInterface
- ec2: DeleteNetworkInterface
- ec2: DescribeNetworkInterfaces
- ec2: DescribeSecurityGroups
- ec2: DescribeSubnets
- ec2: DescribeVpcs
- registri: CreateLogGroup
- registri: CreateLogStream
- registri: PutLogEvents

> **Note**
>
> Quando utilizzi una chiave gestita dal cliente crittografata, aggiungi anche l'autorizzazione `kms:Decrypt`.

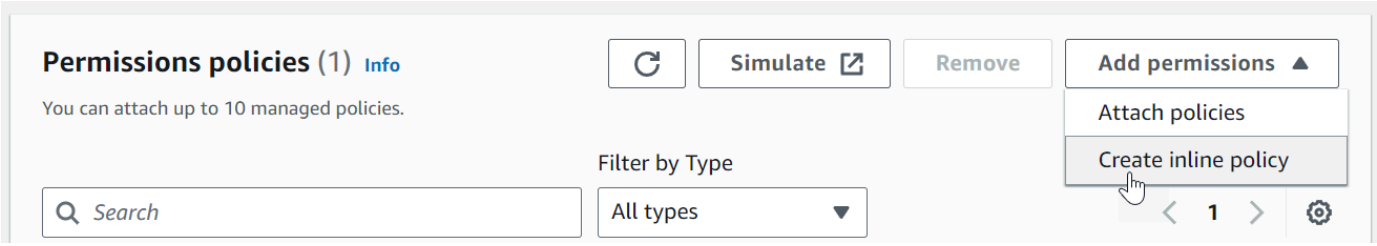
Per aggiungere le autorizzazioni e creare un trigger

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Quindi, seleziona la scheda Configuration (Configurazione) e poi Permissions (Autorizzazioni).

4. In Nome del ruolo, scegli il link al tuo ruolo di esecuzione. Questo ruolo si apre nella console IAM.



5. Scegli Aggiungi autorizzazioni, quindi seleziona Crea policy in linea.



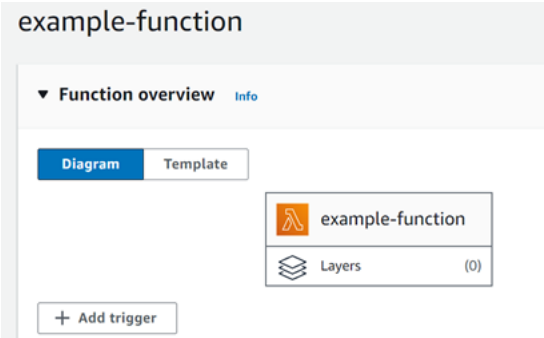
6. Nella sezione Editor di policy, scegli JSON. Immetti la seguente policy. La tua funzione necessita di queste autorizzazioni per leggere da un broker Amazon MQ.

   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**
>
> Quando utilizzi una chiave gestita dal cliente crittografata, è necessario aggiungere anche l'autorizzazione `kms:Decrypt`.

7. Scegli Next (Successivo). Inserire un nome per la policy, quindi seleziona Crea policy.
8. Torna alla funzione nella console Lambda. In Panoramica delle funzioni, scegliere Aggiungi trigger.



9. Seleziona il tipo di trigger MQ.
10. Configurare le opzioni richieste, quindi scegliere Add (Aggiungi).

Lambda supporta le seguenti opzioni per le origini eventi Amazon MQ:

- Broker MQ – Selezionare un broker Amazon MQ.
- Batch size (Dimensioni batch) – Impostare il numero massimo di messaggi da recuperare in un singolo batch.
- Queue name (Nome della coda) - Immettere la coda Amazon MQ da utilizzare.
- Source access configuration (Configurazione dell'accesso di origine) – Immettere le informazioni sull'host virtuale e il segreto di Secrets Manager in cui sono memorizzate le credenziali del broker.
- Abilita trigger – Disabilitare il trigger per interrompere l'elaborazione dei record.

Per attivare o disattivare il trigger (o eliminarlo), scegliere il trigger MQ nella finestra di progettazione. Per riconfigurare il trigger, utilizzare le operazioni API della mappatura dell'origine eventi.

Parametri dello strumento di mappatura dell'origine degli eventi

Tutti i tipi di origine eventi Lambda condividono le stesse operazioni API CreateEventSourceMapping e UpdateEventSourceMapping. Tuttavia, solo alcuni dei parametri si applicano ad Amazon MQ e RabbitMQ.

Parametro	Obbligatorio	Predefinito	Note
BatchSize	N	100	Massimo: 10.000
Abilitato	N	true	nessuno
FunctionName	Y	N/D	nessuno
FilterCriteria	N	N/D	Controllare gli eventi che Lambda invia alla funzione
MaximumBatchingWindowInSeconds	N	500 ms	Comportamento di batching
Queues	N	N/D	Il nome della coda di destinazione del broker Amazon MQ da utilizzare.
SourceAccessConfigurations	N	N/D	Per ActiveMQ, le credenziali BASIC_AUTH. Per RabbitMQ, può contenere sia le credenziali BASIC_AUTH che le informazioni VIRTUAL_HOST.

Filtrare gli eventi da una origine eventi Amazon MQ

Puoi utilizzare il filtraggio degli eventi per controllare quali record di un flusso o di una coda Lambda invia alla funzione. Per informazioni generali sul funzionamento del filtraggio eventi, consulta the section called “Filtro eventi”.

In questa sezione viene descritto il filtraggio degli eventi per le origini di eventi Amazon MQ.

> **Note**
>
> Le mappature delle sorgenti degli eventi di Amazon MQ supportano solo il filtraggio sulla chiave. `data`

Argomenti

- Nozioni di base sul filtraggio degli eventi Amazon MQ

Nozioni di base sul filtraggio degli eventi Amazon MQ

Supponiamo che la coda di messaggi di Amazon MQ contenga messaggi in formato JSON valido o come stringhe semplici. Un record di esempio sarebbe simile al seguente, con i dati convertiti in una stringa codificata Base64 nel campo `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=="
}
```

Per i broker sia Active MQ sia Rabbit MQ, puoi utilizzare il filtraggio degli eventi per filtrare i record utilizzando la chiave `data`. Supponiamo che la coda Amazon MQ contenga messaggi nel formato JSON seguente.

```
{
    "timeout": 0,
    "IPAddress": "203.0.113.254"
}
```

Per filtrare solo i record in cui il campo `timeout` è maggiore di 0, l'oggetto `FilterCriteria` sarebbe il seguente.

```
{
    "Filters": [
        {
            "Pattern": "{ \"data\" : { \"timeout\" : [ { \"numeric\": [ \">\",
 0] } } ] } }"
        }
    ]
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
    "data": {
        "timeout": [ { "numeric": [ ">", 0 ] } ]
        }
```

```
}
```

Puoi aggiungere il filtro utilizzando la console, la AWS CLI o un modello AWS SAM.

Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in Collegamento dei criteri di filtro a una mappatura dell'origine evento (console) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "data" : { "timeout" : [ { "numeric": [ ">", 0 ] } ] } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtraggio utilizzando l'AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
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 ] } ] } }"}]}'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :
 [ { \"numeric\": [ \">\", 0 ] } ] } }"}]}'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : { \"timeout\" :
 [ { \"numeric\": [ \">\", 0 ] } ] } }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML dell'origine degli eventi.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : { "timeout" : [ { "numeric": [ ">", 0 ] } ] } }'
```

Con Amazon MQ, puoi anche filtrare i record in cui il messaggio è una stringa semplice. Supponiamo di voler elaborare solo i record in cui il messaggio inizia con "Risultato: ". L'oggetto `FilterCriteria` dovrebbe avere la struttura seguente.

```
{
    "Filters": [
        {
            "Pattern": "{ \"data\" : [ { \"prefix\": \"Result: \" } ] }"
        }
    ]
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
    "data": [
        {
        "prefix": "Result: "
        }
    ]
}
```

Puoi aggiungere il filtro utilizzando la console, la AWS CLI o un modello AWS SAM.

Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in Collegamento dei criteri di filtro a una mappatura dell'origine evento (console) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "data" : [ { "prefix": "Result: " } ] }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtraggio utilizzando l'AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
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: \" } ] }"}]}'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"data\" : [ { \"prefix\":
 \"Result: \" } ] }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML dell'origine degli eventi.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "data" : [ { "prefix": "Result " } ] }'
```

I messaggi Amazon MQ devono essere stringhe codificate in UTF-8, semplici o in formato JSON. Questo perché Lambda decodifica gli array di byte Amazon MQ in UTF-8 prima di applicare i criteri di filtraggio. Se i messaggi utilizzano un'altra codifica, ad esempio UTF-16 o ASCII o se il formato del messaggio non corrisponde al formato `FilterCriteria`, Lambda elabora solo i filtri di metadati. La tabella seguente riepiloga il comportamento specifico:

Formato messaggio in arrivo	Formato del modello di filtro per le proprietà di messaggi	Operazione risultante
Stringa normale	Stringa normale	Filtri Lambda in base ai criteri di filtro.
Stringa normale	Nessun modello di filtro per le proprietà dei dati	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.
Stringa normale	JSON valido	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.
JSON valido	Stringa normale	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.
JSON valido	Nessun modello di filtro per le proprietà dei dati	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.
JSON valido	JSON valido	Filtri Lambda in base ai criteri di filtro.
Stringa codificata non UTF-8	JSON, stringa semplice o nessun modello	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.

Risolvere i problemi relativi gli errori di mappatura dell'origine eventi di Amazon MQ

Quando una funzione Lambda rileva un errore irreversibile, il consumatore Amazon MQ arresta l'elaborazione dei record. Tutti gli altri consumatori possono continuare a elaborare, a condizione che non riscontrino lo stesso errore. Per determinare la causa potenziale di un consumatore fermato, controllare il campo `StateTransitionReason` nei dettagli di reso del `EventSourceMapping` per uno dei seguenti codici:

`ESM_CONFIG_NOT_VALID`

La configurazione della mappa dell'origine eventi non è valida.

`EVENT_SOURCE_AUTHN_ERROR`

Lambda non è riuscito ad autenticare l'origine eventi.

`EVENT_SOURCE_AUTHZ_ERROR`

Lambda non dispone delle autorizzazioni necessarie per accedere all'origine eventi.

`FUNCTION_CONFIG_NOT_VALID`

La configurazione della funzione non è valida.

I record non verranno elaborati anche se Lambda li scarta a causa delle loro dimensioni. Il limite di dimensioni per i record Lambda è di 6 MB. Per riconsegnare i messaggi in caso di errore di funzione, è possibile utilizzare una coda di messaggi non instradabili (coda DLQ). Per ulteriori informazioni, consultare Message Redelivery and DLQ Handling sul sito Web Apache ActiveMQ e Guida all'affidabilità sul sito Web RabbitMQ.

> **Note**
>
> Lambda non supporta policy di riconsegna personalizzate. Lambda utilizza invece una policy con i valori predefiniti dalla pagina Policy di riconsegna sul sito Web Apache ActiveMQ, con `maximumRedeliveries` impostato su 6.

Utilizzo AWS Lambda con Amazon RDS

Puoi collegare una funzione Lambda a un database Amazon Relational Database Service (Amazon RDS) direttamente e attraverso un Server proxy per Amazon RDS. Le connessioni dirette sono utili in scenari semplici, mentre i server proxy sono consigliati per la produzione. Un proxy del database gestisce un pool di connessioni a database condivisi che consente alla funzione di raggiungere elevati livelli di simultaneità senza esaurire le connessioni al database.

Consigliamo di utilizzare il Server proxy per Amazon RDS per le funzioni Lambda che effettuano connessioni brevi e frequenti al database o aprono e chiudono numerose connessioni al database. Per ulteriori informazioni, consulta Connessione automatica di una funzione Lambda e un'istanza DB nella Guida per gli sviluppatori di Amazon Relational Database Service.

> **Tip**
>
> Per connettere rapidamente una funzione Lambda a un database Amazon RDS, puoi utilizzare la procedura guidata integrata nella console. Per aprire la procedura guidata, esegui queste operazioni:
>
> 1. Aprire la pagina Funzioni della console Lambda.
> 2. Seleziona la funzione a cui desideri connettere un database.
> 3. Nella scheda Configurazione, seleziona Database RDS.
> 4. Scegli Connessione al database RDS.
>
> Dopo aver collegato la funzione a un database, scegli Aggiungi proxy se desideri creare un proxy.

Configurazione della funzione per l'utilizzo con le risorse RDS

Nella console Lambda, puoi eseguire il provisioning, e configurare, delle istanze di database Amazon RDS e risorse proxy. Puoi farlo accedendo ai database RDS nella scheda Configurazione. In alternativa, puoi anche creare e configurare connessioni alle funzioni Lambda nella console Amazon RDS. Quando configuri un'istanza di database RDS da utilizzare con Lambda, tieni presente i seguenti criteri:

- Per effettuare la connessione a un database, la funzione si deve trovare nello stesso Amazon VPC in cui viene eseguito il database.
- Puoi utilizzare i database Amazon RDS con motori MySQL, MariaDB, PostgreSQL o Microsoft SQL Server.
- Puoi anche utilizzare cluster Aurora DB con motori MySQL o PostgreSQL.
- Devi fornire un segreto di Secrets Manager per l'autenticazione del database.
- Un ruolo IAM deve fornire l'autorizzazione all'uso del segreto, mentre una policy di attendibilità deve consentire ad Amazon RDS di assumere il ruolo.
- Il principale IAM che utilizza la console per configurare la risorsa Amazon RDS e connetterla alla tua funzione deve disporre delle seguenti autorizzazioni:

Policy di autorizzazione di esempio

> **Note**
>
> Le autorizzazioni del server proxy per Amazon RDS sono necessarie solo se configuri un server proxy per Amazon RDS per gestire un pool di connessioni al database.

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 addebita una tariffa oraria per i proxy in base alla dimensione dell'istanza di database, consulta Prezzi di Server proxy per RDS per conoscere i dettagli. Per ulteriori informazioni sulle connessioni proxy in generale, consulta Utilizzo di Server proxy per Amazon RDS nella Guida per l'utente di Amazon RDS.

Requisiti SSL/TLS per le connessioni Amazon RDS

Per stabilire SSL/TLS connessioni sicure a un'istanza di database Amazon RDS, la funzione Lambda deve verificare l'identità del server di database utilizzando un certificato affidabile. Lambda gestisce questi certificati in modo diverso a seconda del tipo di pacchetto di implementazione:

- Archivi di file.zip: la gestione dei certificati varia in base al runtime:
  - Node.js 18 e versioni precedenti: Lambda include automaticamente certificati CA e certificati RDS.
  - Node.js 20 e versioni successive: Lambda non carica più certificati CA aggiuntivi per impostazione predefinita. Impostare la variabile di ambiente `NODE_EXTRA_CA_CERTS` su `/var/runtime/ca-cert.pem`.

  Potrebbero essere necessarie fino a 4 settimane prima che i nuovi Regioni AWS certificati Amazon RDS vengano aggiunti ai runtime gestiti Lambda.
- Immagini dei container: le immagini AWS di base includono solo certificati CA. Se la tua funzione si connette a un'istanza del database Amazon RDS, devi includere i certificati appropriati nell'immagine container. Nel tuo Dockerfile, scarica il pacchetto di certificati corrispondente al Regione AWS luogo in cui ospiti il database. Esempio:

```
RUN curl https://truststore.pki.rds.amazonaws.com/us-east-1/us-east-1-bundle.pem -o /us-east-1-bundle.pem
```

Questo comando scarica il bundle di certificati Amazon RDS e lo salva nel percorso assoluto `/us-east-1-bundle.pem` nella directory principale del container. Quando configuri la connessione del database nel codice della funzione, devi fare riferimento a questo percorso esatto. Esempio:

Node.js

La funzione `readFileSync` è necessaria perché i client del database Node.js necessitano del contenuto effettivo del certificato in memoria, non solo del percorso al file del certificato. Senza `readFileSync`, il client interpreta la stringa del percorso come contenuto del certificato, generando un errore di «certificato autofirmato nella catena di certificati».

Example Configurazione della connessione Node.js per la funzione 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 Configurazione della connessione Python per la funzione 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

Per le funzioni Java che utilizzano connessioni JDBC, la stringa di connessione deve includere:

- `useSSL=true`
- `requireSSL=true`
- Un parametro `sslCA` che indica la posizione del certificato Amazon RDS nell'immagine container

Example Stringa di connessione Java per la funzione 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 Stringa di connessione.NET per la connessione MySQL nella funzione 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

Per le funzioni Go che utilizzano connessioni MySQL, carica il certificato Amazon RDS in un pool di certificati e registralo con il driver MySQL. La stringa di connessione deve quindi fare riferimento a questa configurazione utilizzando il parametro `tls`.

Example Codice Go per la connessione MySQL nella funzione 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 Configurazione della connessione Ruby per la funzione 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
)
```

Connessione a un database Amazon RDS in una funzione Lambda

Gli esempi di codice seguenti mostrano come implementare una funzione Lambda che si connette a un database Amazon RDS. La funzione effettua una semplice richiesta al database e restituisce il risultato.

> **Note**
>
> Questi esempi di codice sono validi solo per i pacchetti di implementazione .zip. Se stai implementando la funzione utilizzando un'immagine container, devi specificare il file del certificato Amazon RDS nel codice della funzione, come spiegato nella sezione precedente.

.NET

SDK per .NET

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda con .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda utilizzando. 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)
```

```
  }
};
```

Connessione a un database Amazon RDS in una funzione Lambda utilizzando. 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 per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda tramite 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}")
    }))
}
```

Elaborazione di notifiche di eventi da Amazon RDS

Puoi utilizzare Lambda per elaborare le notifiche degli eventi da un database Amazon RDS. Amazon RDS invia le notifiche a un argomento Amazon Simple Notification Service (Amazon SNS), che puoi configurare per richiamare una funzione Lambda. Amazon SNS avvolge il messaggio proveniente da Amazon RDS nel proprio documento evento e lo invia alla tua funzione.

Per ulteriori informazioni sulla configurazione di un database Amazon RDS per l'invio di notifiche, consulta Utilizzo delle notifiche di eventi di Amazon RDS.

Example Messaggio Amazon RDS in un evento 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"
            }
```

```
        }
      ]
    }
```

Completare il tutorial su Lambda e Amazon RDS

- Uso di una funzione Lambda per accedere a un database Amazon RDS: seguendo la Guida per l'utente di Amazon RDS, impara come utilizzare una funzione Lambda per scrivere dati su un database Amazon RDS attraverso un Server proxy per Amazon RDS. La funzione Lambda leggerà i record da una coda di Amazon SQS e scriverà nuove voci in una tabella del database ogni volta che verrà aggiunto un messaggio.

Seleziona un servizio di database per le tue applicazioni basate su Lambda

Molte applicazioni serverless devono archiviare e recuperare dati. AWS offre diverse opzioni di database che funzionano con le funzioni Lambda. Due delle scelte più popolari sono Amazon DynamoDB, un servizio di database NoSQL, e Amazon RDS, una soluzione di database relazionale tradizionale. Le sezioni seguenti spiegano le principali differenze tra questi servizi quando vengono utilizzati con Lambda e ti aiutano a selezionare il servizio di database giusto per la tua applicazione serverless.

Per saperne di più sugli altri servizi di database offerti da AWS e per comprenderne i casi d'uso e i compromessi in generale, consulta Scelta di un AWS servizio di database. Tutti i servizi di database AWS sono compatibili con Lambda, ma non tutti possono essere adatti al tuo caso d'uso specifico.

Quali sono le tue scelte quando selezioni un servizio di database con Lambda?

AWS offre diversi servizi di database. Per le applicazioni serverless, due delle scelte più popolari sono DynamoDB e Amazon RDS.

- DynamoDB è un servizio di database NoSQL completamente gestito ottimizzato per le applicazioni serverless. Fornisce una scalabilità ottimale e prestazioni costanti in millisecondi su qualsiasi scala.
- Amazon RDS è un servizio di database relazionale gestito che supporta più motori di database tra cui MySQL e PostgreSQL. Fornisce funzionalità SQL familiari con un'infrastruttura gestita.

Suggerimenti se conosci già le tue esigenze

Se hai già le idee chiare sulle tue esigenze, ecco i nostri suggerimenti di base:

Consigliamo DynamoDB per applicazioni serverless che richiedono prestazioni costanti a bassa latenza, scalabilità automatica e non richiedono join o transazioni complesse. È particolarmente adatto per applicazioni basate su Lambda grazie alla sua natura serverless.

Amazon RDS è la scelta migliore quando hai bisogno di query SQL complesse, join o se hai applicazioni esistenti che utilizzano database relazionali. Tuttavia, tieni presente che la connessione delle funzioni Lambda ad Amazon RDS richiede una configurazione aggiuntiva e può influire sui tempi di avvio a freddo.

Aspetti da tenere in considerazione selezioni un servizio di database

Quando scegli tra DynamoDB e Amazon RDS per le tue applicazioni Lambda, considera questi fattori:

- Gestione delle connessioni e avvii a freddo
- Modelli di accesso ai dati
- Complessità delle query
- Requisiti di coerenza dei dati
- Caratteristiche di scalabilità
- Modello di costi

Se conosci questi fattori, puoi selezionare l'opzione che soddisfa al meglio le esigenze del caso d'uso specifico.

Gestione delle connessioni e avvii a freddo

- DynamoDB utilizza un'API HTTP per tutte le operazioni. Le funzioni Lambda possono effettuare richieste immediate senza mantenere le connessioni, con conseguente miglioramento delle prestazioni di avvio a freddo. Ogni richiesta viene autenticata utilizzando AWS credenziali senza sovraccarico di connessione.
- Amazon RDS richiede la gestione dei pool di connessioni poiché utilizza connessioni di database tradizionali. Ciò può influire sugli avvii a freddo poiché le nuove istanze Lambda devono stabilire connessioni. Dovrai implementare strategie di pool di connessioni e potenzialmente utilizzare il Server proxy per Amazon RDS per gestire le connessioni in modo efficace. Tieni presente che l'utilizzo del Server proxy per Amazon RDS comporta costi aggiuntivi.

Modelli di accesso ai dati

- DynamoDB funziona al meglio con modelli di accesso noti e design a tabella singola. È ideale per applicazioni Lambda che richiedono un accesso coerente a bassa latenza ai dati basato su chiavi primarie o indici secondari.
- Amazon RDS offre flessibilità per query complesse e modelli di accesso in evoluzione. È più adatto quando le funzioni Lambda devono eseguire query uniche e personalizzate o join complessi su più tabelle.

Complessità delle query

- DynamoDB eccelle nelle operazioni semplici basate su chiavi e nei modelli di accesso predefiniti. Le query complesse devono essere progettate sulla base di strutture indicizzate e i join devono essere gestiti nel codice dell'applicazione.
- Amazon RDS supporta query SQL complesse con join, sottoquery e aggregazioni. Questo può semplificare il codice della funzione Lambda quando sono necessarie operazioni complesse sui dati.

Requisiti di coerenza dei dati

- DynamoDB offre opzioni di coerenza sia finali che avanzate, con una forte coerenza disponibile per le letture di singoli elementi. Le transazioni sono supportate ma con alcune limitazioni.
- Amazon RDS garantisce la conformità completa di atomicità, coerenza, isolamento e durabilità (ACID) e supporto per transazioni complesse. Se le tue funzioni Lambda richiedono transazioni complesse o una forte coerenza tra più record, Amazon RDS potrebbe essere più adatto.

Caratteristiche di scalabilità

- DynamoDB si adatta automaticamente al carico di lavoro. È in grado di gestire picchi improvvisi di traffico provenienti da funzioni Lambda senza pre-provisioning. Puoi utilizzare la modalità di capacità su richiesta per pagare solo per ciò che usi, in linea con il modello di scalabilità di Lambda.
- Amazon RDS ha una capacità fissa in base alla dimensione dell'istanza scelta. Se più funzioni Lambda tentano di connettersi contemporaneamente, potresti superare la quota di connessione. È necessario gestire con attenzione i pool di connessioni e potenzialmente implementare la logica dei tentativi.

Modello di costi

- I prezzi di DynamoDB si allineano bene con le applicazioni serverless. Con la capacità su richiesta, paghi solo per le letture e le scritture effettive eseguite dalle tue funzioni Lambda. Non sono previsti addebiti per i tempi di inattività.
- Amazon RDS addebita i costi per l'istanza in esecuzione indipendentemente dall'utilizzo. Questo può essere meno conveniente per i carichi di lavoro sporadici che possono essere tipici delle applicazioni serverless. Tuttavia, potrebbe essere più economico per carichi di lavoro ad alto throughput con un utilizzo costante.

Nozioni di base sulle istanze del database da te scelte

Ora che hai letto i criteri per scegliere tra DynamoDB and Amazon RDS e le principali differenze tra questi, puoi selezionare l'opzione più adatta alle tue esigenze e utilizzare le seguenti risorse per iniziare a utilizzarla.

DynamoDB

Iniziare a usare DynamoDB con le seguenti risorse

- Per un'introduzione al servizio DynamoDB, leggi Cos'è DynamoDB? nella Guida per gli sviluppatori di Amazon DynamoDB.
- Segui il tutorial Uso di Lambda con API Gateway per vedere un esempio di utilizzo di una funzione Lambda per eseguire operazioni CRUD su una tabella DynamoDB in risposta a una richiesta API.
- Leggi Programming with DynamoDB e AWS SDKs la Amazon DynamoDB Developer Guide per ulteriori informazioni su come accedere a DynamoDB dall'interno della funzione Lambda utilizzando una delle. AWS SDKs

Amazon RDS

Iniziare a usare Amazon RDS con le seguenti risorse

- Per un'introduzione al servizio Amazon RDS, leggi Cos'è Amazon Relational Database Service (Amazon RDS)? nella Guida per l'utente di Amazon Relational Database Service.
- Segui il tutorial Uso di una funzione Lambda per accedere a un database Amazon RDS nella Guida per l'utente di Amazon Relational Database Service.

- Per ulteriori informazioni sull'uso di Lambda con Amazon RDS, consulta the section called “RDS”.

Elaborare le notifiche di eventi Amazon S3 con Lambda

È possibile utilizzare Lambda per elaborare le notifiche degli eventi da Amazon Simple Storage Service. Amazon S3 può inviare un evento a una funzione Lambda quando un oggetto viene creato o eliminato. È possibile configurare le impostazioni di notifica su un bucket e concedere ad Amazon S3 l'autorizzazione a invocare una funzione sulla policy di autorizzazione basata sulle risorse della funzione.

> **Warning**
>
> Se la funzione Lambda utilizza lo stesso bucket che la attiva, potrebbe causare l'esecuzione della funzione in loop. Ad esempio, se il bucket attiva una funzione ogni volta che un oggetto viene caricato e la funzione carica un oggetto nel bucket, allora la funzione indirettamente lo attiva. Per evitare questo, utilizzare due bucket, oppure configurare il trigger in modo che venga applicato solo a un prefisso utilizzato per gli oggetti in entrata.

Amazon S3 richiama la funzione in modo asincrono con un evento che contiene dettagli sull'oggetto. L'esempio seguente mostra un evento che Amazon S3 ha inviato quando un pacchetto di distribuzione è stato caricato su Amazon S3.

Example Evento di notifica 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"
        }
      }
    }
  ]
}
```

Per richiamare la funzione, Amazon S3 necessita dell'autorizzazione dalla policy basata su risorse della funzione. Quando si configura un trigger Amazon S3 nella console Lambda, la console modifica la policy basata su risorse per consentire ad Amazon S3 di richiamare la funzione se il nome del bucket e l'ID account corrispondono. Se si configura la notifica in Amazon S3, si utilizza l'API Lambda per aggiornare la policy. È inoltre possibile utilizzare l'API Lambda per concedere l'autorizzazione a un altro account o limitare l'autorizzazione a un alias designato.

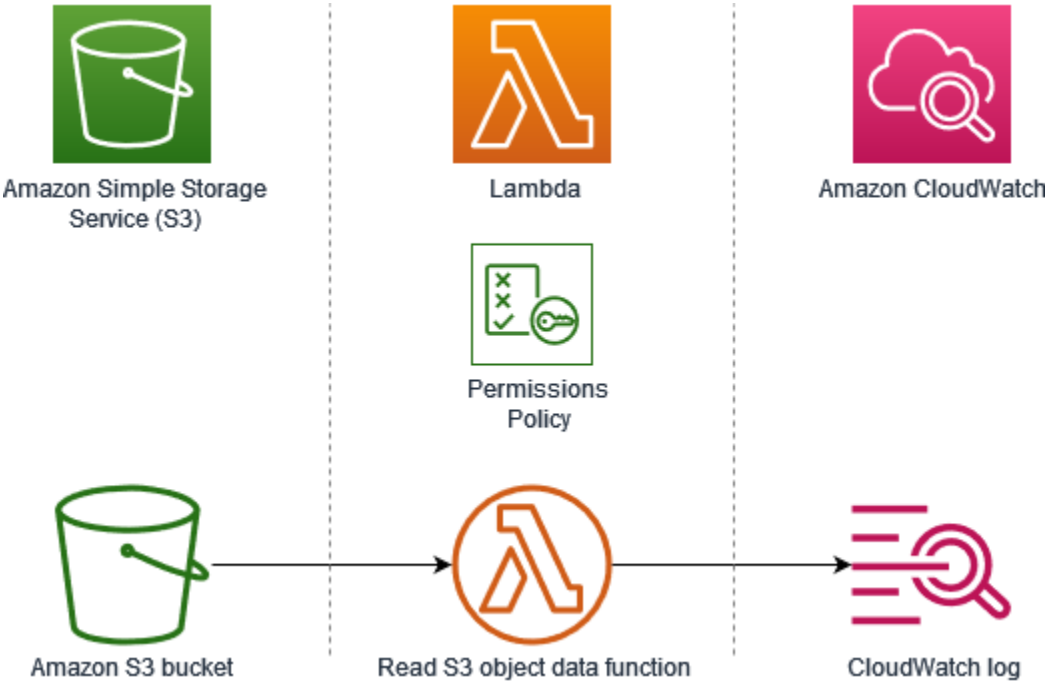
Se la tua funzione utilizza l' AWS SDK per gestire le risorse Amazon S3, necessita anche delle autorizzazioni Amazon S3 nel suo ruolo di esecuzione.

Argomenti

- Tutorial: uso di un trigger Amazon S3 per richiamare una funzione Lambda
- Tutorial: uso di un trigger Amazon S3 per creare immagini in miniatura

Tutorial: uso di un trigger Amazon S3 per richiamare una funzione Lambda

In questo tutorial si utilizzerà la console per creare una funzione Lambda e configurare un trigger per un bucket Amazon Simple Storage Service (Amazon S3). Ogni volta che aggiungi un oggetto al tuo bucket Amazon S3, la funzione viene eseguita e invia il tipo di oggetto in Amazon Logs. CloudWatch

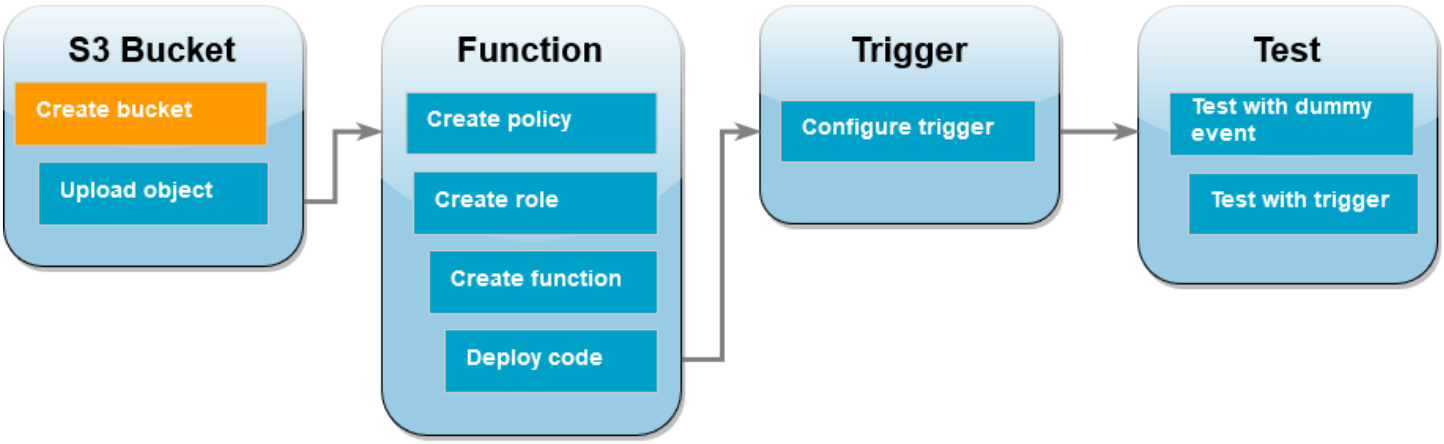


Questo tutorial dimostra come:

1. Crea un bucket Amazon S3.
2. Crea una funzione Lambda che restituisce il tipo di oggetto in un bucket Amazon S3.
3. Configura un trigger Lambda che richiami la tua funzione quando gli oggetti vengono caricati nel bucket.
4. Prova la tua funzione, prima con un evento fittizio e poi usando il trigger.

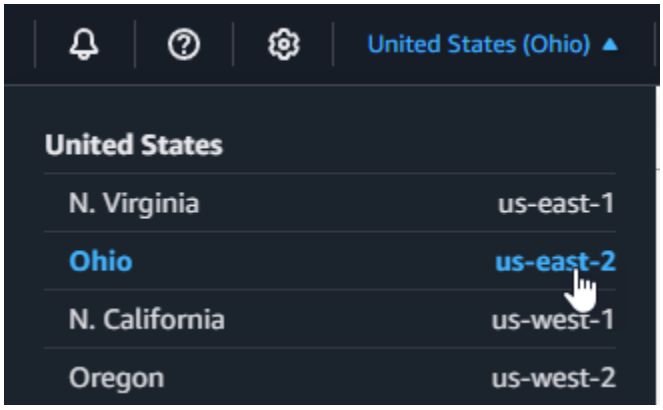
Completando questi passaggi, imparerai come configurare una funzione Lambda da eseguire ogni volta che vengono aggiunti o eliminati oggetti da un bucket Amazon S3. Puoi completare questo tutorial soltanto dalla Console di gestione AWS.

Crea un bucket Amazon S3



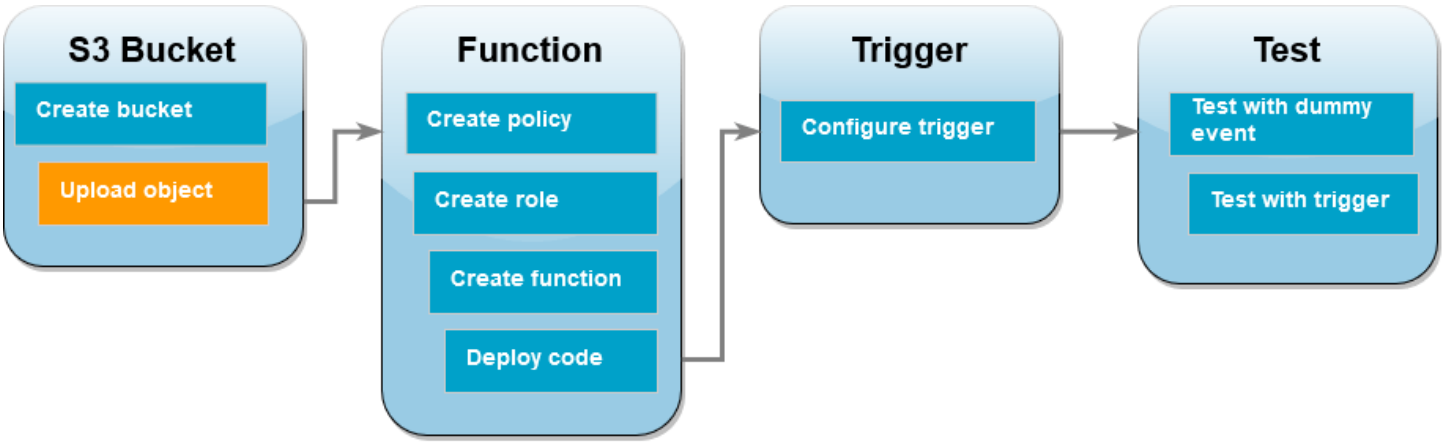
Come creare un bucket Amazon S3.

1. Apri la console Amazon S3 e seleziona la pagina General Purpose bucket.
2. Seleziona quello più Regione AWS vicino alla tua posizione geografica. Puoi modificare la regione utilizzando l'elenco a discesa nella parte superiore dello schermo. Più avanti nel tutorial, è necessario creare la funzione Lambda nella stessa regione.



3. Scegliere Create bucket (Crea bucket).
4. In General configuration (Configurazione generale), eseguire le operazioni seguenti:
   a. Per Tipo di secchio, assicurati che sia selezionata l'opzione Uso generale.
   b. Per Nome del bucket, inserisci un nome univoco globale che soddisfi le regole di denominazione dei bucket di Amazon S3. I nomi dei bucket possono contenere solo lettere minuscole, numeri, punti (.) e trattini (-).
5. Lascia tutte le altre opzioni impostate sui valori predefiniti e scegli Crea bucket.

Caricamento di un oggetto di test in un bucket

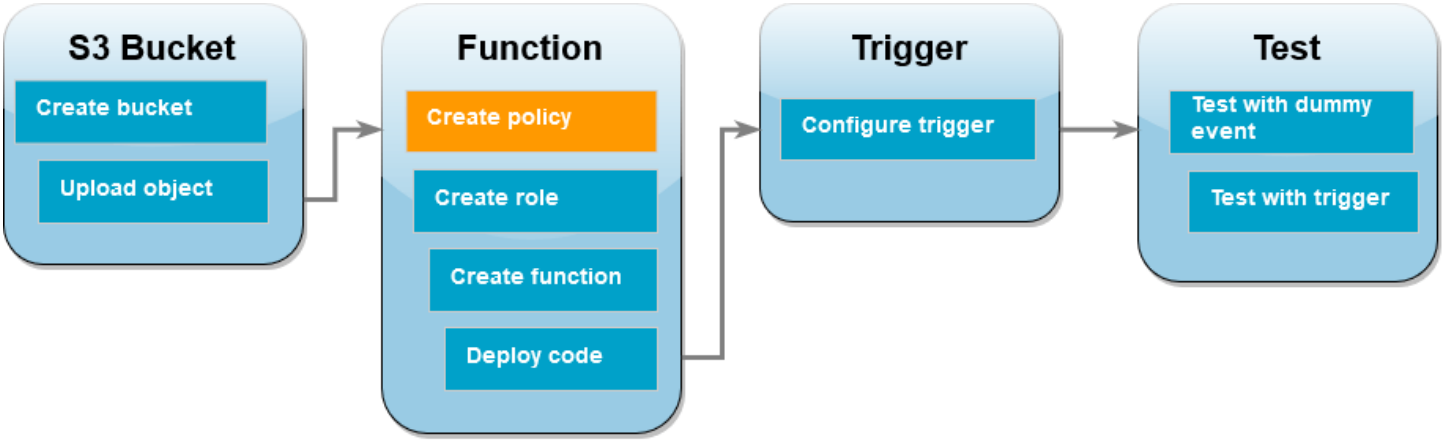


Caricamento di un oggetto di test

1. Apri la pagina Bucket della console Amazon S3 e scegli il bucket che hai creato durante il passaggio precedente.
2. Scegli Carica.
3. Scegli Aggiungi file e seleziona l'oggetto da caricare. È possibile selezionare qualsiasi file (ad esempio, `HappyFace.jpg`).
4. Seleziona Apri, quindi Carica.

Più avanti nel tutorial, testerai la funzione Lambda utilizzando questo oggetto.

Creazione di una policy di autorizzazione



Crea una politica di autorizzazioni che consenta a Lambda di ottenere oggetti da un bucket Amazon S3 e di scriverli su Amazon Logs. CloudWatch

Come creare la policy

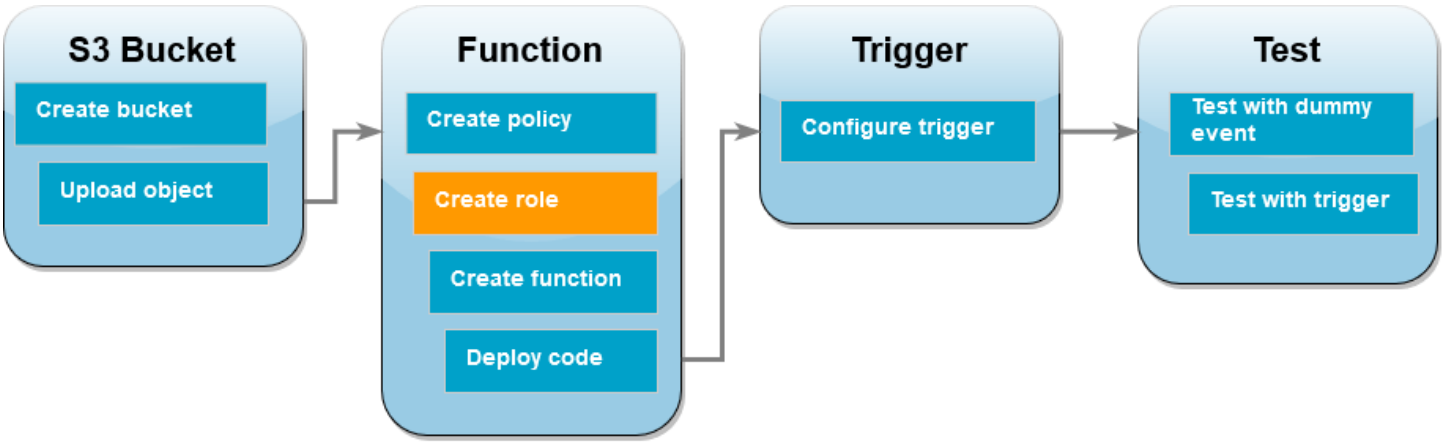
1. Apri la pagina Policies (Policy) nella console IAM.
2. Scegli Crea policy.
3. Scegliere la scheda JSON e quindi incollare la seguente policy personalizzata nell'editor JSON.

   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. Scegliere Next: Tags (Successivo: Tag).
5. Scegliere Next:Review (Successivo: Rivedi).
6. In Rivedi policy, per Nome della policy inserisci **`s3-trigger-tutorial`**.
7. Scegli Crea policy.

Creazione di un ruolo di esecuzione

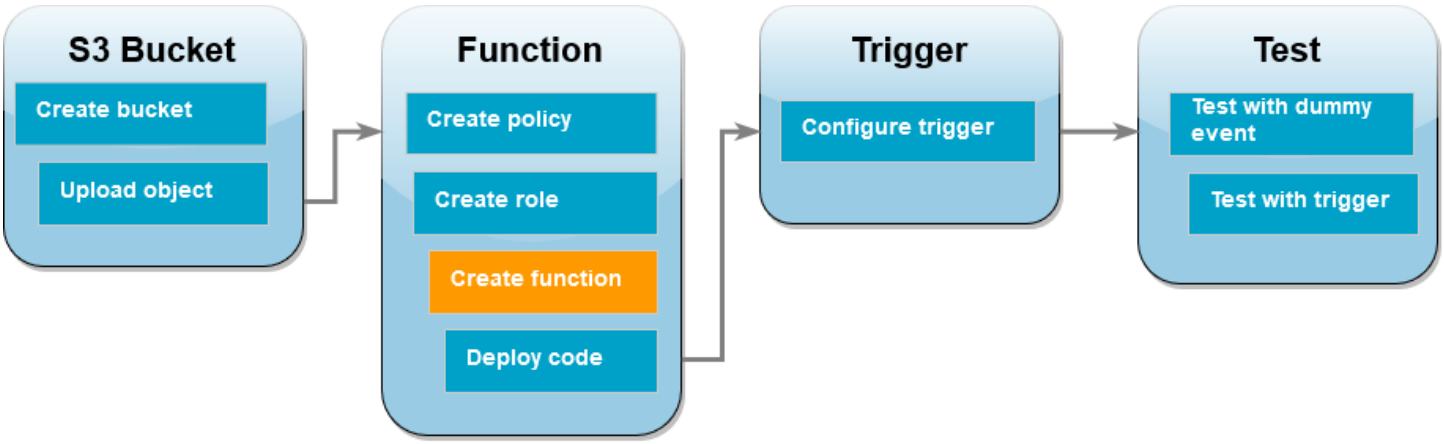


Un ruolo di esecuzione è un ruolo AWS Identity and Access Management (IAM) che concede a una funzione Lambda l'autorizzazione all' Servizi AWS accesso e alle risorse. In questa fase, creare un ruolo di esecuzione utilizzando la policy di autorizzazioni creata nel passaggio precedente.

Creazione di un ruolo di esecuzione e collegamento di una policy di autorizzazione personalizzata

1. Aprire la pagina Roles (Ruoli) della console IAM.
2. Scegli Crea ruolo.
3. Per il tipo di entità attendibile, scegli Servizio AWS , quindi per il caso d'uso seleziona Lambda.
4. Scegli Next (Successivo).
5. Nella casella di ricerca delle policy, immettere **`s3-trigger-tutorial`**.
6. Nei risultati della ricerca, seleziona la policy creata (`s3-trigger-tutorial`), quindi scegli Next (Successivo).
7. In Role details (Dettagli del ruolo), per Role name (Nome del ruolo), specifica **`lambda-s3-trigger-role`**, quindi scegli Create role (Crea ruolo).

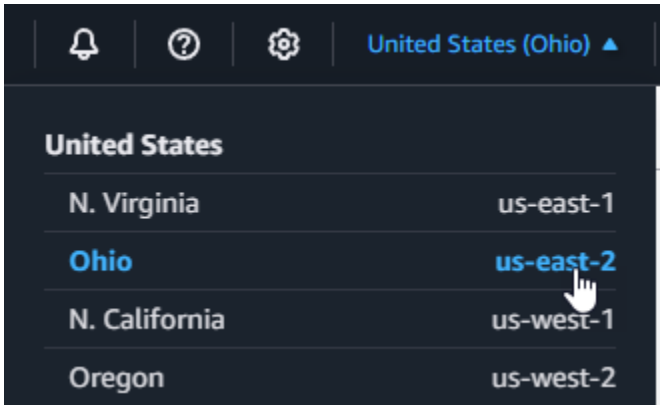
Creazione della funzione Lambda



Crea una funzione Lambda nella console usando il runtime Python 3.14.

Creazione della funzione Lambda

1. Aprire la pagina Funzioni della console Lambda.
2. Assicurati di lavorare nello stesso bucket in Regione AWS cui hai creato il tuo bucket Amazon S3. Puoi modificare la regione utilizzando l'elenco a discesa nella parte superiore dello schermo.



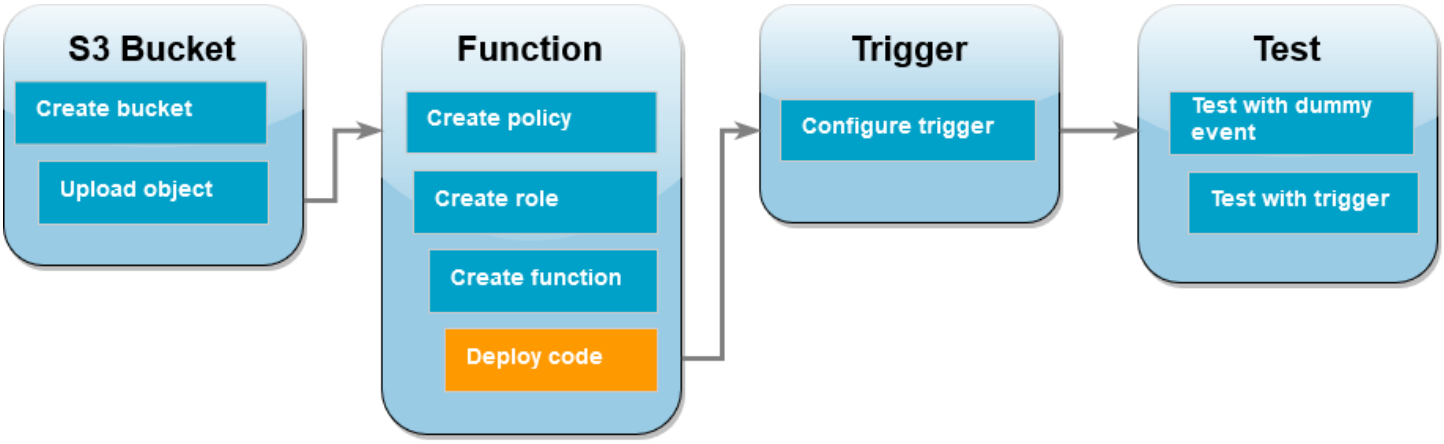
3. Scegli Crea funzione.
4. Scegli Crea da zero.
5. In Basic information (Informazioni di base) eseguire queste operazioni:
   a. Nel campo Nome funzione, inserisci `s3-trigger-tutorial`.
   b. Per Runtime, scegli Python 3.14.
   c. In Architecture (Architettura), scegli x86_64.
6. Nella scheda Modifica ruolo di esecuzione predefinito, effettua le seguenti operazioni:

a. Espandi la scheda, quindi scegli Utilizza un ruolo esistente.

b. Seleziona il `lambda-s3-trigger-role` che hai creato in precedenza.

7. Scegli Crea funzione.

Implementazione del codice della funzione



Questo tutorial utilizza il runtime di Python 3.14, ma abbiamo anche fornito file di codice di esempio per altri runtime. Per visualizzare il codice per il runtime che ti interessa, seleziona la scheda corrispondente nella casella seguente.

La funzione Lambda recupera il nome della chiave dell'oggetto caricato e il nome del bucket dal parametro `event` che riceve da Amazon S3. La funzione utilizza quindi il metodo get_object di AWS SDK per Python (Boto3) per recuperare i metadati dell'oggetto, incluso il tipo di contenuto (tipo MIME) dell'oggetto caricato.

Implementazione del codice della funzione

1. Scegli la scheda Python nella casella seguente e copia il codice.

   .NET

   SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento S3 con JavaScript Lambda utilizzando.

```
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);
    }
};
```

Consumo di un evento S3 con TypeScript Lambda utilizzando.

```
// 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite 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");

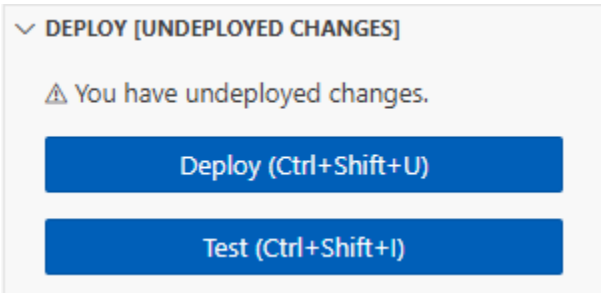
    tracing::info!("Request is for {} and object {}", bucket, key);

    let s3_get_object_result = s3_client
        .get_object()
        .bucket(bucket)
        .key(key)
        .send()
        .await;

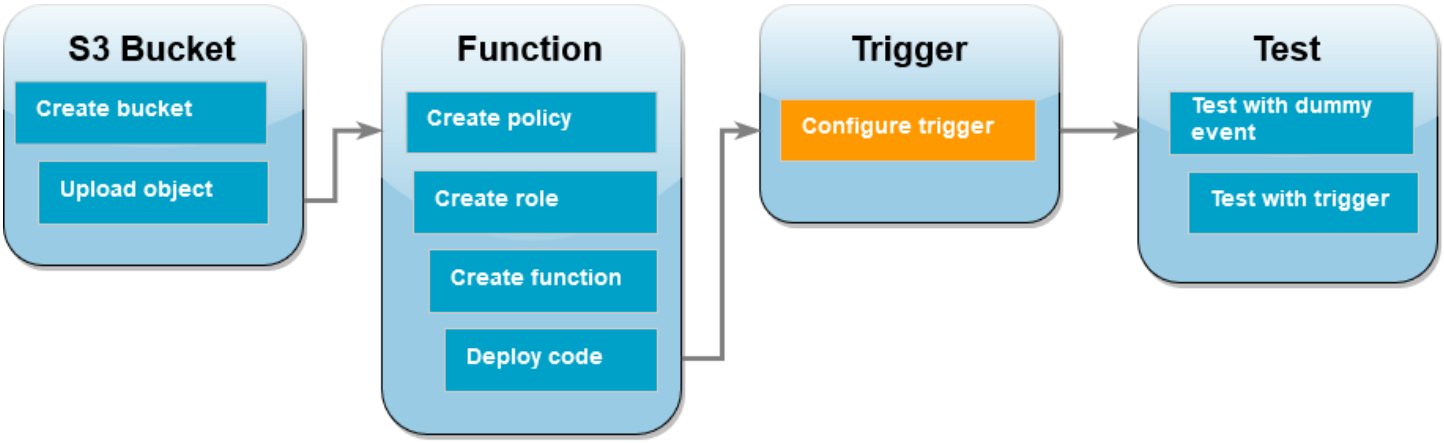
    match s3_get_object_result {
        Ok(_) => tracing::info!("S3 Get Object success, the s3GetObjectResult
 contains a 'body' property of type ByteStream"),
        Err(_) => tracing::info!("Failure with S3 Get Object request")
    }

    Ok(())
}
```

2. Nel riquadro Codice sorgente della console Lambda, incolla il codice nell'editor di codice, sostituendo il codice creato da Lambda.
3. Nella sezione DEPLOY, scegli Implementa per aggiornare il codice della tua funzione:

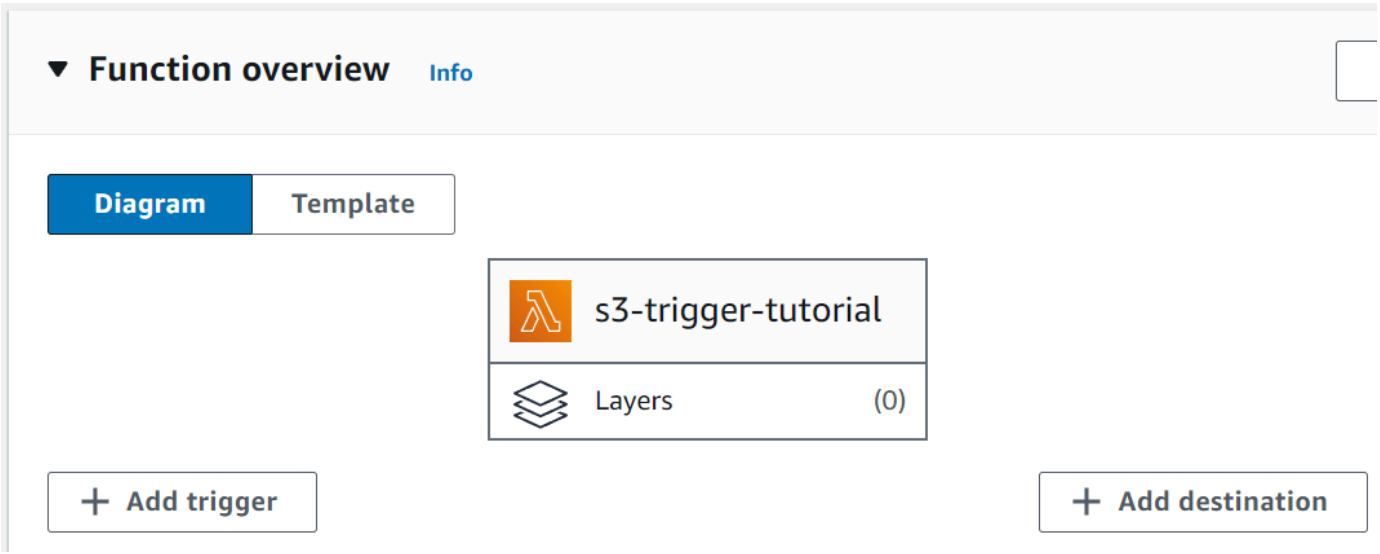


Creazione del trigger Amazon S3



Creazione del trigger Amazon S3

1. Nel riquadro Panoramica della funzione, scegli Aggiungi trigger.

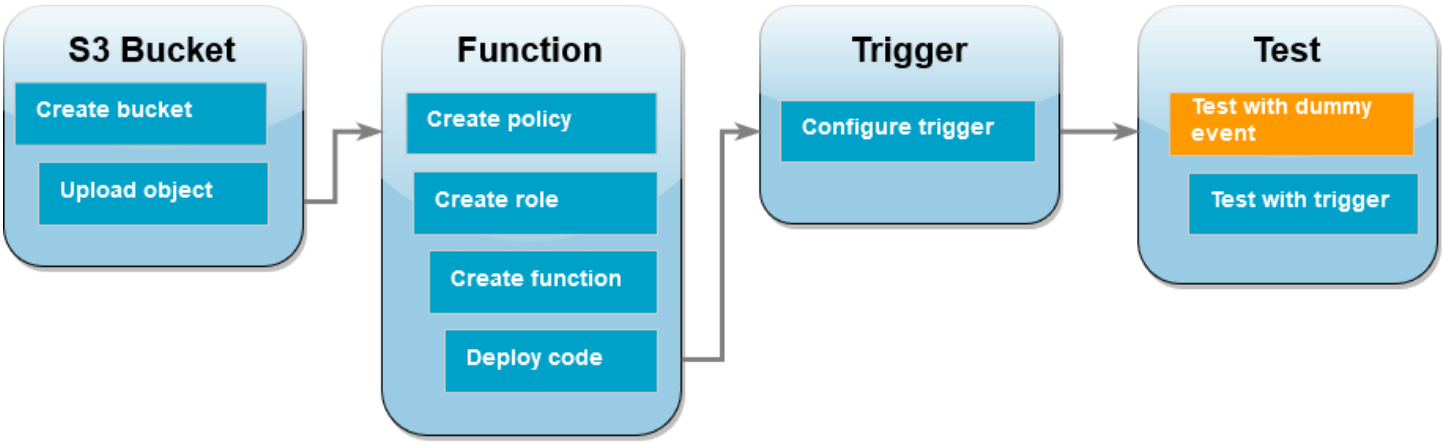


2. Seleziona S3.
3. In Bucket, seleziona il bucket che hai creato in precedenza nel tutorial.

4. In Tipi di eventi, verifica che sia selezionata l'opzione Tutti gli eventi di creazione di oggetti.
5. In Invocazione ricorsiva, seleziona la casella di controllo per confermare che non è consigliabile utilizzare lo stesso bucket Amazon S3 per input e output.
6. Scegliere Aggiungi.

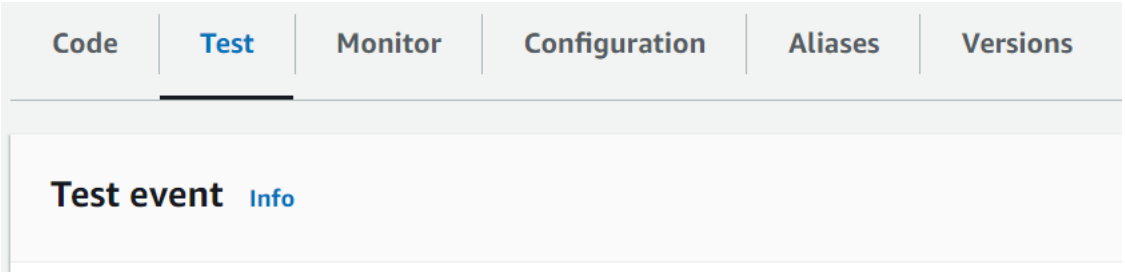
> **Note**
>
> Quando crei un trigger Amazon S3 per una funzione Lambda utilizzando la console Lambda, Amazon S3 configura una notifica degli eventi sul bucket specificato. Prima di configurare questa notifica di evento, Amazon S3 esegue una serie di controlli per confermare che la destinazione dell'evento esista e disponga delle policy IAM richieste. Amazon S3 esegue questi test anche su qualsiasi altra notifica di eventi configurata per quel bucket.
> A causa di questo controllo, se il bucket ha precedentemente configurato destinazioni di eventi per risorse che non esistono più o per risorse che non dispongono delle policy di autorizzazione richieste, Amazon S3 non sarà in grado di creare la nuova notifica di evento. Verrà visualizzato il seguente messaggio di errore che indica che non è stato possibile creare il trigger:
>
> ```
> An error occurred when creating the trigger: Unable to validate the following
>  destination configurations.
> ```
>
> Puoi visualizzare questo errore se in precedenza hai configurato un trigger per un'altra funzione Lambda utilizzando lo stesso bucket e da allora hai eliminato la funzione o modificato le sue policy di autorizzazioni.

Test di una funzione Lambda con un evento fittizio



Test della funzione Lambda con un evento fittizio

1. Nella pagina della console Lambda della funzione, seleziona la scheda Test.



2. Per Event name (Nome evento) immettere `MyTestEvent`.
3. Nella sezione JSON dell'evento, incolla il seguente evento di test. Assicurati di sostituire i seguenti valori:

   - Sostituisci `us-east-1` con la regione in cui è stato creato il bucket Amazon S3.
   - Sostituisci entrambe le istanze di `amzn-s3-demo-bucket` con il nome del bucket Amazon S3.
   - Sostituisci `test%2FKey` con il nome dell'oggetto di test che hai caricato in precedenza nel tuo bucket (ad esempio, `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. Scegli Save (Salva).
5. Scegli Test (Esegui test).
6. Se la tua funzione viene eseguita correttamente, vedrai un output simile al seguente nella scheda Risultati dell'esecuzione.

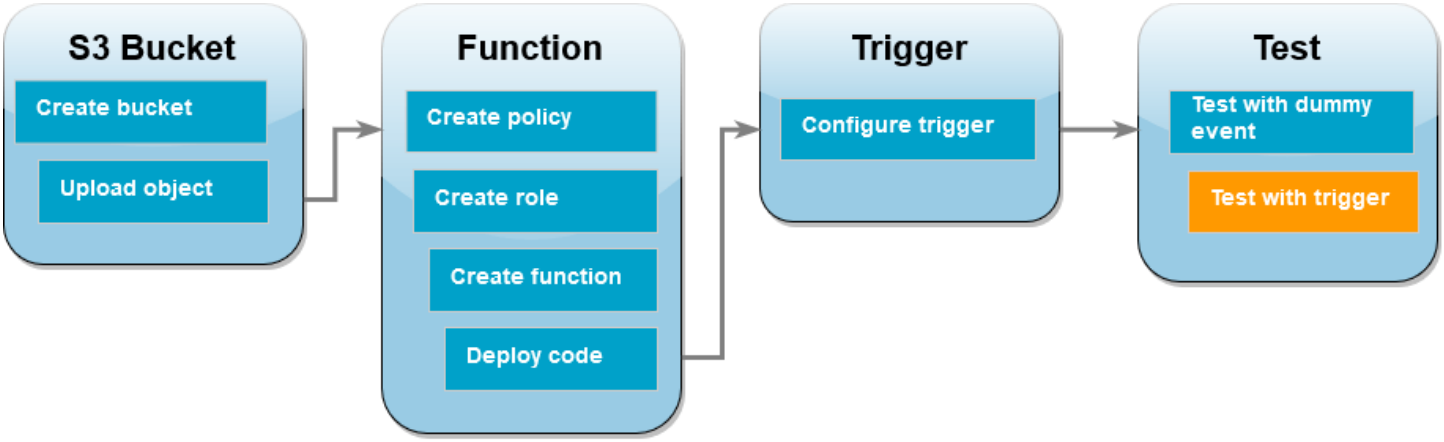
```
Response
"image/jpeg"

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
```

Test della funzione Lambda mediante il trigger Amazon S3



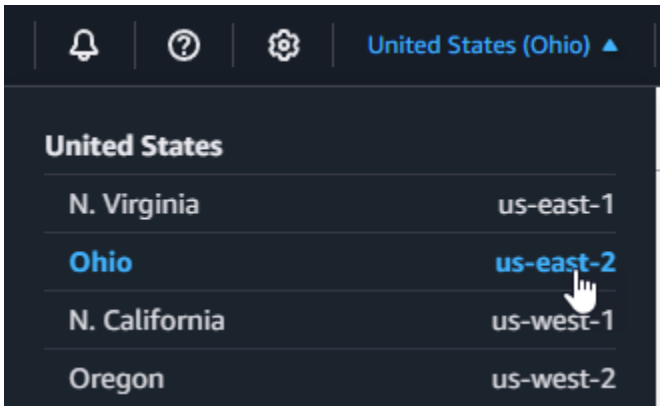
Per verificare la funzione con il trigger configurato, carica un oggetto sul bucket Amazon S3 utilizzando la console. Per verificare che la funzione Lambda sia stata eseguita come previsto, usa CloudWatch Logs per visualizzare l'output della funzione.

Caricamento di un oggetto nel bucket Amazon S3

1. Apri la pagina Bucket della console Amazon S3 e scegli il bucket che hai creato in precedenza.
2. Scegli Carica.
3. Scegli Aggiungi file e usa il selettore di file per scegliere un oggetto da caricare. Tale oggetto può essere qualsiasi file desideri.
4. Seleziona Apri, quindi Carica.

Per verificare l'invocazione della funzione utilizzando Logs CloudWatch

1. Aprire la console CloudWatch.
2. Assicurati di lavorare nella stessa modalità in Regione AWS cui hai creato la funzione Lambda. Puoi modificare la regione utilizzando l'elenco a discesa nella parte superiore dello schermo.



3. Scegli Log e quindi Gruppi di log.
4. Scegli il nome del gruppo di log per la funzione (`/aws/lambda/s3-trigger-tutorial`).
5. In Flussi di log, scegli il flusso di log più recente.
6. Se la tua funzione è stata richiamata correttamente in risposta al trigger Amazon S3, vedrai un output simile al seguente. Il `CONTENT TYPE` visualizzato dipende dal tipo di file che hai caricato nel bucket.

```
2022-05-09T23:17:28.702Z 0cae7f5a-b0af-4c73-8563-a3430333cc10 INFO CONTENT
 TYPE: image/jpeg
```

Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili a tuo carico. Account AWS

Per eliminare la funzione Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **`confirm`** nel campo di immissione del testo, quindi scegli Elimina.

Come eliminare il ruolo di esecuzione

1. Aprire la pagina Ruoli della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

Per eliminare il bucket S3

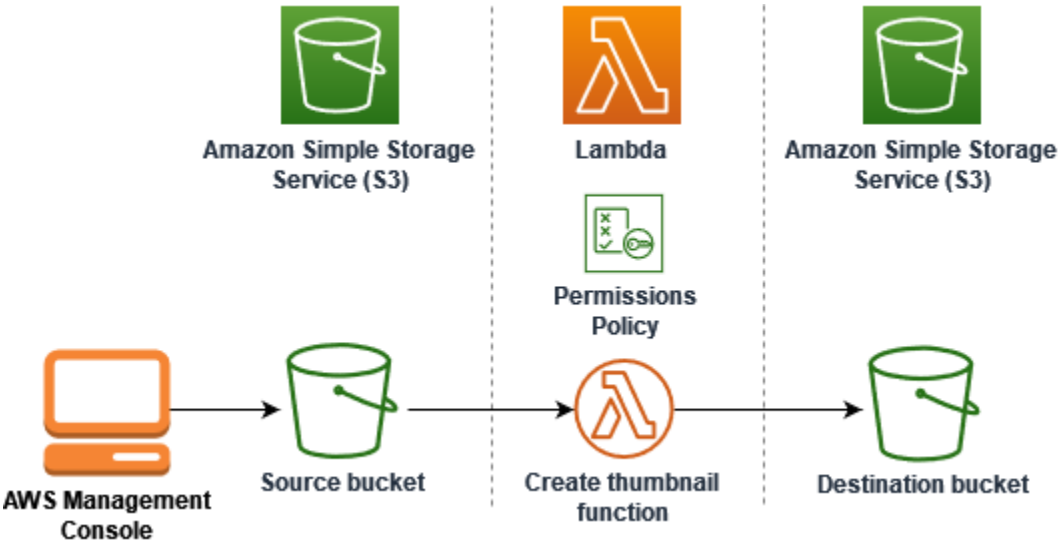
1. Aprire la console Amazon S3.
2. Selezionare il bucket creato in precedenza.
3. Scegli Elimina.
4. Inserisci il nome del bucket nel campo di immissione testo.
5. Scegli Delete Bucket (Elimina bucket).

Fasi successive

In Tutorial: uso di un trigger Amazon S3 per creare immagini in miniatura, il trigger Amazon S3 richiama una funzione per creare un'immagine in miniatura per ogni file immagine caricato nel bucket. Questo tutorial richiede un livello moderato di conoscenza del AWS dominio Lambda. Dimostra come creare risorse utilizzando AWS Command Line Interface (AWS CLI) e come creare un pacchetto di distribuzione di archivi di file.zip per la funzione e le sue dipendenze.

Tutorial: uso di un trigger Amazon S3 per creare immagini in miniatura

In questo tutorial, creerai e configurerai una funzione Lambda che ridimensiona le immagini aggiunte a un bucket Amazon Simple Storage Service (Amazon S3). Quando aggiungi un file di immagine al bucket, Amazon S3 richiama la tua funzione Lambda. La funzione crea quindi una versione in miniatura dell'immagine e la invia a un altro bucket Amazon S3.



Per completare questo tutorial, completa le seguenti attività:

1. Crea i bucket Amazon S3 di origine e destinazione e carica un'immagine di esempio.
2. Crea una funzione Lambda che ridimensiona un'immagine e restituisce una miniatura in un bucket Amazon S3.
3. Configura un trigger Lambda che richiama la tua funzione quando gli oggetti vengono caricati nel bucket di origine.
4. Testa la tua funzione, prima con un evento fittizio e poi caricando un'immagine nel tuo bucket di origine.

Completando questi passaggi, imparerai come utilizzare Lambda per eseguire un'attività di elaborazione di file su oggetti aggiunti a un bucket Amazon S3. È possibile completare questo tutorial utilizzando AWS Command Line Interface (AWS CLI) o. Console di gestione AWS

Se stai cercando un esempio più semplice per imparare a configurare un trigger Amazon S3 per Lambda, puoi provare Tutorial: utilizzo di un trigger Amazon S3 per richiamare una funzione Lambda.

Argomenti

- Prerequisiti
- Creazione di due bucket Amazon S3
- Caricamento di un'immagine di test nel bucket di origine
- Creazione di una policy di autorizzazione
- Creazione di un ruolo di esecuzione

- Creazione del pacchetto di implementazione della funzione
- Creazione della funzione Lambda
- Configurazione di Amazon S3 per richiamare la funzione
- Test di una funzione Lambda con un evento fittizio
- Test della funzione tramite il trigger Amazon S3
- Pulizia delle risorse

Prerequisiti

Se vuoi usare il per AWS CLI completare il tutorial, installa la versione più recente di AWS Command Line Interface.

Per il codice della funzione Lambda, puoi utilizzare Python o Node.js. Installa gli strumenti di supporto linguistico e un gestore di pacchetti per il linguaggio che desideri utilizzare.

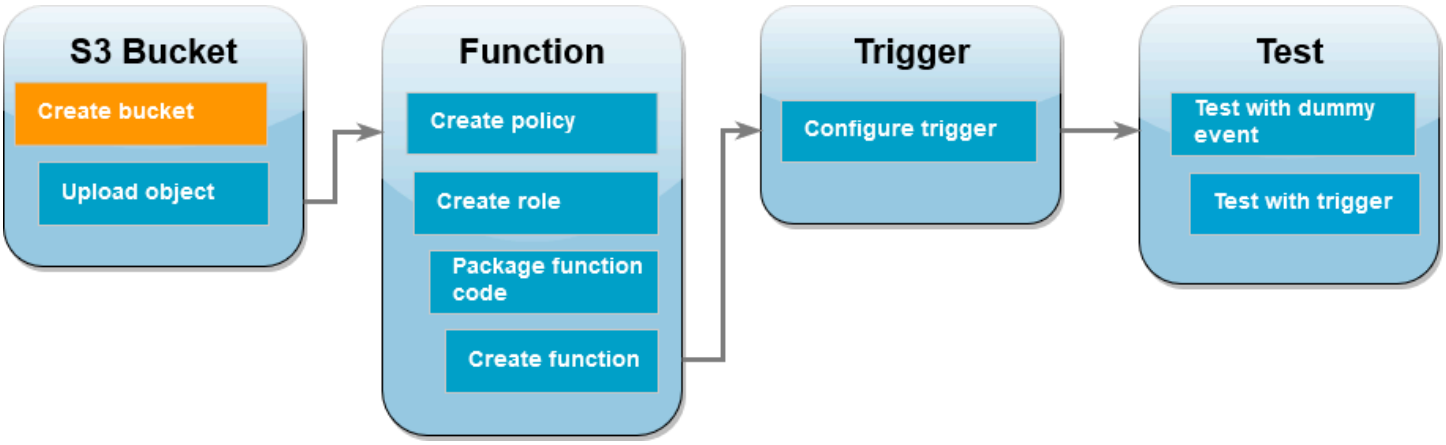
Installa il AWS Command Line Interface

Se non l'hai ancora installato AWS Command Line Interface, segui i passaggi indicati in Installazione o aggiornamento della versione più recente di AWS CLI per installarlo.

Per eseguire i comandi nel tutorial, sono necessari un terminale a riga di comando o una shell (interprete di comandi). In Linux e macOS, utilizza la shell (interprete di comandi) e il gestore pacchetti preferiti.

> **Note**
>
> Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere una versione integrata su Windows di Ubuntu e Bash, installa il sottosistema Windows per Linux.

Creazione di due bucket Amazon S3

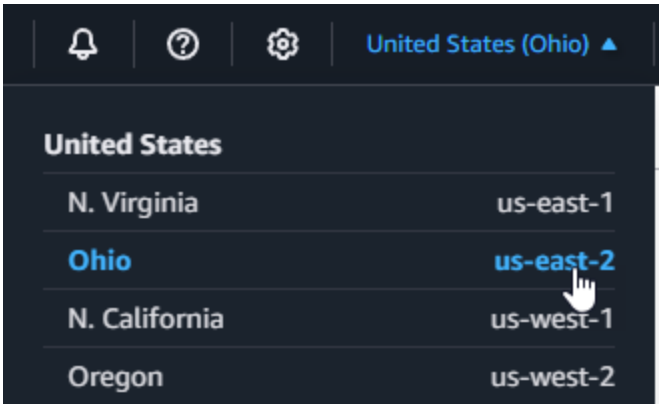


Per prima cosa, crea due bucket Amazon S3. Il primo bucket è il bucket di origine in cui caricherai le tue immagini. Il secondo bucket viene utilizzato da Lambda per salvare la miniatura ridimensionata quando richiami la tua funzione.

Console di gestione AWS

Creazione di bucket Amazon S3 (console)

1. Apri la console Amazon S3 e seleziona la pagina General Purpose bucket.
2. Seleziona quello più Regione AWS vicino alla tua posizione geografica. Puoi modificare la regione utilizzando l'elenco a discesa nella parte superiore dello schermo. Più avanti nel tutorial, è necessario creare la funzione Lambda nella stessa regione.



3. Scegliere Create bucket (Crea bucket).
4. In General configuration (Configurazione generale), eseguire le operazioni seguenti:

   a. Per Tipo di secchio, assicurati che sia selezionata l'opzione Uso generale.

b. Per Nome del bucket, inserisci un nome univoco globale che soddisfi le regole di denominazione dei bucket di Amazon S3. I nomi dei bucket possono contenere solo lettere minuscole, numeri, punti (.) e trattini (-).

5. Lascia tutte le altre opzioni impostate sui valori predefiniti e scegli Crea bucket.
6. Ripeti i passaggi da 1 a 5 per creare il bucket di destinazione. Per Nome del bucket, inserisci **amzn-s3-demo-source-bucket-resized**, dove **amzn-s3-demo-source-bucket** è il nome del bucket di origine che hai appena creato.

AWS CLI

Creazione dei bucket Amazon S3 (AWS CLI)

1. Esegui il comando della CLI sotto riportato per creare il bucket di origine. Il nome che scegli per il bucket deve essere univoco a livello globale e seguire le regole di denominazione dei bucket di Amazon S3. I nomi possono contenere solo lettere minuscole, numeri, punti (.) e trattini (-). Per `region` e `LocationConstraint`, scegli la Regione AWS più vicina alla tua posizione geografica.

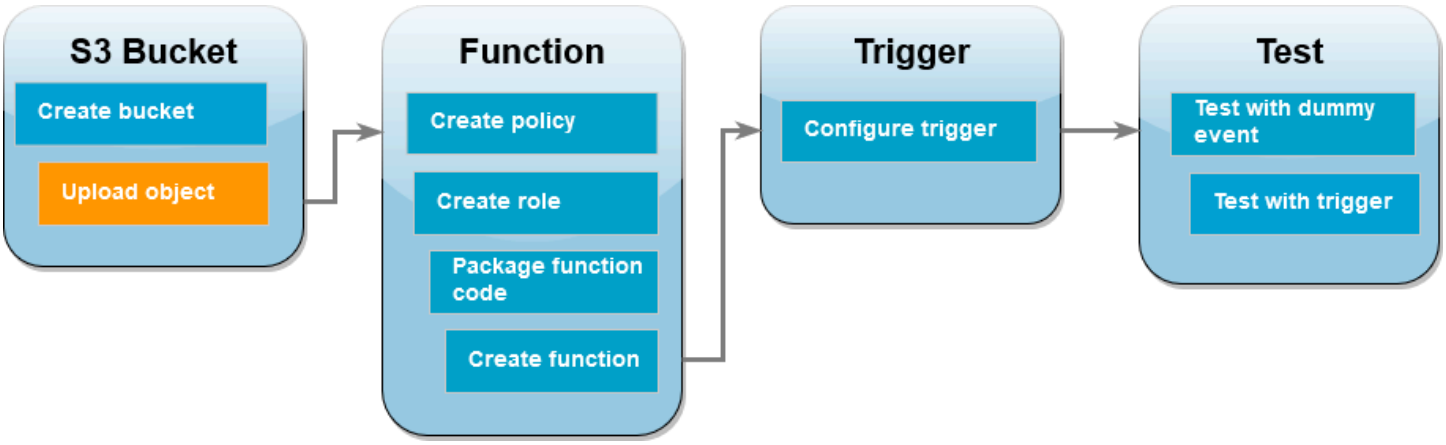
```
aws s3api create-bucket --bucket amzn-s3-demo-source-bucket --region us-east-1 \
--create-bucket-configuration LocationConstraint=us-east-1
```

Più avanti nel tutorial, devi creare la tua funzione Lambda nello Regione AWS stesso bucket di origine, quindi prendi nota della regione che hai scelto.

2. Esegui il comando sotto riportato per creare il bucket di destinazione. Per il nome del bucket, devi usare **amzn-s3-demo-source-bucket-resized**, dove **amzn-s3-demo-source-bucket** è il nome del bucket di origine che hai creato nel passaggio 1. Per `region` e`LocationConstraint`, scegli lo stesso Regione AWS che hai usato per creare il bucket sorgente.

```
aws s3api create-bucket --bucket amzn-s3-demo-source-bucket-resized --region us-
east-1 \
--create-bucket-configuration LocationConstraint=us-east-1
```

Caricamento di un'immagine di test nel bucket di origine



Più avanti nel tutorial, testerai la tua funzione Lambda invocandola utilizzando la console o AWS CLI la console Lambda. Per confermare che la funzione funzioni correttamente, il bucket di origine deve contenere un'immagine di test. Questa immagine può essere qualsiasi file JPG o PNG che scegli.

Console di gestione AWS

Caricamento di un'immagine di test nel bucket di origine (console)

1. Nella console Amazon S3, apri la pagina Bucket.
2. Seleziona il bucket di origine che hai creato nella fase precedente.
3. Scegli Carica.
4. Scegli Aggiungi file e usa il selettore di file per scegliere l'oggetto da caricare.
5. Seleziona Apri, quindi Carica.

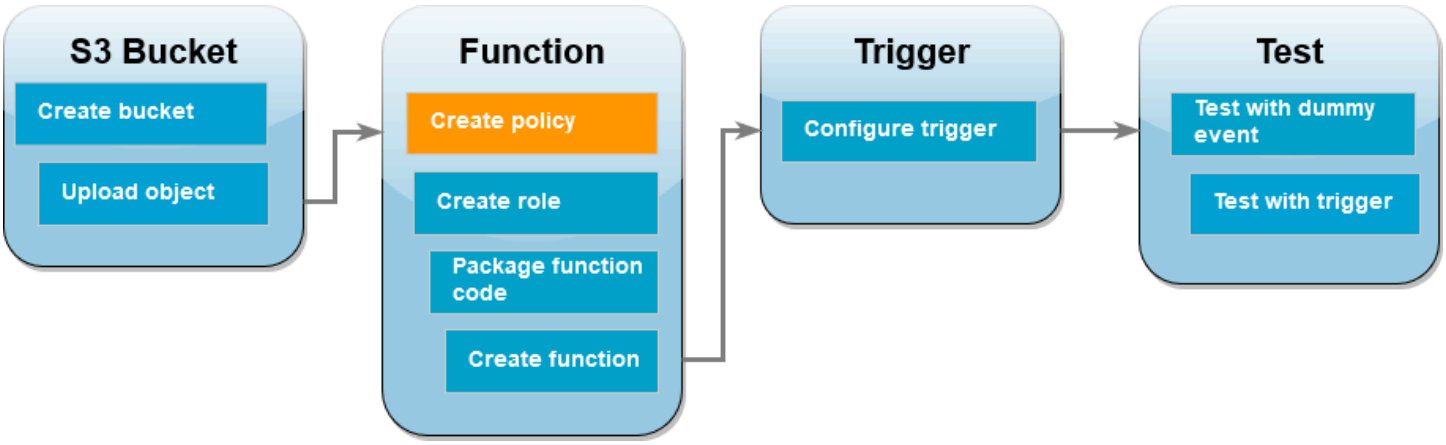
AWS CLI

Caricamento di un'immagine di test nel bucket di origine (AWS CLI)

- Dalla directory contenente l'immagine che desideri caricare, esegui il comando della CLI sotto riportato. Sostituisci il parametro `--bucket` con il nome del bucket di origine. Per i parametri `--key` e `--body`, usa il nome del file dell'immagine di test.

```
aws s3api put-object --bucket amzn-s3-demo-source-bucket --key HappyFace.jpg --
body ./HappyFace.jpg
```

Creazione di una policy di autorizzazione



Il primo passo per creare una funzione Lambda consiste nel creare una policy di autorizzazione. Questa politica fornisce alla funzione le autorizzazioni necessarie per accedere ad altre risorse. AWS Per questo tutorial, la policy fornisce le autorizzazioni di lettura e scrittura Lambda per i bucket Amazon S3 e consente di scrivere su Amazon Logs. CloudWatch

Console di gestione AWS

Creazione della policy (console)

1. Apri la pagina Policies della console (IAM). AWS Identity and Access Management
2. Scegli Crea policy.
3. Scegliere la scheda JSON e quindi incollare la seguente policy personalizzata nell'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. Scegli Next (Successivo).
5. In Dettagli sulla policy, per Nome policy, inserisci **`LambdaS3Policy`**.
6. Scegli Crea policy.

AWS CLI

Creazione della policy (AWS CLI)

1. Salva il seguente JSON in un file denominato `policy.json`.

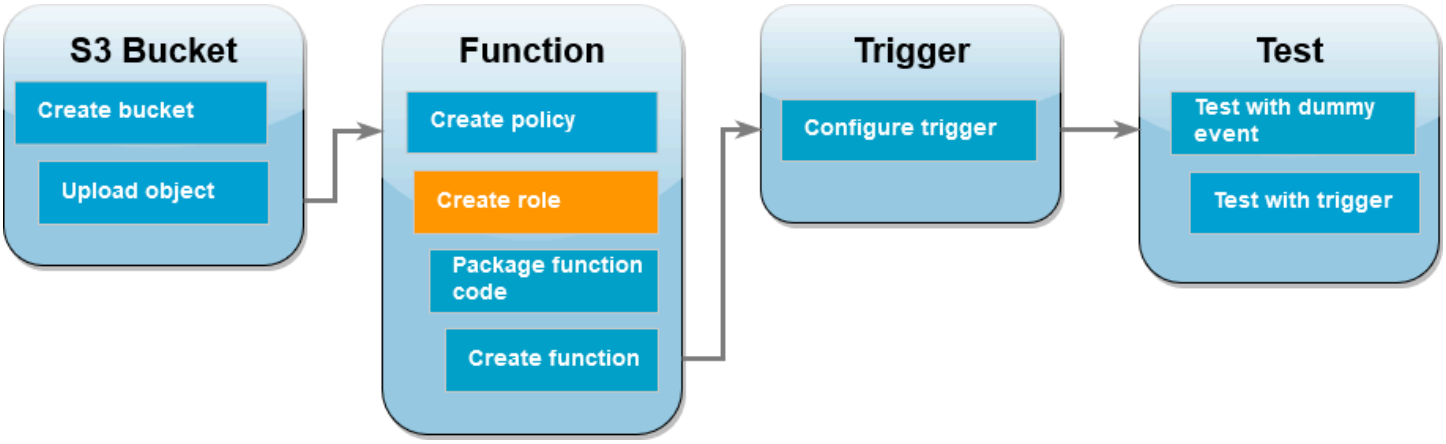
```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:CreateLogStream"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::*/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject"
            ],
            "Resource": "arn:aws:s3:::*/*"
        }
    ]
}
```

2. Nella directory in cui hai salvato il documento della policy JSON, esegui il comando della CLI sotto riportato.

```
aws iam create-policy --policy-name LambdaS3Policy --policy-document file://
policy.json
```

Creazione di un ruolo di esecuzione



Un ruolo di esecuzione è un ruolo IAM che concede a una funzione Lambda l'autorizzazione all' Servizi AWS accesso e alle risorse. Per concedere alla funzione l'accesso in lettura e scrittura a un bucket Amazon S3, è necessario collegare la policy di autorizzazione che hai creato nel passaggio precedente.

Console di gestione AWS

Creazione di un ruolo di esecuzione e collegamento di una policy di autorizzazione (console)

1. Apri la pagina Ruoli della console (IAM).
2. Scegli Crea ruolo.
3. Per Tipo di entità attendibile, seleziona Servizio AWS, mentre per Caso d'uso, seleziona Lambda.
4. Scegli Next (Successivo).
5. Aggiungi la policy di autorizzazione che hai creato nel passaggio precedente effettuando le seguenti operazioni:
   a. Nella casella di ricerca delle policy, immettere **`LambdaS3Policy`**.
   b. Nei risultati di ricerca, seleziona la casella di controllo per `LambdaS3Policy`.
   c. Scegli Next (Successivo).
6. In Dettagli ruolo, per Nome ruolo, inserisci **`LambdaS3Role`**.
7. Scegli Crea ruolo.

AWS CLI

Creazione di un ruolo di esecuzione e collegamento di una policy di autorizzazione (AWS CLI)

1. Salva il seguente JSON in un file denominato `trust-policy.json`. Questa politica di fiducia consente a Lambda di utilizzare le autorizzazioni del ruolo concedendo al servizio principale l'`lambda.amazonaws.com`autorizzazione a chiamare l'azione AWS Security Token Service ()AWS STS. `AssumeRole`

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

```
    ]
}
```

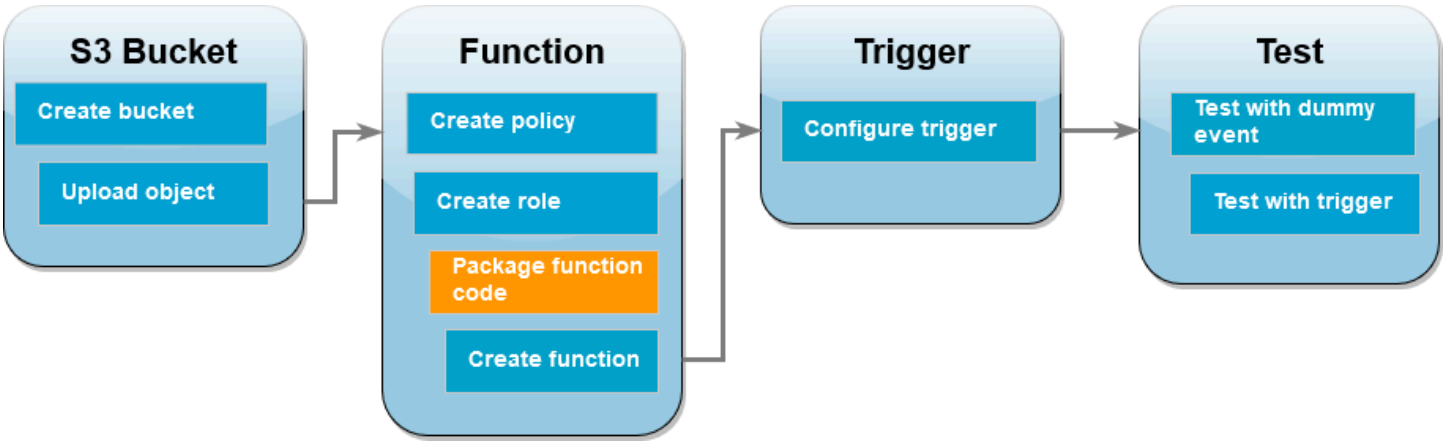
2. Nella directory in cui hai salvato il documento della policy di attendibilità JSON, esegui il comando della CLI sotto riportato per creare il ruolo di esecuzione.

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

3. Per collegare la policy di autorizzazione creata nel passaggio precedente, esegui il comando della CLI sotto riportato. Sostituisci il Account AWS numero nell'ARN della polizza con il tuo numero di conto.

```
aws iam attach-role-policy --role-name LambdaS3Role --policy-arn
 arn:aws:iam::123456789012:policy/LambdaS3Policy
```

Creazione del pacchetto di implementazione della funzione



Per creare la funzione, occorre creare un pacchetto di implementazione contenente la funzione e le rispettive dipendenze. Per questa funzione `CreateThumbnail`, il codice della funzione utilizza una libreria separata per il ridimensionamento dell'immagine. Segui le istruzioni per il linguaggio scelto per creare un pacchetto di implementazione contenente la libreria richiesta.

Node.js

Creazione del pacchetto di implementazione (Node.js)

1. Crea una directory denominata `lambda-s3` per il codice della funzione e le dipendenze e naviga al suo interno.

```
mkdir lambda-s3
cd lambda-s3
```

2. Crea un nuovo progetto Node.js con `npm`. Per accettare le opzioni predefinite fornite nell'esperienza interattiva, premi `Enter`.

```
npm init
```

3. Salvare il codice della funzione seguente in un file denominato `index.mjs`. Assicurati di sostituire `us-east-1` con la Regione AWS in cui hai creato i tuoi bucket di origine e destinazione.

```
// 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. Nella directory `lambda-s3`, installa la libreria sharp utilizzando npm. Tieni presente che l'ultima versione di sharp (0.33) non è compatibile con Lambda. Per completare questo tutorial, installa la versione 0.32.6.

```
npm install sharp@0.32.6
```

Il comando `install` di npm crea una directory `node_modules` per i tuoi moduli. Dopo questo passaggio, la struttura di directory dovrebbe avere un aspetto simile al seguente.

```
lambda-s3
|- index.mjs
|- node_modules
|  |- base64js
|  |- bl
|  |- buffer
...
|- package-lock.json
|- package.json
```

5. Crea un pacchetto di implementazione .zip contenente il codice della funzione e le rispettive dipendenze. Su MacOS o Linux, esegui il comando sotto riportato.

```
zip -r function.zip .
```

Su Windows, utilizza il tuo strumento di compressione preferito per creare il file .zip. Assicurati che i tuoi file `index.mjs`, `package.json` e `package-lock.json` e la tua directory `node_modules` si trovino tutti nella directory principale del tuo file .zip.

Python

Creazione del pacchetto di implementazione (Python)

1. Salva il codice di esempio come un file denominato `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. Nella stessa directory in cui hai creato il file `lambda_function.py`, crea una nuova directory denominata `package` e installa la libreria Pillow (PIL) e AWS SDK per Python (Boto3). Sebbene il runtime Lambda di Python includa una versione dell'SDK Boto3, ti consigliamo di aggiungere tutte le dipendenze della funzione al pacchetto di implementazione, anche se sono incluse nel runtime. Per ulteriori informazioni, consulta Dipendenze del runtime in Python.

   ```
   mkdir package
   pip install \
   --platform manylinux2014_x86_64 \
   --target=package \
   --implementation cp \
   --python-version 3.12 \
   --only-binary=:all: --upgrade \
   pillow boto3
   ```

   La libreria Pillow contiene codice C/C++. Utilizzando le opzioni `--platform manylinux_2014_x86_64` e `--only-binary=:all:`, pip scaricherà e installerà una versione di Pillow che contiene file binari precompilati compatibili con il sistema operativo Amazon Linux 2. Ciò garantisce che il pacchetto di implementazione funzioni nell'ambiente di esecuzione Lambda, indipendentemente dal sistema operativo e dall'architettura del computer di compilazione locale.

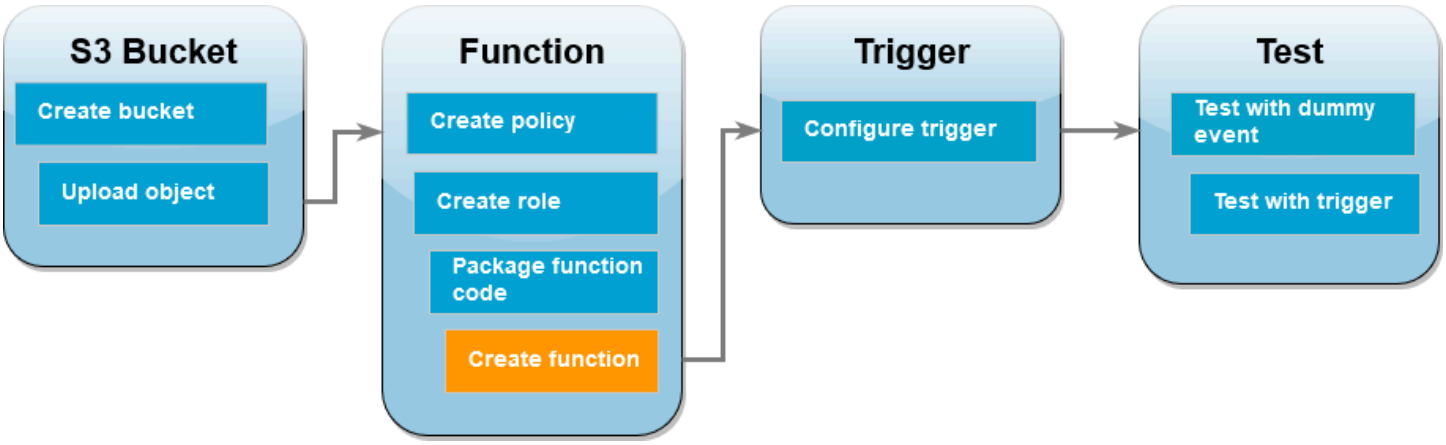
3. Crea un file .zip contenente il codice dell'applicazione e le librerie Pillow e Boto3. Su Linux o MacOS, esegui i comandi riportati di seguito dall'interfaccia della linea di comando.

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

   Su Windows, usa il tuo strumento di compressione preferito per creare il file `lambda_function.zip`. Assicurati che il tuo file `lambda_function.py` e le cartelle contenenti le tue dipendenze si trovino tutti nella directory principale del file .zip.

Puoi creare il tuo pacchetto di implementazione anche utilizzando un ambiente virtuale Python. Per informazioni, consultare Utilizzo di archivi di file .zip per le funzioni Lambda in Python.

Creazione della funzione Lambda



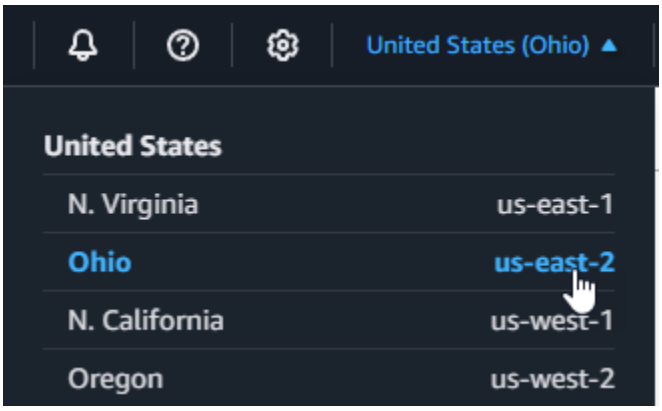
Puoi creare la tua funzione Lambda utilizzando la console AWS CLI o la console Lambda. Segui le istruzioni per il linguaggio scelto per creare la funzione.

Console di gestione AWS

Creazione della funzione (console)

Per creare la tua funzione Lambda utilizzando la console, devi prima creare una funzione di base contenente del codice "Hello world". Quindi, sostituisci questo codice con il codice della tua funzione caricando il file .zip o JAR creato nel passaggio precedente.

1. Aprire la pagina Funzioni della console Lambda.
2. Assicurati di lavorare nello stesso bucket in Regione AWS cui hai creato il tuo bucket Amazon S3. Puoi modificare la regione utilizzando l'elenco a discesa nella parte superiore dello schermo.



3. Selezionare Create function (Crea funzione).
4. Scegli Author from scratch (Crea da zero).

5. In Basic information (Informazioni di base) eseguire queste operazioni:
   a. Nel campo Function name (Nome funzione), immettere **`CreateThumbnail`**.
   b. Per Runtime, scegli Node.js 22.x o Python 3.12, a seconda del linguaggio che hai scelto per la funzione.
   c. In Architecture (Architettura), scegli x86_64.
6. Nella scheda Modifica ruolo di esecuzione predefinito, effettua le seguenti operazioni:
   a. Espandi la scheda, quindi scegli Utilizza un ruolo esistente.
   b. Seleziona il `LambdaS3Role` che hai creato in precedenza.
7. Scegli Crea funzione.

Caricamento del codice della funzione (console)

1. Nel riquadro Origine del codice, scegli Carica da.
2. Scegli File .zip.
3. Scegli Carica.
4. Nel selettore di file, seleziona il tuo file .zip e scegli Apri.
5. Scegli Save (Salva).

AWS CLI

Creazione della funzione (AWS CLI)

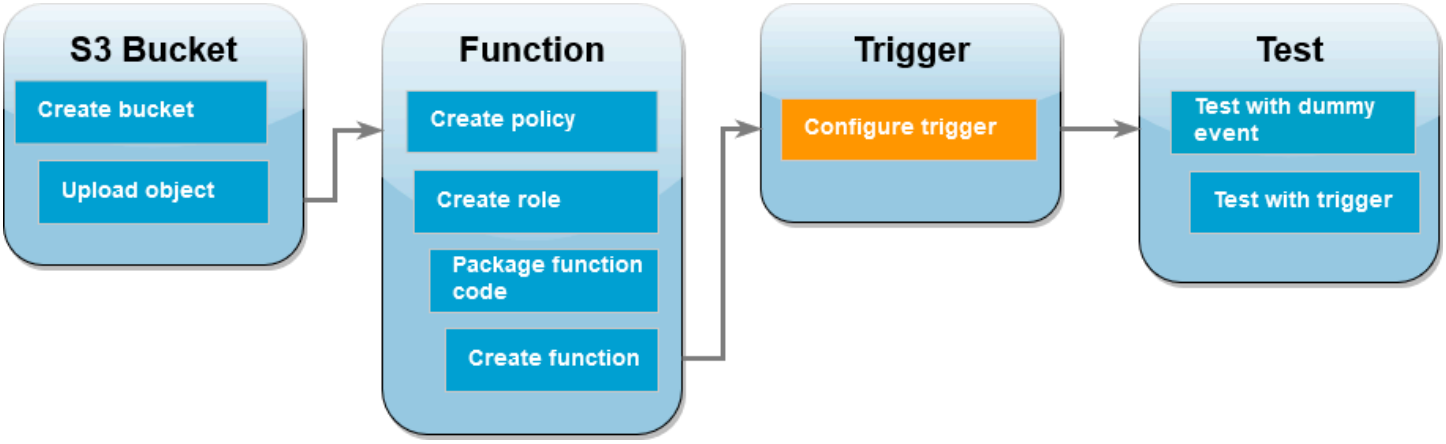
- Esegui il comando della CLI per il linguaggio che hai scelto. Per il `role` parametro, assicurati di sostituirlo `123456789012` con il tuo ID. Account AWS Per il parametro `region`, sostituisci `us-east-1` con la regione in cui hai creato i bucket Amazon S3.
  - Per Node.js, esegui il comando sotto riportato dalla directory contenente il file `function.zip`.

```
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
```

- Per Python, esegui il comando sotto riportato dalla directory contenente il file `lambda_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
```

Configurazione di Amazon S3 per richiamare la funzione



Affinché la funzione Lambda venga eseguita quando carichi un'immagine nel bucket di origine, devi configurare un trigger per la funzione. È possibile configurare il trigger Amazon S3 utilizzando la console Lambda o la AWS CLI.

> **Important**
>
> Questa procedura configura il bucket Amazon S3 per richiamare la funzione ogni volta che un oggetto viene creato nel bucket. Assicurati di configurare questa opzione solo sul bucket di origine. Se la tua funzione Lambda crea oggetti nello stesso bucket che la richiama, la tua funzione può essere richiamata continuamente in un ciclo ricorsivo (loop). Ciò può comportare la fatturazione di addebiti imprevisti al tuo Account AWS.

Console di gestione AWS

Configurazione del trigger Amazon S3 (console)

1. Apri la pagina Funzioni della console Lambda e scegli la tua funzione (`CreateThumbnail`).
2. Selezionare Add trigger (Aggiungi trigger).
3. Seleziona S3.
4. In Bucket, seleziona il tuo bucket di origine.
5. In Tipi di eventi, seleziona Tutti gli eventi di creazione di oggetti.
6. In Invocazione ricorsiva, seleziona la casella di controllo per confermare che non è consigliabile utilizzare lo stesso bucket Amazon S3 per input e output. Per maggiori informazioni sui modelli di invocazione ricorsivi in Lambda, consulta Schemi ricorsivi che causano loop indeterminati delle funzioni Lambda in Serverless Land.
7. Scegliere Aggiungi.

   Quando crei un trigger utilizzando la console Lambda, Lambda crea automaticamente una policy basata sulle risorse per concedere al servizio selezionato l'autorizzazione a richiamare la funzione.

AWS CLI

Configurazione del trigger Amazon S3 (AWS CLI)

1. Affinché il bucket di origine Amazon S3 richiami la funzione quando aggiungi un file di immagine, devi prima configurare le autorizzazioni per la funzione utilizzando una policy basata sulle risorse. Una dichiarazione politica basata sulle risorse fornisce altre Servizi AWS autorizzazioni per richiamare la funzione. Per autorizzare Amazon S3 a richiamare la tua funzione, esegui il comando della CLI comando sotto riportato. Assicurati di sostituire il `source-account` parametro con il tuo Account AWS ID e di utilizzare il tuo nome del bucket di origine.

```
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
```

La policy che definisci con questo comando consente ad Amazon S3 di richiamare la tua funzione solo quando viene eseguita un'operazione sul tuo bucket di origine.

> **Note**
>
> Sebbene i nomi dei bucket Amazon S3 siano univoci a livello globale, quando utilizzi policy basate sulle risorse è consigliabile specificare che il bucket deve appartenere al tuo account. Questo perché se elimini un bucket, è possibile che un altro lo Account AWS crei con lo stesso Amazon Resource Name (ARN).

2. Salva il seguente JSON in un file denominato `notification.json`. Quando viene applicato al tuo bucket di origine, questo JSON configura il bucket in modo che invii una notifica alla funzione Lambda ogni volta che viene aggiunto un nuovo oggetto. Sostituisci il Account AWS numero e Regione AWS nella funzione Lambda ARN con il tuo numero di account e la tua regione.

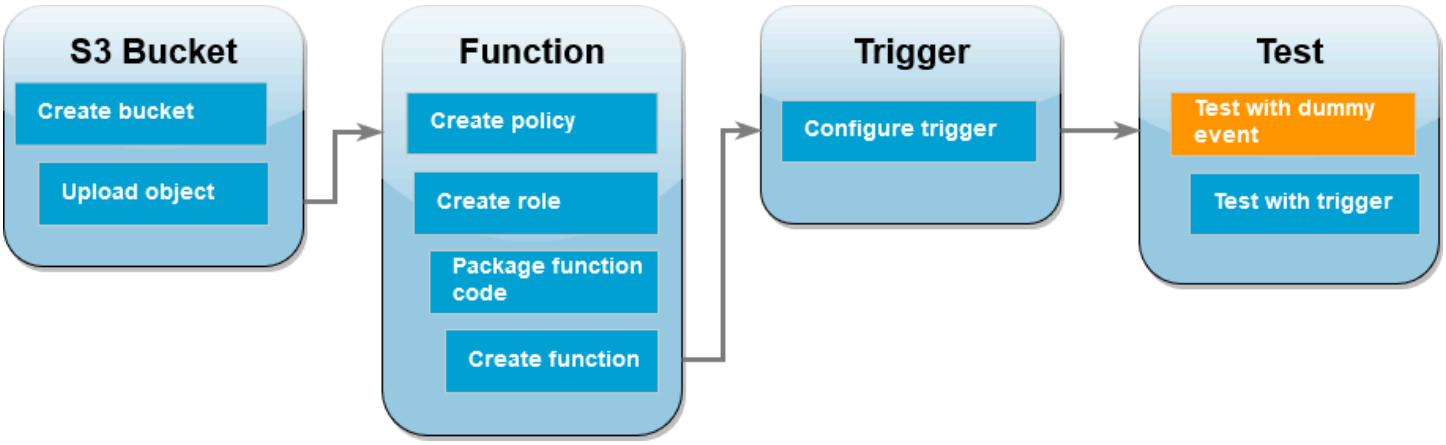
```
{
"LambdaFunctionConfigurations": [
    {
      "Id": "CreateThumbnailEventConfiguration",
      "LambdaFunctionArn": "arn:aws:lambda:us-
east-1:123456789012:function:CreateThumbnail",
      "Events": [ "s3:ObjectCreated:Put" ]
    }
  ]
}
```

3. Esegui il comando della CLI comando sotto riportato per applicare le impostazioni di notifica nel file JSON che hai creato al tuo bucket di origine. Sostituisci `amzn-s3-demo-source-bucket` con il nome del tuo bucket di origine.

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

Per ulteriori informazioni sul `put-bucket-notification-configuration` comando e sull'`notification-configuration`opzione, consulta put-bucket-notification-configurationla AWS CLI Command Reference.

Test di una funzione Lambda con un evento fittizio



Prima di testare l'intera configurazione aggiungendo un file di immagine al tuo bucket di origine Amazon S3, verifica che la tua funzione Lambda funzioni correttamente richiamandola con un evento fittizio. Un evento in Lambda è un documento in formato JSON che contiene i dati che una funzione deve elaborare. Quando la funzione viene richiamata da Amazon S3, l'evento inviato alla funzione contiene informazioni come il nome del bucket, l'ARN del bucket e la chiave dell'oggetto.

Console di gestione AWS

Test di una funzione Lambda con un evento fittizio (console)

1. Apri la pagina Funzioni della console Lambda e scegli la tua funzione (`CreateThumbnail`).
2. Seleziona la scheda Test.
3. Per creare il tuo evento di test, nel riquadro Evento di test, procedi come segue:
   a. In Operazione evento di test, seleziona Crea nuovo evento.
   b. Per Event name (Nome evento) immettere **`myTestEvent`**.
   c. Per Modello, seleziona S3 Put.
   d. Sostituisci i valori dei seguenti parametri con i tuoi valori.
      - Infatti`awsRegion`, `us-east-1` sostituiscilo con quello in Regione AWS cui hai creato i bucket Amazon S3.
      - Per `name`, sostituisci `amzn-s3-demo-bucket` con il nome del bucket di origine Amazon S3.

- Per `key`, sostituisci `test%2Fkey` con il nome del file dell'oggetto di test che hai caricato nel bucket di origine durante il passaggio Caricamento di un'immagine di test nel bucket di origine.

```
{
  "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. Scegli Save (Salva).

4. Nel riquadro Evento di test, scegli Test.
5. Per verificare che la funzione abbia creato una versione ridimensionata dell'immagine e l'abbia archiviata nel bucket Amazon S3 di destinazione, procedi come segue:

   a. Nella console Amazon S3, apri la pagina Bucket.

   b. Scegli il bucket di destinazione e conferma che il file ridimensionato sia elencato nel riquadro Oggetti.

AWS CLI

Test di una funzione Lambda con un evento fittizio (AWS CLI)

1. Salva il seguente JSON in un file denominato `dummyS3Event.json`. Sostituisci i valori dei seguenti parametri con i tuoi valori:

   - Infatti`awsRegion`, `us-east-1` sostituiscilo con quello in Regione AWS cui hai creato i bucket Amazon S3.
   - Per `name`, sostituisci `amzn-s3-demo-bucket` con il nome del bucket di origine Amazon S3.
   - Per `key`, sostituisci `test%2Fkey` con il nome del file dell'oggetto di test che hai caricato nel bucket di origine durante il passaggio Caricamento di un'immagine di test nel bucket di origine.

```
{
  "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. Nella directory in cui hai salvato il file dummyS3Event.json, richiama la funzione eseguendo il seguente comando della CLI. Questo comando richiama la funzione Lambda in modo sincrono specificando RequestResponse come valore del parametro invocation-type. Per ulteriori informazioni sulla chiamata sincrona e asincrona, consulta la pagina Invocazione delle funzioni Lambda.

```
aws lambda invoke --function-name CreateThumbnail \
--invocation-type RequestResponse --cli-binary-format raw-in-base64-out \
--payload file://dummyS3Event.json outputfile.txt
```

L' cli-binary-formatopzione è obbligatoria se si utilizza la versione 2 di. AWS CLI Per rendere questa impostazione come predefinita, esegui aws configure set cli-binary-format raw-in-base64-out. Per ulteriori informazioni, consulta Opzioni della riga di comando globali supportate da AWS CLI.

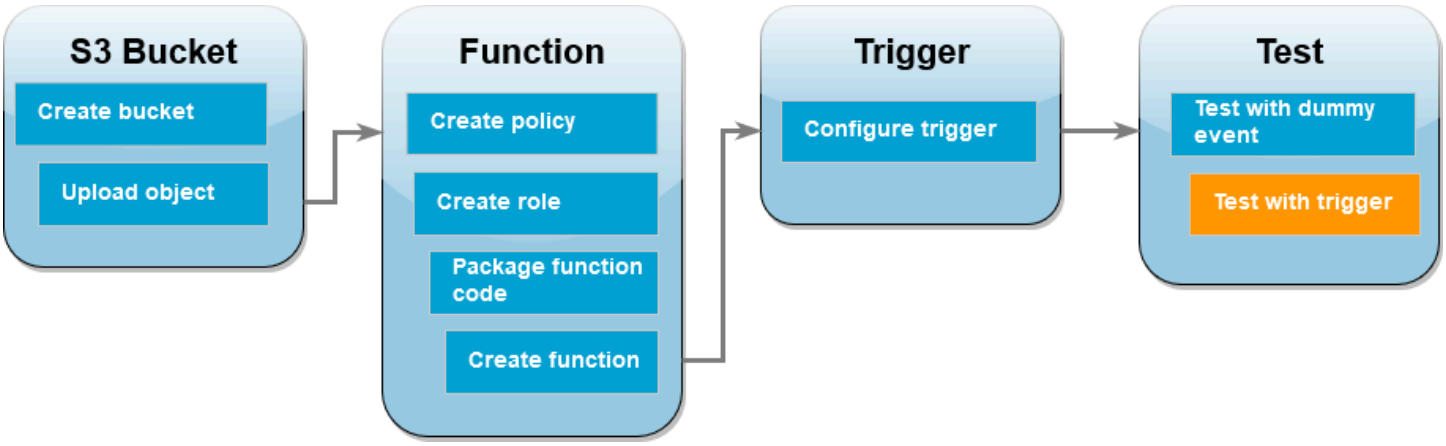
3. Verifica che la funzione abbia creato una versione in miniatura dell'immagine e l'abbia salvata nel bucket Amazon S3 di destinazione. Esegui il comando della CLI sotto riportato sostituendo `amzn-s3-demo-source-bucket-resized` con il nome del tuo bucket di destinazione.

```
aws s3api list-objects-v2 --bucket amzn-s3-demo-source-bucket-resized
```

Verrà visualizzato un output simile al seguente. Il parametro `Key` mostra il nome del file di immagine ridimensionato.

```
{
    "Contents": [
        {
            "Key": "resized-HappyFace.jpg",
            "LastModified": "2023-06-06T21:40:07+00:00",
            "ETag": "\"d8ca652ffe83ba6b721ffc20d9d7174a\"",
            "Size": 2633,
            "StorageClass": "STANDARD"
        }
    ]
}
```

Test della funzione tramite il trigger Amazon S3



Ora che hai verificato che la funzione Lambda funziona correttamente, puoi testare la configurazione completa aggiungendo un file di immagine al tuo bucket di origine Amazon S3. Quando aggiungi l'immagine al bucket di origine, la tua funzione Lambda dovrebbe essere richiamata

automaticamente. La tua funzione crea una versione ridimensionata del file e la archivia nel bucket di destinazione.

Console di gestione AWS

Test della funzione Lambda tramite il trigger Amazon S3 (console)

1. Per caricare un'immagine nel bucket Amazon S3, procedi come segue:
   a. Apri la pagina Bucket della console Amazon S3 e scegli il bucket di origine.
   b. Scegli Carica.
   c. Scegli Aggiungi file e utilizza il selettore di file per scegliere il file di immagine da caricare. L'oggetto dell'immagine può essere qualsiasi file .jpg o .png.
   d. Seleziona Apri, quindi Carica.
2. Verifica che Lambda abbia salvato una versione ridimensionata del tuo file di immagine nel bucket di destinazione effettuando le seguenti operazioni:
   a. Torna alla pagina Bucket della console Amazon S3 e scegli il bucket di destinazione.
   b. Nel riquadro Oggetti, ora dovresti vedere due file di immagine ridimensionati, uno per ogni test della tua funzione Lambda. Per scaricare l'immagine ridimensionata, seleziona il file, quindi scegli Scarica.

AWS CLI

Test della funzione Lambda tramite il trigger Amazon S3 (AWS CLI)

1. Dalla directory contenente l'immagine che desideri caricare, esegui il comando della CLI sotto riportato. Sostituisci il parametro `--bucket` con il nome del bucket di origine. Per i parametri `--key` e `--body`, usa il nome del file dell'immagine di test. L'immagine di test può essere qualsiasi file .jpg o .png.

```
aws s3api put-object --bucket amzn-s3-demo-source-bucket --key SmileyFace.jpg --
body ./SmileyFace.jpg
```

2. Verifica che la funzione abbia creato una versione in miniatura dell'immagine e l'abbia salvata nel bucket Amazon S3 di destinazione. Esegui il comando della CLI sotto riportato sostituendo `amzn-s3-demo-source-bucket-resized` con il nome del tuo bucket di destinazione.

```
aws s3api list-objects-v2 --bucket amzn-s3-demo-source-bucket-resized
```

Se la tua funzione viene eseguita correttamente, vedrai un output simile al seguente. Il bucket di destinazione dovrebbe ora contenere due file ridimensionati.


```
{
    "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"
        }
    ]
}
```


Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili a tuo Account AWS carico.

Per eliminare la funzione Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Digita **`confirm`** nel campo di immissione testo e scegli Delete (Elimina).

Per eliminare la policy creata

1. Aprire la pagina Policies (Policy) nella console IAM.
2. Seleziona la politica che hai creato (AWSLambdaS3Policy).
3. Scegliere Policy actions (Operazioni policy), Delete (Elimina).
4. Scegliere Delete (Elimina).

Per eliminare il ruolo di esecuzione

1. Aprire la pagina Ruoli della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

Per eliminare il bucket S3

1. Aprire la console Amazon S3.
2. Selezionare il bucket creato in precedenza.
3. Scegli Elimina.
4. Inserisci il nome del bucket nel campo di immissione testo.
5. Scegli Delete Bucket (Elimina bucket).

Uso dei segreti in Secrets Manager nelle funzioni Lambda

Gestione dei segreti AWS ti aiuta a gestire credenziali, chiavi API e altri segreti necessari alle tue funzioni Lambda. Esistono due approcci principali per il recupero dei segreti nelle funzioni Lambda, entrambi offrono prestazioni migliori e costi inferiori rispetto al recupero diretto dei segreti tramite l'SDK: AWS

- AWS parametri e segreti Estensione Lambda: una soluzione indipendente dal runtime che fornisce una semplice interfaccia HTTP per il recupero dei segreti
- Powertools for AWS Lambda parameters utility: una soluzione integrata nel codice che supporta più provider (Secrets Manager, Parameter Store AppConfig) con trasformazioni integrate

Entrambi gli approcci mantengono cache locali di segreti, eliminando la necessità per la funzione di chiamare Secrets Manager per ogni invocazione. Quando la funzione richiede un segreto, la cache viene controllata per prima. Se il segreto è disponibile e non è scaduto, viene restituito immediatamente. Altrimenti, viene recuperato da Secrets Manager, memorizzato nella cache e restituito. Questo meccanismo di memorizzazione nella cache consente tempi di risposta più rapidi e costi ridotti riducendo al minimo le chiamate API.

Scelta di un approccio

Considerate questi fattori quando scegliete tra l'estensione e: PowerTools

Usa l'estensione Lambda AWS Parametri e segreti quando:

- Vuoi una soluzione indipendente dal runtime che funzioni con qualsiasi runtime Lambda
- Preferisci non aggiungere dipendenze di codice alla tua funzione
- Devi solo recuperare i segreti da Secrets Manager o Parameter Store

Utilizzate Powertools per l'utilità AWS Lambda dei parametri quando:

- Vuoi un'esperienza di sviluppo integrata con il codice dell'applicazione
- È necessario il supporto per più provider (Secrets Manager, Parameter Store, AppConfig)
- Desideri trasformazioni di dati integrate (analisi JSON, decodifica base64)
- Stai usando i runtime Python TypeScript, Java o.NET

Quando usare Secrets Manager con Lambda

Gli scenari più comuni per l'utilizzo di Secrets Manager con Lambda includono:

- Memorizzazione delle credenziali del database utilizzate dalla funzione per connettersi ad Amazon RDS o ad altri database
- Gestione delle chiavi API per i servizi esterni chiamati dalle tue funzioni
- Memorizzazione delle chiavi di crittografia o di altri dati di configurazione sensibili
- Rotazione automatica delle credenziali senza dover aggiornare il codice della funzione

Utilizzo dei AWS parametri e dei segreti (estensione Lambda)

L'estensione Lambda AWS Parameters and Secrets utilizza una semplice interfaccia HTTP compatibile con qualsiasi runtime Lambda. Per impostazione predefinita, memorizza nella cache i segreti per 300 secondi (5 minuti) e può contenere fino a 1.000 segreti. Puoi personalizzare queste impostazioni con variabili di ambiente.

Usa Secrets Manager in una funzione Lambda

Questa sezione presuppone che tu abbia già un segreto Secrets Manager. Per creare un segreto, vedi Creare un Gestione dei segreti AWS segreto.

Creazione del pacchetto di implementazione

Scegli il tuo runtime preferito e segui i passaggi per creare una funzione che recuperi segreti da Secrets Manager. La funzione di esempio recupera un segreto da Secrets Manager e può essere utilizzata per accedere alle credenziali del database, alle chiavi API o ad altri dati di configurazione sensibili nelle applicazioni.

Python

Per creare una funzione Python

1. Crea e vai in una nuova directory di progetto. Esempio:

```
mkdir my_function
cd my_function
```

2. Crea un file denominato `lambda_function.py` con il codice seguente. Per `secret_name`, utilizza il nome o il nome della risorsa Amazon (ARN) del segreto.

```
import json
import os
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"

        secrets_extension_endpoint = f"http://localhost:2773/secretsmanager/get?
secretId={secret_name}"
        headers = {"X-Aws-Parameters-Secrets-Token":
 os.environ.get('AWS_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. Creare un file denominato `requirements.txt` con il seguente contenuto:

```
requests
```

4. Installare le dipendenze:

```
pip install -r requirements.txt -t .
```

5. Crea un file .zip contenente tutti i file:

```
zip -r function.zip .
```

Node.js

Per creare una funzione Node.js.

1. Crea e vai in una nuova directory di progetto. Esempio:

```
mkdir my_function
cd my_function
```

2. Crea un file denominato `index.mjs` con il codice seguente. Per `secret_name`, utilizza il nome o il nome della risorsa Amazon (ARN) del segreto.

```
import http from 'http';

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";
        const options = {
            hostname: 'localhost',
            port: 2773,
            path: `/secretsmanager/get?secretId=${secretName}`,
            headers: {
                'X-Aws-Parameters-Secrets-Token': process.env.AWS_SESSION_TOKEN
            }
        };

        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. Crea un file .zip contenente il file `index.mjs`:

```
zip -r function.zip index.mjs
```

Java

Per creare una funzione Java

1. Crea un progetto Maven:

```
mvn archetype:generate \
    -DgroupId=example \
    -DartifactId=lambda-secrets-demo \
    -DarchetypeArtifactId=maven-archetype-quickstart \
    -DarchetypeVersion=1.4 \
    -DinteractiveMode=false
```

2. Vai alla directory del progetto:

```
cd lambda-secrets-demo
```

3. Apri `pom.xml` e sostituisci il contenuto con quanto riportato di seguito.

```
<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>
    </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. Rinomina `/lambda-secrets-demo/src/main/java/example/App.java` in `Hello.java` in modo che corrisponda al nome del gestore Java predefinito di Lambda (`example.Hello::handleRequest`):

```
mv src/main/java/example/App.java src/main/java/example/Hello.java
```

5. Apri il file `Hello.java` e sostituisci il contenuto con quanto riportato di seguito. Per `secretName`, utilizza il nome o il nome della risorsa Amazon (ARN) del segreto.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
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();

    @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;

            HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(endpoint))
                .header("X-Aws-Parameters-Secrets-Token",
 System.getenv("AWS_SESSION_TOKEN"))
                .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. Rimuovi la directory di test. Maven la crea per impostazione predefinita, ma non ne abbiamo bisogno per questo esempio.

```
rm -rf src/test
```

7. Compila il progetto:

```
mvn package
```

8. Scarica il file JAR (`target/function.jar`) per un uso successivo.

Creazione della funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli Crea funzione.
3. Scegli Crea da zero.
4. Nel campo Function name (Nome funzione), immettere **`secret-retrieval-demo`**.
5. Scegli il runtime preferito.
6. Scegli Crea funzione.

Per caricare il pacchetto di implementazione

1. Nella scheda Codice della funzione, scegli Carica da e seleziona il file.zip (per Python e Node.js) o.jar (per Java).
2. Carica il pacchetto di implementazione creato in precedenza.
3. Scegli Save (Salva).

Aggiunta dell'estensione

Per aggiungere l'estensione Lambda AWS Parameters and Secrets come livello

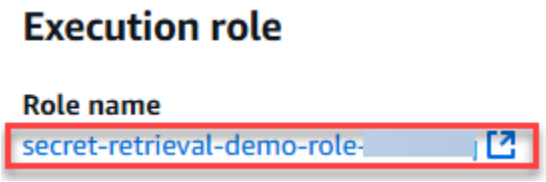
1. Nella scheda Codice della funzione, scorri verso il basso fino a Livelli.
2. Scegliere Add a layer (Aggiungi un livello).
3. Seleziona AWS livelli.
4. Scegli AWS-Parameters-and-Secrets-Lambda-Extension.
5. Scegli la versione più recente.
6. Scegliere Aggiungi.

Aggiunta di autorizzazioni

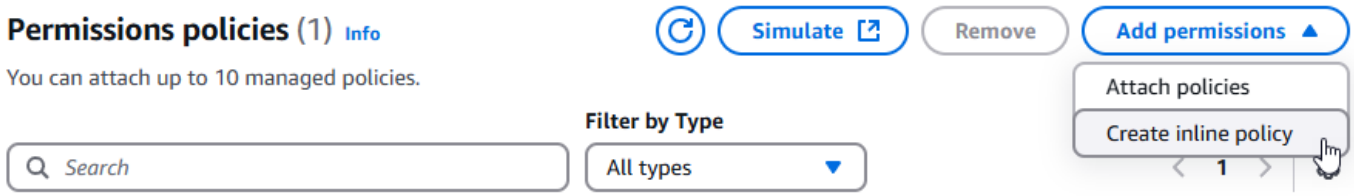
Per aggiungere i permessi di Secrets Manager al ruolo di esecuzione

1. Quindi, seleziona la scheda Configuration (Configurazione) e poi Permissions (Autorizzazioni).

2. In Nome del ruolo, scegli il link al tuo ruolo di esecuzione. Questo ruolo si apre nella console IAM.



3. Scegli Aggiungi autorizzazioni, quindi seleziona Crea policy in linea.



4. Scegli la scheda JSON e aggiungi la policy seguente. Per `Resource`, inserisci l'ARN del tuo segreto.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": "arn:aws:secretsmanager:us-
east-1:111122223333:secret:SECRET_NAME"
        }
    ]
}
```

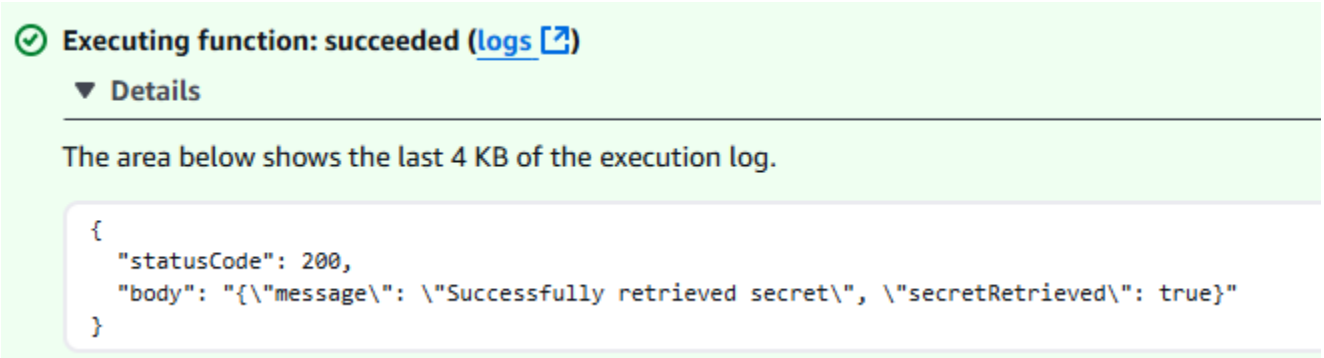
5. Scegli Next (Successivo).
6. Inserisci un nome per la policy.
7. Scegli Crea policy.

Test della funzione

Per testare la funzione

1. Torna alla console Lambda.

2. Seleziona la scheda Test.
3. Scegli Test (Esegui test). Dovrebbe essere visualizzata la seguente risposta:



Variabili di ambiente

L'estensione Lambda AWS Parameters and Secrets utilizza le seguenti impostazioni predefinite. Puoi sovrascrivere queste impostazioni creando le variabili ambiente corrispondenti. Per visualizzare le impostazioni correnti di una funzione, imposta `PARAMETERS_SECRETS_EXTENSION_LOG_LEVEL` su `DEBUG`. L'estensione registrerà le informazioni di configurazione in CloudWatch Logs all'inizio di ogni chiamata di funzione.

Impostazione	Valore predefinito	Valori validi	Variabile di ambiente	Informazioni
Porta HTTP	2773	1 - 65535	PARAMETERS_SECRETS_EXTENSION_HTTP_PORT	Porta per il server HTTP locale
Cache abilitata	TRUE	TRUE | FALSE	PARAMETERS_SECRETS_EXTENSION_CACHE_ENABLED	Abilita o disabilita la cache
Dimensioni della cache	1000	0 - 1.000	PARAMETERS_SECRETS_EXTENSION_CACHE_SIZE	Imposta su 0 per disabilitare la memorizzazione nella cache
TTL di Secrets Manager.	300 secondi	0 - 300 secondi	SECRETS_MANAGER_TTL	Time-to-live per segreti memorizzati nella cache. Imposta su

Impostazione	Valore predefinito	Valori validi	Variabile di ambiente	Informazioni
				0 per disabilitare la memorizzazione nella cache. Questa variabile viene ignorata se il valore per PARAMETERS_SECRETS_EXTENSION_CACHE_SIZE è 0.
TTL Parameter Store	300 secondi	0 - 300 secondi	SSM_PARAMETER_STORE_TTL	Time-to-live per i parametri memorizzati nella cache. Imposta su 0 per disabilitare la memorizzazione nella cache. Questa variabile viene ignorata se il valore per PARAMETERS_SECRETS_EXTENSION_CACHE_SIZE è 0.
Livello di log	INFO	DEBUG | INFO | WARN | ERROR | NONE	PARAMETERS_SECRETS_EXTENSION_LOG_LEVEL	Il livello di dettaglio riportato nei log per l'estensione
Connessioni max	3	Uguale o maggiore di 1	PARAMETERS_SECRETS_EXTENSION_MAX_CONNECTIONS	Numero massimo di connessioni HTTP per richieste a Parameter Store o Secrets Manager
Timeout di Secrets Manager	0 (nessun timeout)	Tutti i numeri interi	SECRETS_MANAGER_TIMEOUT_MILLIS	Timeout per le richieste a Secrets Manager (in millisecondi)

Impostazione	Valore predefinito	Valori validi	Variabile di ambiente	Informazioni
Timeout Parameter Store	0 (nessun timeout)	Tutti i numeri interi	SSM_PARAMETER_STORE_TIMEOUT_MILLIS	Timeout per le richieste a Parameter Store (in millisecondi).

Operazioni con rotazione di un segreto

Se ruoti spesso i segreti, la durata predefinita di 300 secondi della cache potrebbe far sì che la funzione utilizzi segreti obsoleti. Sono disponibili due opzioni per garantire che la funzione utilizzi il valore segreto più recente:

- Ridurre il TTL della cache impostando la variabile di ambiente `SECRETS_MANAGER_TTL` su un valore inferiore (in secondi). Ad esempio, impostandolo su `60` garantire che la funzione non utilizzi mai un segreto vecchio più di un minuto.
- Usare le etichette temporanee `AWSCURRENT` o `AWSPREVIOUS` nella tua richiesta di segreto per assicurarti di ottenere la versione specifica che desideri:

```
secretsmanager/get?secretId=YOUR_SECRET_NAME&versionStage=AWSCURRENT
```

Scegli l'approccio che meglio bilancia le tue esigenze di prestazioni e aggiornamento. Un TTL inferiore significa chiamate più frequenti a Secrets Manager, ma garantisce l'utilizzo dei valori segreti più recenti.

Utilizzo dell'utilità di parametrizzazione di Powertools per AWS Lambda

L'utilità dei parametri di Powertools for AWS Lambda fornisce un'interfaccia unificata per il recupero di segreti da più provider, tra cui Secrets Manager, Parameter Store e. AppConfig Gestisce la memorizzazione nella cache, le trasformazioni e offre un'esperienza di sviluppo più integrata rispetto all'approccio di estensione.

Vantaggi dell'utilità parametri

- Provider multipli: recupera i parametri da Secrets Manager, Parameter Store e AppConfig utilizza la stessa interfaccia
- Trasformazioni integrate: analisi JSON automatica, decodifica base64 e altre trasformazioni dei dati

- Memorizzazione nella cache integrata: memorizzazione nella cache configurabile con supporto TTL per ridurre le chiamate API
- Sicurezza dei tipi: forte supporto alla digitazione in altri TypeScript runtime supportati
- Gestione degli errori: logica di ripetizione dei tentativi e gestione degli errori integrate

Esempi di codice

Negli esempi seguenti viene illustrato come recuperare i segreti utilizzando l'utilità Parametri in runtime diversi:

Python

> **Note**
>
> Per esempi completi e istruzioni di configurazione, consulta la documentazione dell'utilità Parametri.

Recupero di segreti da Secrets Manager con l'utilità Powertools for AWS Lambda Parameters.

```
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**
>
> Per esempi completi e istruzioni di configurazione, consulta la documentazione dell'utilità Parametri.

Recupero di segreti da Secrets Manager con l'utilità Powertools for AWS Lambda Parameters.

```
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**
>
> Per esempi completi e istruzioni di configurazione, consulta la documentazione dell'utilità Parametri.

Recupero di segreti da Secrets Manager con l'utilità Powertools for AWS Lambda Parameters.

```
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**
>
> Per esempi completi e istruzioni di configurazione, consulta la documentazione dell'utilità Parametri.

Recupero di segreti da Secrets Manager con l'utilità Powertools for AWS Lambda Parameters.

```
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
    }
}
```

Configurazione e autorizzazioni

Per utilizzare l'utilità Parametri, devi:

1. Installa Powertools per AWS Lambda il tuo runtime. Per informazioni dettagliate, vedi Utensili elettrici per AWS Lambda.
2. Aggiungere le autorizzazioni IAM necessarie al ruolo di esecuzione della tua funzione. Fare riferimento a Gestione delle autorizzazioni in AWS Lambda per ulteriori dettagli.
3. Configura eventuali impostazioni opzionali tramite variabili di ambiente.

Le autorizzazioni IAM richieste sono le stesse dell'approccio di estensione. L'utilità gestirà automaticamente la memorizzazione nella cache e le chiamate API a Secrets Manager in base alla configurazione dell'utente.

Utilizzo di Lambda con Amazon SQS

> **Note**
>
> Se desideri inviare dati a una destinazione diversa da una funzione Lambda o arricchire i dati prima di inviarli, consulta Amazon Pipes. EventBridge

È possibile utilizzare una funzione Lambda per elaborare i messaggi in una coda Amazon Simple Queue Service (Amazon SQS). Lambda supporta sia code standard che code first-in, first-out (FIFO) per gli strumenti di mappatura dell'origine degli eventi. Puoi anche utilizzare la modalità provisioned per allocare risorse di polling dedicate per le mappature delle sorgenti degli eventi Amazon SQS. La funzione Lambda e la coda Amazon SQS devono trovarsi nella Regione AWS stessa posizione, anche se possono trovarsi in posizioni diverse. Account AWS

Durante l'elaborazione di messaggi Amazon SQS, è necessario implementare una logica di risposta batch parziale per evitare che i messaggi elaborati correttamente vengano ritentati quando alcuni messaggi di un batch falliscono. L'utilità Batch Processor di Powertools AWS Lambda semplifica questa implementazione gestendo automaticamente la logica di risposta batch parziale, riducendo i tempi di sviluppo e migliorando l'affidabilità.

Argomenti

- Informazioni sul comportamento di polling e batch per gli strumenti di mappatura dell'origine degli eventi di Amazon SQS
- Utilizzo della modalità provisioning con le mappature delle sorgenti degli eventi di Amazon SQS
- Configurazione della modalità di provisioning per la mappatura delle sorgenti degli eventi Amazon SQS
- Esempio di evento con messaggio di coda standard
- Esempio di evento di messaggio di coda FIFO
- Creazione e configurazione di uno strumento di mappatura dell'origine degli eventi Amazon SQS
- Configurazione del comportamento di dimensionamento per gli strumenti di mappatura dell'origine degli eventi
- Gestione degli errori per un'origine eventi SQS in Lambda
- Parametri Lambda per gli strumenti di mappatura dell'origine degli eventi di Amazon SQS
- Utilizzo del filtro eventi con un'origine eventi Amazon SQS

- Tutorial: Utilizzo di Lambda con Amazon SQS
- Tutorial: utilizzo di una coda Amazon SQS tra più account come origine eventi

Informazioni sul comportamento di polling e batch per gli strumenti di mappatura dell'origine degli eventi di Amazon SQS

Con gli strumenti di mappatura dell'origine degli eventi di Amazon SQS, Lambda esegue il polling della coda e richiama la funzione in modo sincrono con un evento. Ogni evento può contenere un batch di più messaggi dalla coda. Lambda riceve questi eventi un batch alla volta e richiama la funzione una volta per ogni batch. Quando la funzione elabora correttamente un batch, Lambda elimina i relativi messaggi dalla coda.

Quando Lambda legge un batch, i messaggi rimangono nella coda ma vengono nascosti per la durata del timeout visibilità della coda. Se la funzione elabora correttamente tutti i messaggi nel batch, Lambda elimina i messaggi dalla coda. Per impostazione predefinita, se la funzione rileva un errore durante l'elaborazione di un batch, una volta scaduto il timeout di visibilità tutti i messaggi in quel batch diventeranno nuovamente visibili nella coda. Per questo motivo, il codice della funzione deve riuscire a elaborare lo stesso messaggio più volte, senza intoppi indesiderati.

> **Warning**
>
> Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center. AWS

Per impedire a Lambda di elaborare un messaggio più volte, puoi configurare la mappatura dell'origine degli eventi per includere gli errori degli elementi in batch nella risposta della funzione oppure puoi utilizzare l'DeleteMessageAPI per rimuovere i messaggi dalla coda man mano che la funzione Lambda li elabora correttamente.

Per ulteriori informazioni sui parametri di configurazione supportati da Lambda per gli strumenti di mappatura dell'origine degli eventi di SQS, consulta the section called “Creazione di uno strumento di mappatura dell'origine degli eventi SQS”.

Utilizzo della modalità provisioning con le mappature delle sorgenti degli eventi di Amazon SQS

Per i carichi di lavoro in cui è necessario ottimizzare il throughput dello strumento di mappatura dell'origine degli eventi, è possibile utilizzare la modalità provisioning. In modalità provisioning, vengono definiti i limiti minimi e massimi per la quantità di poller di eventi assegnati. Questi poller di eventi con provisioning sono dedicati allo strumento di mappatura dell'origine degli eventi e possono gestire picchi di messaggi imprevisti tramite un dimensionamento automatico reattivo. La mappatura delle sorgenti di eventi di Amazon SQS configurata con la modalità Provisioned è 3 volte più veloce (fino a 1.000 richiami simultanei al minuto) e supporta una contemporaneità 16 volte superiore (fino a 20.000 richiami simultanei) rispetto alla funzionalità di mappatura delle sorgenti di eventi di Amazon SQS predefinita. Ti consigliamo di utilizzare la modalità provisioned per i carichi di lavoro basati su eventi di Amazon SQS che hanno requisiti prestazionali rigorosi, come le società di servizi finanziari che elaborano i feed di dati di mercato, le piattaforme di e-commerce che forniscono consigli personalizzati in tempo reale e le società di gioco che gestiscono le interazioni tra giocatori dal vivo. L'utilizzo della modalità provisioning comporta costi aggiuntivi. Per i prezzi dettagliati, consulta la pagina dei prezzi.AWS Lambda

Ogni event poller in modalità provisioned può gestire fino all'1% MB/s del throughput, fino a 10 chiamate simultanee o fino a 10 chiamate API di polling di Amazon SQS al secondo. L'intervallo di valori accettati per il numero minimo di event poller (MinimumPollers) è compreso tra 2 e 200, con il valore predefinito di 2. L'intervallo di valori accettati per il numero massimo di event poller (MaximumPollers) è compreso tra 2 e 2.000, con il valore predefinito di 200. MaximumPollers deve essere maggiore o uguale a. MinimumPollers

Determinazione dei sondaggi di eventi richiesti

Per stimare il numero di event poller necessari per garantire prestazioni ottimali di elaborazione dei messaggi quando si utilizza la modalità provisioned per SQS ESM, raccogli le seguenti metriche per la tua applicazione: eventi SQS di picco al secondo che richiedono un'elaborazione a bassa latenza, dimensione media del payload degli eventi SQS, durata media della funzione Lambda e dimensione del batch configurato.

Innanzitutto puoi stimare il numero di eventi SQS al secondo (EPS) supportati da un event poller per il tuo carico di lavoro utilizzando la seguente formula:

```
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)
                )
```

Quindi, puoi calcolare il numero minimo di poller richiesti utilizzando la formula seguente. Questo calcolo garantisce la disponibilità di una capacità sufficiente per gestire i requisiti di traffico di picco.

```
Required event pollers = (Peak number of events per second in Queue) / EPS per event
 poller
```

Prendiamo in considerazione un carico di lavoro con una dimensione del batch predefinita di 10, una dimensione media degli eventi di 3 KB, una durata media della funzione di 100 ms e un requisito per gestire 1.000 eventi al secondo. In questo scenario, ogni event poller supporterà circa 100 eventi al secondo (EPS). Pertanto, è necessario impostare un numero minimo di poller su 10 per gestire adeguatamente i requisiti di traffico di picco. Se il carico di lavoro ha le stesse caratteristiche ma la durata media della funzione è di 1 secondo, ogni poller supporterà solo 10 EPS, il che richiede la configurazione di almeno 100 poller per supportare 1.000 eventi al secondo a bassa latenza.

Ti consigliamo di utilizzare una dimensione batch predefinita pari o superiore a 10 per massimizzare l'efficienza dei poller di eventi in modalità provisioned. Le dimensioni dei batch più elevate consentono a ciascun poller di elaborare più eventi per chiamata, migliorando la produttività e l'efficienza in termini di costi. Quando pianificate la capacità dei poller del vostro evento, tenete conto dei potenziali picchi di traffico e valutate la possibilità di impostare il valore MinimumPollers leggermente più alto del minimo calcolato per fornire un buffer. Inoltre, monitora le caratteristiche del carico di lavoro nel tempo, poiché le modifiche alla dimensione dei messaggi, alla durata delle funzioni o ai modelli di traffico potrebbero richiedere modifiche alla configurazione del poller degli eventi per mantenere prestazioni ed efficienza dei costi ottimali. Per una pianificazione precisa della capacità, consigliamo di testare il carico di lavoro specifico per determinare l'EPS effettivo che ogni event poller può generare.

Configurazione della modalità di provisioning per la mappatura delle sorgenti degli eventi Amazon SQS

Puoi configurare la modalità di provisioning per la mappatura delle sorgenti degli eventi Amazon SQS utilizzando la console o l'API Lambda.

Per configurare la modalità di provisioning per una mappatura delle sorgenti di eventi Amazon SQS esistente (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la funzione con la mappatura delle sorgenti degli eventi di Amazon SQS per cui desideri configurare la modalità di provisioning.
3. Scegli la scheda Configurazione, quindi scegli Trigger.
4. Scegli la mappatura delle sorgenti degli eventi Amazon SQS per cui desideri configurare la modalità di provisioning, quindi scegli Modifica.
5. In Configurazione dello strumento di mappatura dell'origine degli eventi, scegli Configura la modalità provisioning.
   - Per Minimum event poller, inserisci un valore compreso tra 2 e 200. Se non si specifica un valore, Lambda sceglie un valore predefinito pari a 2.
   - Per Maximum event poller, inserisci un valore compreso tra 2 e 2.000. Questo valore deve essere maggiore o uguale al valore specificato in Numero minimo di poller di eventi. Se non si specifica un valore, Lambda assegna il valore predefinito 200.
6. Scegli Save (Salva).

È possibile configurare la modalità di provisioning a livello di codice utilizzando l'`ProvisionedPollerConfig`oggetto in. `EventSourceMappingConfiguration` Ad esempio, il seguente comando `UpdateEventSourceMapping` CLI configura un `MinimumPollers` valore di 5 e un `MaximumPollers` valore di 100.

```
aws lambda update-event-source-mapping \
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
    --provisioned-poller-config '{"MinimumPollers": 5, "MaximumPollers": 100}'
```

Dopo aver configurato la modalità provisioning, puoi osservare l'utilizzo dei poller di eventi per il tuo carico di lavoro monitorando il parametro `ProvisionedPollers`. Per ulteriori informazioni, consulta Metriche di mappatura della fonte degli eventi.

Per disabilitare la modalità provisioning e tornare alla modalità predefinita (su richiesta), puoi utilizzare il seguente comando CLI`UpdateEventSourceMapping`:

```
aws lambda update-event-source-mapping \
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
```

```
    --provisioned-poller-config '{}'
```

> **Note**
>
> La modalità provisioned non può essere utilizzata insieme all'impostazione della massima concorrenza. Quando si utilizza la modalità provisioned, è possibile controllare la massima concorrenza mediante il numero massimo di sondaggi di eventi.

Per ulteriori informazioni sulla configurazione della modalità provisioned, vedere. the section called “Creare lo strumento di mappatura”

Esempio di evento con messaggio di coda standard

Example Evento messaggio Amazon SQS (coda standard)


```
{
    "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"
        }
    ]
}
```

Per impostazione predefinita, Lambda eseguirà il polling di un massimo di 10 messaggi contemporaneamente nella coda e invierà il batch alla funzione. Per evitare di richiamare la funzione con un piccolo numero di record, è possibile configurare l'origine eventi per memorizzare nel buffer i record per un massimo di 5 minuti definendo un periodo di batch. Prima di richiamare la funzione, Lambda continua a eseguire il polling dei messaggi dalla coda standard fino alla scadenza del periodo di batch, al raggiungimento della quota della dimensione del payload di richiesta o al raggiungimento della dimensione massima per la configurazione di un batch.

Se stai utilizzando un periodo di batch e la tua coda SQS contiene un traffico molto basso, Lambda potrebbe attendere fino a 20 secondi prima di richiamare la tua funzione. Lo stesso vale anche se imposti un periodo di batch inferiore a 20 secondi.

> **Note**
>
> In Java, potresti riscontrare errori di puntatore null durante la deserializzazione di JSON. Ciò potrebbe essere dovuto al modo in cui "Records" e "eventSourceARN" vengono convertiti dal mappatore di oggetti JSON.

Esempio di evento di messaggio di coda FIFO

Per le code FIFO, i record contengono attributi aggiuntivi correlati alla deduplicazione e al sequenziamento.

Example Evento messaggio Amazon SQS (coda 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"
        }
    ]
}
```

Creazione e configurazione di uno strumento di mappatura dell'origine degli eventi Amazon SQS

Per elaborare i messaggi Amazon SQS con Lambda, configura la coda con le impostazioni appropriate, quindi crea uno strumento di mappatura dell'origine degli eventi Lambda.

Argomenti

- Configurazione di una coda da utilizzare con Lambda
- Impostazione delle autorizzazioni del ruolo di esecuzione Lambda
- Creazione di uno strumento di mappatura dell'origine degli eventi SQS

Configurazione di una coda da utilizzare con Lambda

Se non disponi già di una coda Amazon SQS esistente, creane una da utilizzare come origine eventi per la funzione Lambda. La funzione Lambda e la coda Amazon SQS devono trovarsi nella Regione AWS stessa posizione, anche se possono trovarsi in posizioni diverse. Account AWS

Per concedere alla funzione il tempo necessario per elaborare ogni batch di record, imposta il timeout di visibilità della coda di origine su un valore pari ad almeno sei volte il timeout di configurazione per la funzione. Il tempo aggiuntivo consente a Lambda di riprovare se l'esecuzione della funzione viene limitata durante l'elaborazione di un batch precedente.

> **Note**
>
> Il timeout della funzione deve essere inferiore o uguale al timeout di visibilità della coda. Lambda convalida questo requisito quando si crea o si aggiorna una mappatura dell'origine degli eventi e restituisce un errore se il timeout della funzione supera il timeout di visibilità della coda.

Per impostazione predefinita, se Lambda rileva un errore in qualsiasi momento durante l'elaborazione di un batch, tutti i messaggi in quel batch ritorneranno nella coda. Dopo il timeout di visibilità, i messaggi diventano nuovamente visibili a Lambda. È possibile configurare lo strumento di mappatura dell'origine degli eventi in modo da utilizzare risposte batch parziali per restituire alla coda solo i messaggi non riusciti. Se la funzione dovesse non riuscire più volte a elaborare un messaggio, Amazon SQS può inviarlo a una coda DLQ. Ti consigliamo di impostare `maxReceiveCount` sulla policy redrive della coda di origine su un valore pari almeno a 5. Ciò offre a Lambda alcune possibilità di riprovare prima di inviare i messaggi non riusciti direttamente alla coda DLQ.

Impostazione delle autorizzazioni del ruolo di esecuzione Lambda

La policy AWSLambdaSQSQueueExecutionRole AWS gestita include le autorizzazioni di cui Lambda ha bisogno per leggere dalla coda Amazon SQS. Puoi aggiungere questa policy gestita al ruolo di esecuzione della tua funzione.

Facoltativamente, se utilizzi una coda crittografata, devi anche aggiungere la seguente autorizzazione al tuo ruolo di esecuzione:

- kms:Decrypt

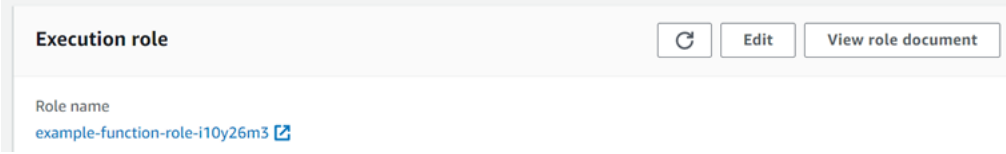
Creazione di uno strumento di mappatura dell'origine degli eventi SQS

Creare una mappatura dell'origine eventi per indicare a Lambda di inviare le voci dalla coda a una funzione Lambda. È possibile creare più mappature delle origini eventi per elaborare elementi da più code con una singola funzione. Quando Lambda richiama la funzione di destinazione, l'evento può contenere più voci, fino a una dimensione batch massima configurabile.

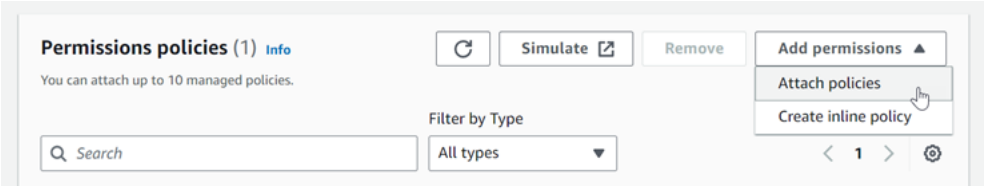
Per configurare la tua funzione per la lettura da Amazon SQS, collega la policy AWSLambdaSQSQueueExecutionRole AWS gestita al tuo ruolo di esecuzione. Quindi, crea uno strumento di mappatura dell'origine degli eventi SQS dalla console utilizzando i seguenti passaggi.

Per aggiungere le autorizzazioni e creare un trigger

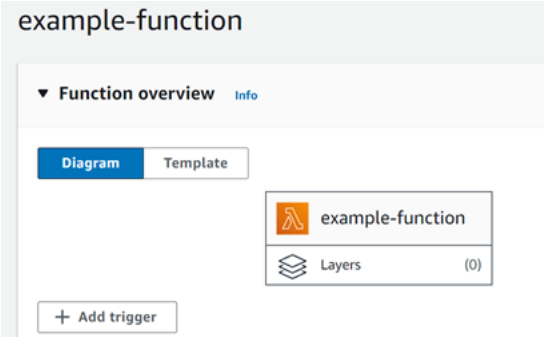
1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Quindi, seleziona la scheda Configuration (Configurazione) e poi Permissions (Autorizzazioni).
4. In Nome del ruolo, scegli il link al tuo ruolo di esecuzione. Questo ruolo si apre nella console IAM.



5. Seleziona Aggiungi autorizzazioni, quindi seleziona Collega policy.



6. Inserisci `AWSLambdaSQSQueueExecutionRole` nel campo di ricerca. Aggiungi questa policy al tuo ruolo di esecuzione. Si tratta di una policy AWS gestita che contiene le autorizzazioni di cui la funzione ha bisogno per leggere da una coda Amazon SQS. Per ulteriori informazioni su questa politica, consulta il AWS Managed Policy AWSLambdaSQSQueueExecutionRoleReference.
7. Torna alla funzione nella console Lambda. In Panoramica delle funzioni, scegliere Aggiungi trigger.



8. Scegliere un tipo di trigger.
9. Configurare le opzioni richieste, quindi scegliere Add (Aggiungi).

Lambda supporta le seguenti opzioni di configurazione per le origini eventi di Amazon SQS:

Coda SQS

La coda Amazon SQS da cui leggere i record. La funzione Lambda e la coda Amazon SQS devono trovarsi nella Regione AWS stessa posizione, anche se possono trovarsi in posizioni diverse. Account AWS

Abilita trigger

Lo stato dello strumento di mappatura dell'origine degli eventi. L'opzione Enable trigger (Abilita trigger) è selezionata per impostazione predefinita.

Dimensione batch

Il numero massimo di record da inviare alla funzione in ogni batch. Per una coda standard il numero di registri può arrivare fino a 10.000. Per una coda FIFO il massimo è 10. Per un batch di dimensioni superiori a 10, è inoltre necessario la finestra batch (`MaximumBatchingWindowInSeconds`) su almeno 1 secondo.

Configura il timeout della funzione per consentire il tempo necessario all'elaborazione di un intero batch di elementi. Se gli elementi richiedono tempi di elaborazione più lunghi, scegli un batch di dimensioni più piccole. Un batch di grandi dimensioni può migliorare l'efficienza per i carichi di lavoro molto veloci o con costi di gestione molto elevati. Se si configura la concorrenza riservata sulla funzione, impostare un minimo di cinque esecuzioni simultanee per ridurre le probabilità di errori di limitazione (della larghezza di banda della rete) quando Lambda richiama la funzione.

Lambda passa tutti i registri del batch alla funzione in una singola chiamata, purché la dimensione totale degli eventi non superi la quota della dimensione del payload di invocazione per una

invocazione sincrona (6 MB). Per ogni registro, vengono generati metadati sia da Lambda che da Amazon SQS. Questi metadati aggiuntivi vengono conteggiati per la dimensione totale del payload e possono far sì che il numero totale di registri inviati in un batch sia inferiore alla dimensione del batch configurato. I campi di metadati inviati da Amazon SQS possono essere di lunghezza variabile. Per ulteriori informazioni sui campi di metadati di Amazon SQS, consulta la documentazione sul funzionamento delle ReceiveMessageAPI nell'Amazon Simple Queue Service API Reference.

Finestra batch

Il tempo massimo in secondi per la raccolta dei record prima di richiamare la funzione. Questo parametro si applica solo alle code standard.

Se si utilizza un periodo batch superiore a 0 secondi, è necessario tenere conto dell'aumento del tempo di elaborazione nel timeout di visibilità della coda. Si consiglia di impostare il timeout di visibilità della coda ad un tempo sei volte maggiore rispetto al timeout della funzione, più il valore di `MaximumBatchingWindowInSeconds`. Ciò consente alla funzione Lambda di elaborare ogni batch di eventi e riprovare in caso di errore di throttling.

Quando i messaggi diventano disponibili, Lambda avvia l'elaborazione dei messaggi in batch. Lambda inizia a elaborare cinque batch alla volta con cinque chiamate simultanee della funzione. Se i messaggi sono ancora disponibili, Lambda aggiunge fino a 300 richiami simultanei della funzione al minuto, fino a un massimo di 1.250 richiami simultanei. Quando si utilizza la modalità provisioned, ogni event poller può gestire fino all'1% MB/s del throughput, fino a 10 richiami simultanei o fino a 10 chiamate API di polling di Amazon SQS al secondo. Lambda ridimensiona il numero di event poller tra il minimo e il massimo configurati, aggiungendo rapidamente fino a 1.000 richiami simultanei al minuto per fornire un'elaborazione a bassa latenza degli eventi Amazon SQS. Puoi controllare la scalabilità e la concorrenza tramite queste impostazioni minime e massime del poller degli eventi. Per ulteriori informazioni sul ridimensionamento e sulla concorrenza delle funzioni, consulta. Informazioni sulla scalabilità della funzione Lambda

Per elaborare più messaggi, puoi ottimizzare la funzione Lambda per un maggiore throughput. Per ulteriori informazioni, consulta Comprendere la AWS Lambda scalabilità con le code standard di Amazon SQS.

Criteri di filtro

Aggiungi i criteri di filtro per controllare gli eventi che Lambda invia alla funzione per l'elaborazione. Per ulteriori informazioni, consulta Controllare gli eventi che Lambda invia alla funzione.

Simultaneità massima

Il numero massimo di funzioni simultanee che l'origine eventi può richiamare. Non può essere utilizzato con la modalità Provisioned Mode abilitata. Per ulteriori informazioni, consulta Configurazione della simultaneità massima per le origini eventi di Amazon SQS.

Modalità assegnata

Se abilitata, alloca risorse di polling dedicate per la mappatura delle sorgenti degli eventi. È possibile configurare il numero minimo (2-200) e massimo (2-2000) di sondaggi di eventi. Ogni event poller può gestire fino all'1% MB/sec del throughput, fino a 10 chiamate simultanee o fino a 10 chiamate API di polling di Amazon SQS al secondo.

> **Note**
>
> Nota: non è possibile utilizzare contemporaneamente Provisioned Mode e Maximum Concurrency. Quando la modalità Provisioned Mode è abilitata, utilizza l'impostazione del numero massimo di poller per controllare la concorrenza.

Configurazione del comportamento di dimensionamento per gli strumenti di mappatura dell'origine degli eventi

Puoi controllare il comportamento di scalabilità delle mappature delle sorgenti degli eventi di Amazon SQS tramite impostazioni di massima concorrenza o abilitando la modalità provisioning. Si tratta di opzioni che si escludono a vicenda.

Per impostazione predefinita, Lambda ridimensiona automaticamente i poller di eventi in base al volume dei messaggi. Quando abiliti la modalità provisioned, allochi un numero minimo e massimo di risorse di polling dedicate che rimangono pronte a gestire i modelli di traffico previsti. Ciò consente di ottimizzare le prestazioni della mappatura delle sorgenti degli eventi in due modi:

- Modalità standard (impostazione predefinita): Lambda gestisce automaticamente la scalabilità, a partire da un numero limitato di poller e aumentando o diminuendo in base al carico di lavoro.
- Modalità provisioned: è possibile configurare risorse di polling dedicate con limiti minimi e massimi, che consentono una scalabilità 3 volte più veloce e una capacità di elaborazione fino a 16 volte superiore.

Per le code standard, Lambda utilizza il long polling per eseguire il polling di una coda fino a quando questa non diventa attiva. Quando sono disponibili dei messaggi, Lambda inizia a elaborare cinque batch alla volta con cinque chiamate simultanee della funzione. Se i messaggi sono ancora disponibili, Lambda aumenta il numero di processi che leggono i batch fino a 300 chiamate simultanee in più al minuto. Il numero massimo di richiami che una mappatura dell'origine degli eventi può elaborare contemporaneamente è 1.250. Quando il traffico è basso, Lambda ridimensiona l'elaborazione a cinque chiamate simultanee e può ottimizzarla fino a un minimo di 2 richiami simultanei per ridurre le chiamate Amazon SQS e i costi corrispondenti. Tuttavia, questa ottimizzazione non è disponibile quando si abilita l'impostazione della massima concorrenza.

Per le code FIFO, Lambda invia messaggi alla funzione nell'ordine in cui li riceve. Quando invii un messaggio a una coda FIFO, è necessario specificare un ID gruppo di messaggi. Amazon SQS garantisce che i messaggi dello stesso gruppo vengano consegnati a Lambda in ordine. Quando Lambda legge i messaggi in batch, ogni batch può contenere messaggi provenienti da più di un gruppo di messaggi, ma l'ordine dei messaggi viene mantenuto. Se la funzione restituisce un errore, questa esegue nuovamente tutti i tentativi necessari sui messaggi interessati, prima che Lambda riceva ulteriori messaggi dallo stesso gruppo.

Quando si utilizza la modalità provisioned, ogni event poller può gestire fino all'1% MB/sec del throughput, fino a 10 richiami simultanei o fino a 10 chiamate API di polling di Amazon SQS al secondo. Lambda ridimensiona il numero di event poller tra il minimo e il massimo configurati, aggiungendo rapidamente fino a 1.000 simultanee al minuto per fornire un'elaborazione coerente e a bassa latenza degli eventi Amazon SQS. L'utilizzo della modalità provisioning comporta costi aggiuntivi. Per i prezzi dettagliati, consulta la pagina dei prezzi.AWS Lambda Ogni event poller utilizza lunghi sondaggi sulla coda SQS con un massimo di 10 sondaggi al secondo, il che comporta il costo delle richieste API SQS. Per ulteriori informazioni, consulta i prezzi di Amazon SQS. Puoi controllare la scalabilità e la concorrenza tramite queste impostazioni minime e massime del poller di eventi, anziché utilizzare l'impostazione di concorrenza massima, poiché queste opzioni non possono essere utilizzate insieme.

> **Note**
>
> Non è possibile utilizzare contemporaneamente l'impostazione di massima concorrenza e la modalità di provisioning. Quando la modalità provisioned è abilitata, puoi controllare la scalabilità e la concorrenza della mappatura delle sorgenti di eventi di Amazon SQS attraverso il numero minimo e massimo di event poller.

Configurazione della simultaneità massima per le origini eventi di Amazon SQS

È possibile utilizzare l'impostazione di simultaneità massima per controllare il comportamento di dimensionamento delle origini eventi SQS. Tieni presente che non è possibile utilizzare la massima concorrenza con la modalità provisioning abilitata. L'impostazione della simultaneità massima limita il numero di istanze simultanee della funzione che l'origine dell'evento Amazon SQS può richiamare. La simultaneità massima è un'impostazione a livello di origine dell'evento. Se disponi di più origini degli eventi Amazon SQS mappate a una funzione, ogni origine di evento può avere un'impostazione di simultaneità massima separata. È possibile utilizzare la simultaneità massima per evitare che una coda utilizzi tutta la simultaneità riservata della funzione o il resto della quota di simultaneità dell'account. Non è previsto alcun addebito per la configurazione della simultaneità massima su un'origine di eventi Amazon SQS.

È importante sottolineare che la simultaneità massima e la simultaneità riservata sono due impostazioni indipendenti. Non è possibile impostare la simultaneità massima su un valore maggiore della simultaneità riservata della funzione. Dopo aver configurato la simultaneità massima, assicurati di non ridurre la simultaneità riservata della funzione a un valore inferiore alla simultaneità massima totale per tutte le origini di eventi Amazon SQS sulla funzione. Altrimenti, Lambda potrebbe limitare i messaggi.

Quando la quota di simultaneità del tuo account è impostata sul valore predefinito di 1.000, uno strumento di mappatura dell'origine degli eventi Amazon SQS può scalare per richiamare istanze di funzioni fino a questo valore, a meno che non si specifichi una simultaneità massima.

Se ricevi un aumento della quota di simultaneità predefinita del tuo account, Lambda potrebbe non essere in grado di richiamare istanze di funzioni simultanee fino alla tua nuova quota. Per impostazione predefinita, Lambda può scalare per richiamare fino a 1.250 istanze di funzioni simultanee per uno strumento di mappatura dell'origine degli eventi Amazon SQS. Se questo non è sufficiente per il tuo caso d'uso, contatta l' AWS assistenza per discutere di un aumento della contemporaneità di mappatura dei sorgenti degli eventi Amazon SQS del tuo account.

> **Note**
>
> Per le code FIFO, le chiamate simultanee sono limitate dal numero di gruppi di messaggi IDs () `messageGroupId` o dall'impostazione di concorrenza massima, a seconda di quale sia inferiore. Ad esempio, se avete sei gruppi di messaggi IDs e la concorrenza massima è impostata su 10, la funzione può avere un massimo di sei chiamate simultanee.

Puoi configurare la simultaneità massima sugli strumenti di mappatura dell'origine degli eventi Amazon SQS nuovi ed esistenti.

Configurazione della simultaneità massima tramite la console Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. In Function overview (Panoramica delle funzioni), scegli SQS. Viene aperta la scheda Configuration (Configurazione).
4. Seleziona il trigger Amazon SQS e scegli Edit (Modifica).
5. In Maximum concurrency (Simultaneità massima), inserisci un numero compreso tra 2 e 1.000. Per disattivare la simultaneità massima, lascia la casella vuota.
6. Scegli Save (Salva).

Configura la massima concorrenza utilizzando () AWS Command Line InterfaceAWS CLI

Utilizza il comando update-event-source-mapping con l'opzione `--scaling-config`. Esempio:

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --scaling-config '{"MaximumConcurrency":5}'
```

Per disattivare la simultaneità massima, inserisci un valore vuoto per `--scaling-config`:

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --scaling-config "{}"
```

Configurazione della simultaneità massima tramite l'API Lambda

Usa l'UpdateEventSourceMappingazione CreateEventSourceMappingo con un ScalingConfigoggetto.

Gestione degli errori per un'origine eventi SQS in Lambda

Per gestire gli errori relativi a un'origine eventi SQS, Lambda utilizza automaticamente una strategia di riprova con una strategia di backoff. Puoi anche personalizzare il comportamento di gestione degli errori configurando lo strumento di mappatura dell'origine degli eventi SQS per restituire risposte parziali in batch.

Strategia di backoff per le chiamate non riuscite

Quando una chiamata non riesce, Lambda riprovare la chiamata mentre implementa una strategia di backoff. La strategia di backoff varia leggermente a seconda che Lambda abbia riscontrato l'errore a causa di un errore nel codice della funzione o a causa di una limitazione.

- Se il codice della funzione ha causato l'errore, Lambda interromperà l'elaborazione e riproverà l'invocazione. Allo stesso tempo, Lambda si interrompe gradualmente, riducendo la quantità di simultaneità allocata allo strumento di mappatura dell'origine degli eventi di Amazon SQS. Una volta scaduto il timeout di visibilità della coda, il messaggio riapparirà nuovamente nella coda.
- Se la chiamata non riesce a causa della limitazione, Lambda interrompe gradualmente i nuovi tentativi riducendo la quantità di simultaneità allocata allo strumento di mappatura dell'origine degli eventi di Amazon SQS. Lambda continua a riprovare il messaggio fino a quando il timestamp del messaggio non supera il timeout di visibilità della coda e a quel punto Lambda elimina il messaggio.

Implementazione di risposte batch parziali

Per impostazione predefinita, se la funzione rileva un errore durante l'elaborazione di un batch, tutti i messaggi in quel batch diventano nuovamente visibili nella coda, inclusi i messaggi che Lambda ha elaborato correttamente. Di conseguenza, la tua funzione potrebbe elaborare lo stesso messaggio più volte.

Per evitare di rielaborare tutti i messaggi correttamente elaborati in un batch con errori, puoi configurare lo strumento di mappatura dell'origine degli eventi in modo da rendere nuovamente visibili solo i messaggi con errori. Questa operazione è nota come risposta batch parziale. Per attivare le risposte parziali in batch, specifica l'FunctionResponseTypesazione da eseguire durante `ReportBatchItemFailures` la configurazione della mappatura delle sorgenti degli eventi. Ciò consente alla funzione di restituire un completamento parziale, riducendo così il numero di tentativi non necessari sui registri.

> **Note**
>
> L'utilità Batch Processor di Powertools AWS Lambda gestisce automaticamente tutta la logica di risposta batch parziale. Questa utilità semplifica l'implementazione dei modelli di elaborazione in batch e riduce il codice personalizzato necessario per gestire correttamente gli errori degli articoli in batch. È disponibile per Python, Java, Typescript e.NET.

Se `ReportBatchItemFailures` è attivata, Lambda non riduce il polling dei messaggi quando le invocazioni delle funzioni hanno esito negativo. Se prevedi che alcuni messaggi falliscano e non desideri che tali errori influiscano sulla velocità di elaborazione dei messaggi, utilizza `ReportBatchItemFailures`.

> **Note**
>
> Quando utilizzi le risposte batch, tieni presente quanto segue:
>
> - Se la funzione restituisce un'eccezione, l'intero batch viene considerato completamente non riuscito.
> - Se si utilizza questa caratteristica con una coda FIFO, la funzione dovrebbe interrompere l'elaborazione dei messaggi dopo il primo errore e restituire tutti i messaggi di errore e non elaborati in `batchItemFailures`. Ciò aiuta a preservare l'ordine dei messaggi nella coda.

Attivazione di report batch parziali

1. Rivedi le best practice per l'implementazione di risposte batch parziali.
2. Per attivare `ReportBatchItemFailures` per la tua funzione, esegui il comando riportato di seguito. Per recuperare l'UUID della mappatura della sorgente degli eventi, esegui il comando. list-event-source-mappings AWS CLI

   ```
   aws lambda update-event-source-mapping \
   --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
   --function-response-types "ReportBatchItemFailures"
   ```

3. Aggiorna il codice della funzione per rilevare tutte le eccezioni e restituire i messaggi con errori in una risposta `batchItemFailures` in JSON. La `batchItemFailures` risposta deve includere un elenco di messaggi IDs, sotto forma di valori JSON. `itemIdentifier`

   Ad esempio, supponiamo di avere un batch di cinque messaggi, IDs `id1`, `id2` `id3``id4`, e. `id5` La tua funzione elabora correttamente `id1`, `id3`, e `id5`. Per rendere i messaggi `id2` e `id4` di nuovo visibili nella coda, la funzione dovrebbe restituire la seguente risposta:

   ```
   {
     "batchItemFailures": [
           {
               "itemIdentifier": "id2"
   ```

```
        },
        {
            "itemIdentifier": "id4"
        }
    ]
}
```

Di seguito sono riportati alcuni esempi di codice di funzione che restituiscono l'elenco dei messaggi non riusciti IDs nel batch:

.NET

SDK per .NET

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione degli errori degli elementi batch SQS utilizzando 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}`);
}
```

Segnalazione degli errori degli elementi batch SQS utilizzando 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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
}
```

Se gli eventi non riusciti non tornano in coda, vedi Come posso risolvere i problemi della funzione Lambda SQS? ReportBatchItemFailures nel Knowledge Center. AWS

Condizioni di successo e di errore

Lambda considera un batch completamente riuscito se la funzione restituisce uno dei seguenti elementi:

- Una `batchItemFailures` lista vuota
- Un `batchItemFailures` elenco nullo
- Un vuoto `EventResponse`
- Un valore nullo `EventResponse`

Lambda considera un batch completamente non riuscito se la funzione restituisce uno dei seguenti elementi:

- Risposta JSON non valida
- Una stringa vuota `itemIdentifier`
- Un valore nullo `itemIdentifier`
- Un `itemIdentifier` con un nome chiave errato
- Un `itemIdentifier` valore di con un ID messaggio inesistente

CloudWatch metriche

Per determinare se la tua funzione riporta correttamente gli errori degli articoli in batch, puoi monitorare i parametri `NumberOfMessagesDeleted` e `ApproximateAgeOfOldestMessage` Amazon SQS in Amazon. CloudWatch

- `NumberOfMessagesDeleted` tiene traccia del numero di messaggi rimossi dalla coda. Se questo scende a 0, significa che la risposta alla funzione non sta restituendo correttamente i messaggi non riusciti.

- `ApproximateAgeOfOldestMessage` tiene traccia del tempo di permanenza del messaggio meno recente all'interno della coda. Un incremento considerevole di questo parametro può indicare che la funzione non sta restituendo correttamente i messaggi non riusciti.

Utilizzo di Powertools per il processore batch AWS Lambda

L'utilità per il processore batch di Powertools gestisce AWS Lambda automaticamente la logica di risposta parziale in batch, riducendo la complessità dell'implementazione della segnalazione degli errori dei batch. Ecco alcuni esempi di utilizzo del processore batch:

Python

> **Note**
>
> Per esempi completi e istruzioni di configurazione, consultate la documentazione del processore batch.

Elaborazione di messaggi Amazon SQS con processore 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**
>
> Per esempi completi e istruzioni di configurazione, consulta la documentazione del processore batch.

Elaborazione di messaggi Amazon SQS con processore 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,
    });
};
```

Parametri Lambda per gli strumenti di mappatura dell'origine degli eventi di Amazon SQS

Tutti i tipi di sorgenti di eventi Lambda condividono le stesse operazioni CreateEventSourceMappinge quelle dell'UpdateEventSourceMappingAPI. Tuttavia, solo alcuni dei parametri si applicano ad Amazon SQS.

Parametro	Obbligatorio	Predefinito	Note
BatchSize	N	10	Per le code standard il massimo è 10.000. Per le code FIFO il massimo è 10.
Abilitato	N	true	nessuno
EventSourceArn	Y	N/D	L'ARN del flusso dei dati o di un consumatore di flusso
FunctionName	Y	N/D	nessuno
FilterCriteria	N	N/D	Controllare gli eventi che Lambda invia alla funzione
FunctionResponseTypes	N	N/D	Per consentire alla funzione di segnalare errori specifici in un batch, includi il valore `ReportBatchItemFailures` in `FunctionResponseTypes`. Per ulteriori informazioni, consulta Implementazione di risposte batch parziali.
MaximumBatchingWindowInSeconds	N	0	La creazione di finestre di batch non è supportata per le code FIFO

Parametro	Obbligatorio	Predefinito	Note
ProvisionedPollerConfig	N	N/D	Configura il numero minimo (2-200) e massimo (2-2000) di poller di eventi dedicati per la mappatura delle sorgenti degli eventi SQS. Ogni poller può gestire fino all'1% del throughput e 10 richiami simultanei MB/sec .
ScalingConfig	N	N/D	Configurazione della simultaneità massima per le origini eventi di Amazon SQS

Utilizzo del filtro eventi con un'origine eventi Amazon SQS

Puoi utilizzare il filtraggio degli eventi per controllare quali record di un flusso o di una coda Lambda invia alla funzione. Per informazioni generali sul funzionamento del filtraggio eventi, consulta the section called “Filtro eventi”.

In questa sezione viene descritto il filtraggio degli eventi per le origini di eventi Amazon MQ.

> **Note**
>
> Le mappature delle sorgenti degli eventi di Amazon SQS supportano solo il filtraggio sulla chiave. `body`

Argomenti

- Nozioni di base sul filtraggio degli eventi di Amazon SQS

Nozioni di base sul filtraggio degli eventi di Amazon SQS

Supponiamo che la coda Amazon SQS contenga messaggi nel formato JSON seguente.

```
{
    "RecordNumber": 1234,
    "TimeStamp": "yyyy-mm-ddThh:mm:ss",
    "RequestCode": "AAAA"
}
```

Un record di esempio per questa coda sarebbe il seguente.

```
{
    "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"
}
```

Per filtrare in base al contenuto dei messaggi Amazon SQS, utilizza la chiave `body` nel record dei messaggi Amazon SQS. Supponiamo di voler elaborare solo i record nei quali il `RequestCode` del messaggio Amazon SQS è "BBBB". L'oggetto `FilterCriteria` dovrebbe avere la struttura seguente.

```
{
    "Filters": [
        {
            "Pattern": "{ \"body\" : { \"RequestCode\" : [ \"BBBB\" ] } }"
        }
    ]
```

```
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
    "body": {
        "RequestCode": [ "BBBB" ]
        }
}
```

Puoi aggiungere il filtro utilizzando la console, la AWS CLI o un modello AWS SAM.

Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in Collegamento dei criteri di filtro a una mappatura dell'origine evento (console) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "body" : { "RequestCode" : [ "BBBB" ] } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtraggio utilizzando l'AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
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\" ] } }"}]}'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RequestCode\" :
 [ \"BBBB\" ] } }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML dell'origine degli eventi.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "body" : { "RequestCode" : [ "BBBB" ] } }'
```

Supponiamo che tu voglia che la tua funzione elabori solo i record in cui `RecordNumber` è maggiore di 9999. L'oggetto `FilterCriteria` dovrebbe avere la struttura seguente.

```
{
    "Filters": [
        {
            "Pattern": "{ \"body\" : { \"RecordNumber\" : [ { \"numeric\": [ \">\",
 9999 ] } ] } }"
        }
    ]
}
```

Per una maggiore chiarezza, ecco il valore del `Pattern` del filtro espanso in JSON semplice.

```
{
    "body": {
        "RecordNumber": [
            {
                "numeric": [ ">", 9999 ]
            }
        ]
    }
}
```

Puoi aggiungere il filtro utilizzando la console, la AWS CLI o un modello AWS SAM.

Console

Per aggiungere questo filtro utilizzando la console, segui le istruzioni riportate in Collegamento dei criteri di filtro a una mappatura dell'origine evento (console) e inserisci la seguente stringa per i criteri di filtraggio.

```
{ "body" : { "RecordNumber" : [ { "numeric": [ ">", 9999 ] } ] } }
```

AWS CLI

Per creare una nuova mappatura dell'origine degli eventi con questi criteri di filtraggio utilizzando l'AWS Command Line Interface (AWS CLI), esegui il comando seguente.

```
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 ] } ] } }"}]}'
```

Per aggiungere questi criteri di filtraggio a una mappatura dell'origine degli eventi esistente, esegui il comando seguente.

```
aws lambda update-event-source-mapping \
    --uuid "a1b2c3d4-5678-90ab-cdef-11111EXAMPLE" \
    --filter-criteria '{"Filters": [{"Pattern": "{ \"body\" : { \"RecordNumber\" :
 [ { \"numeric\": [ \">\", 9999 ] } ] } }"}]}'
```

AWS SAM

Per aggiungere questo filtro utilizzando AWS SAM, aggiungi il seguente frammento al modello YAML dell'origine degli eventi.

```
FilterCriteria:
  Filters:
    - Pattern: '{ "body" : { "RecordNumber" : [ { "numeric": [ ">", 9999 ] } ] } }'
```

Per Amazon SQS, il corpo del messaggio può essere qualsiasi stringa. Tuttavia, questo può essere problematico se il `FilterCriteria` si aspetta che `body` sia in un formato JSON valido. Anche lo scenario inverso è vero: se il corpo del messaggio in arrivo è in formato JSON ma i criteri di filtraggio si aspettano che `body` sia una stringa semplice, allora si potrebbe riscontrare un comportamento indesiderato.

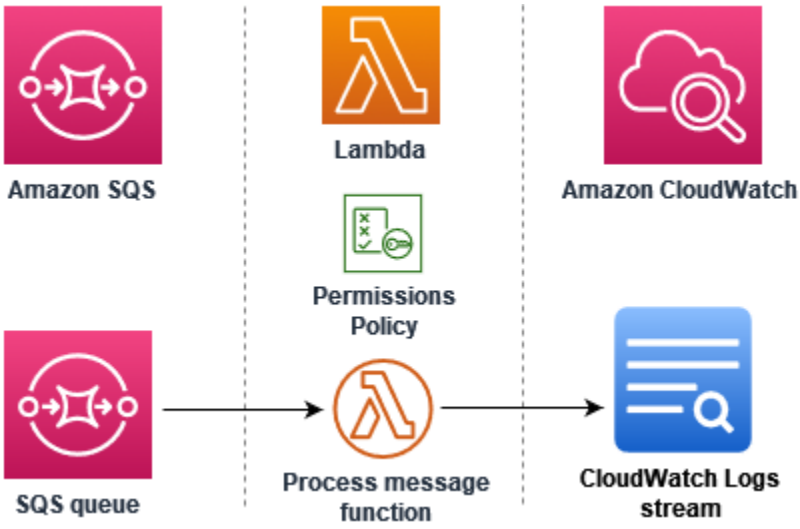
Per evitare questo problema, assicurati che il formato del corpo nei `FilterCriteria` corrisponda al formato previsto per il `body` nei messaggi ricevuti dalla coda. Prima di filtrare i messaggi, Lambda

valuta automaticamente il formato del corpo del messaggio in arrivo e il modello di filtraggio per `body`. In caso di mancata corrispondenza, Lambda rilascia il messaggio. La tabella seguente riepiloga questa valutazione:

Formato **body** messaggio in arrivo	Formato **body** modello di filtro	Operazione risultante
Stringa normale	Stringa normale	Filtri Lambda in base ai criteri di filtro.
Stringa normale	Nessun modello di filtro per le proprietà dei dati	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.
Stringa normale	JSON valido	Lambda rilascia il messaggio.
JSON valido	Stringa normale	Lambda rilascia il messaggio.
JSON valido	Nessun modello di filtro per le proprietà dei dati	Filtri Lambda (solo sulle altre proprietà dei metadati) in base ai criteri di filtro.
JSON valido	JSON valido	Filtri Lambda in base ai criteri di filtro.

Tutorial: Utilizzo di Lambda con Amazon SQS

In questo tutorial creerai una funzione Lambda che utilizza messaggi da una coda di Amazon Simple Queue Service (Amazon SQS). La funzione Lambda viene eseguita ogni volta che viene aggiunto un nuovo messaggio alla coda. La funzione scrive i messaggi in un flusso Amazon CloudWatch Logs. Il seguente diagramma illustra le risorse AWS utilizzate per completare il tutorial.



Per completare questo tutorial, completa le seguenti attività:

1. Crea una funzione Lambda che scrive messaggi nei registri. CloudWatch
2. Creare una coda Amazon SQS.
3. Crea una mappatura dell'origine degli eventi Lambda. La mappatura dell'origine degli eventi legge la coda di Amazon SQS e richiama la funzione Lambda quando viene aggiunto un nuovo messaggio.
4. Verifica la configurazione aggiungendo messaggi alla coda e monitorando i risultati in Logs. CloudWatch

Prerequisiti

Installa il AWS Command Line Interface

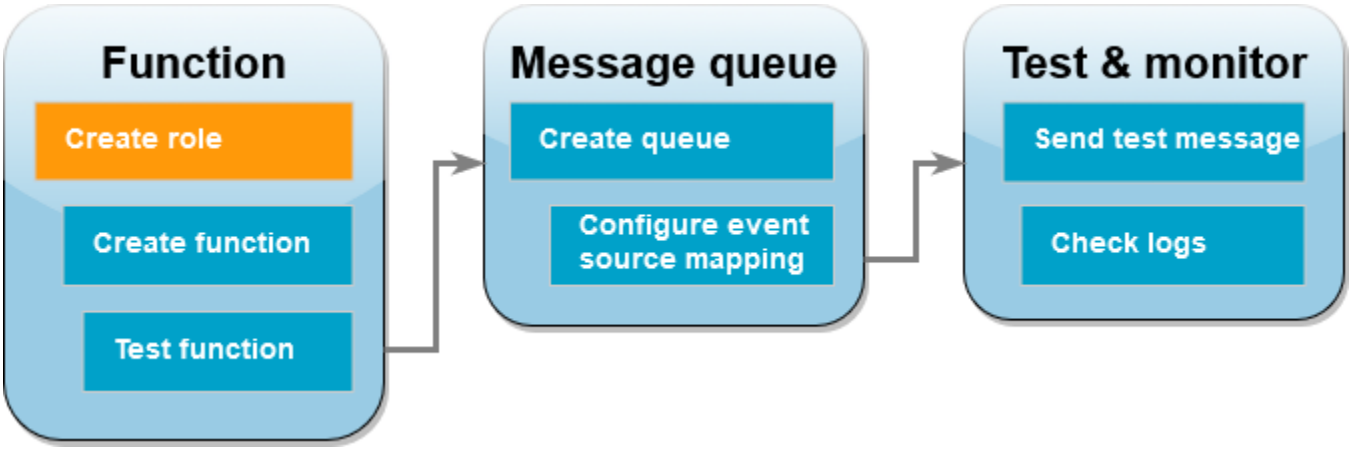
Se non l'hai ancora installato AWS Command Line Interface, segui i passaggi indicati in Installazione o aggiornamento della versione più recente di AWS CLI per installarlo.

Per eseguire i comandi nel tutorial, sono necessari un terminale a riga di comando o una shell (interprete di comandi). In Linux e macOS, utilizza la shell (interprete di comandi) e il gestore pacchetti preferiti.

> **Note**
>
> Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere

una versione integrata su Windows di Ubuntu e Bash, installa il sottosistema Windows per Linux.

Creazione del ruolo di esecuzione



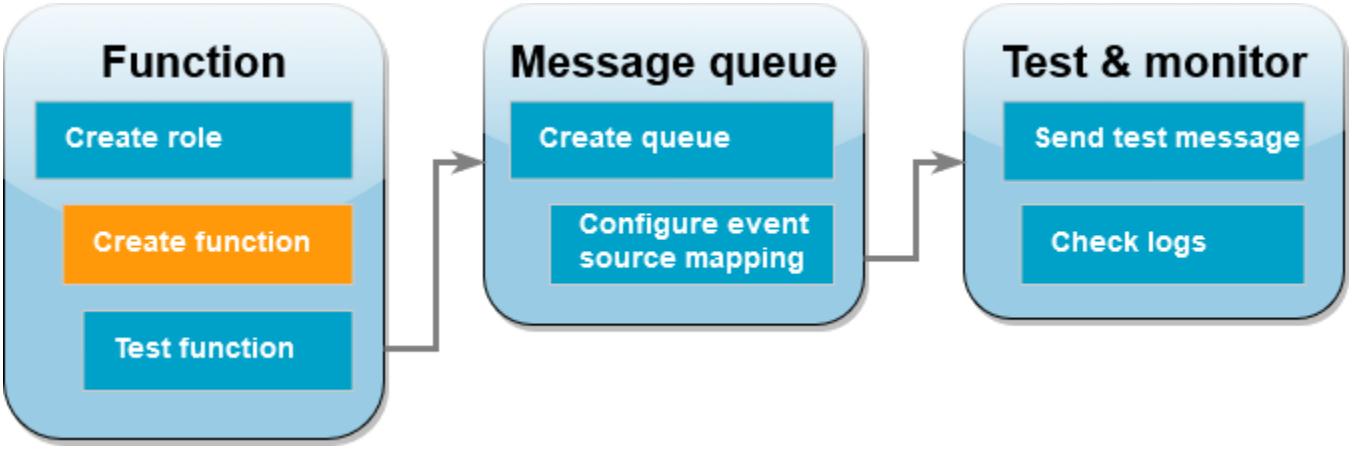
Un ruolo di esecuzione è un ruolo AWS Identity and Access Management (IAM) che concede a una funzione Lambda l'autorizzazione all' Servizi AWS accesso e alle risorse. Per consentire alla tua funzione di leggere articoli da Amazon SQS, allega la politica di AWSLambdaSQSQueueExecutionRoleautorizzazione.

Creazione di un ruolo di esecuzione e collegamento di una policy di autorizzazione Amazon SQS personalizzata

1. Aprire la pagina Roles (Ruoli) della console IAM.
2. Scegli Crea ruolo.
3. Per Tipo di entità attendibile, scegli Servizio AWS .
4. In Caso d'uso, scegli Lambda.
5. Scegli Next (Successivo).
6. Nella casella di ricerca Policy di autorizzazione, inserisci **`AWSLambdaSQSQueueExecutionRole`**.
7. Seleziona la AWSLambdaSQSQueueExecutionRolepolitica, quindi scegli Avanti.
8. In Dettagli del ruolo, per Nome del ruolo inserisci **`lambda-sqs-role`**, quindi scegli Crea ruolo.

Dopo la creazione del ruolo, prendi nota del valore del nome della risorsa Amazon (ARN) del ruolo di esecuzione. Ne avrai bisogno nelle fasi successive.

Creazione della funzione



Crea una funzione Lambda che elabora i messaggi Amazon SQS. Il codice della funzione registra il corpo del messaggio CloudWatch Amazon SQS in Logs.

Questo tutorial utilizza il runtime Node.js 24, ma abbiamo anche fornito codice di esempio in altri linguaggi di runtime. Per visualizzare il codice per il runtime che ti interessa, seleziona la scheda corrispondente nella casella seguente. Il JavaScript codice che utilizzerai in questo passaggio si trova nel primo esempio mostrato nella JavaScriptscheda.

.NET

SDK per .NET

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento SQS con JavaScript Lambda utilizzando.

```
// 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;
  }
}
```

Consumo di un evento SQS con TypeScript Lambda utilizzando.

```
// 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite 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
}
```

Creazione di una funzione Lambda in Node.js

1. Crea una directory per il progetto, quindi passa a quella directory.

```
mkdir sqs-tutorial
cd sqs-tutorial
```

2. Copia il JavaScript codice di esempio in un nuovo file denominato`index.js`.
3. Crea un pacchetto di implementazione utilizzando il seguente comando `zip`.

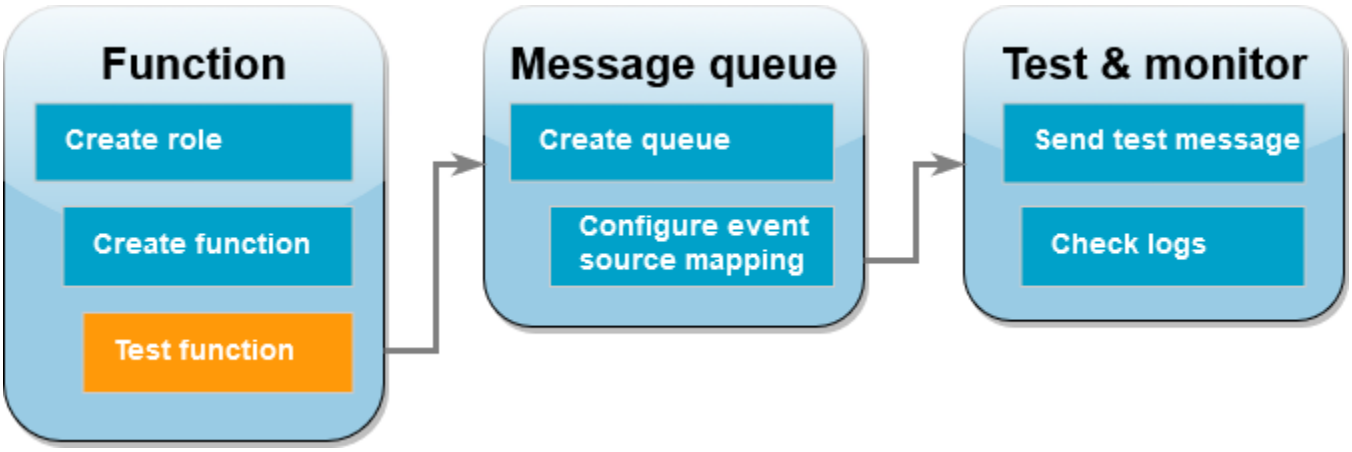
```
zip function.zip index.js
```

4. Crea una funzione Lambda utilizzando il comando create-function della AWS CLI . Per il `role` parametro, immettere l'ARN del ruolo di esecuzione creato in precedenza.

> **Note**
>
> La funzione Lambda e la coda Amazon SQS devono trovarsi nella stessa Regione 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
```

Test della funzione



Richiama la funzione Lambda manualmente utilizzando `invoke` AWS CLI il comando e un evento Amazon SQS di esempio.

Invocazione della funzione Lambda con un evento di esempio

1. Salva il seguente JSON come un file denominato `input.json`. Questo JSON simula un evento che Amazon SQS potrebbe inviare alla tua funzione Lambda, dove `"body"` contiene il messaggio effettivo dalla coda. In questo esempio, il messaggio è `"test"`.

   Example Evento Amazon SQS

   Questo è un evento di test: non è necessario modificare il messaggio o il numero di account.

   ```
   {
       "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. Esegui il seguente comando invoke. AWS CLI Questo comando restituisce CloudWatch i log nella risposta. Per ulteriori informazioni sul recupero di oggetti, consulta Accedi ai log con AWS CLI.

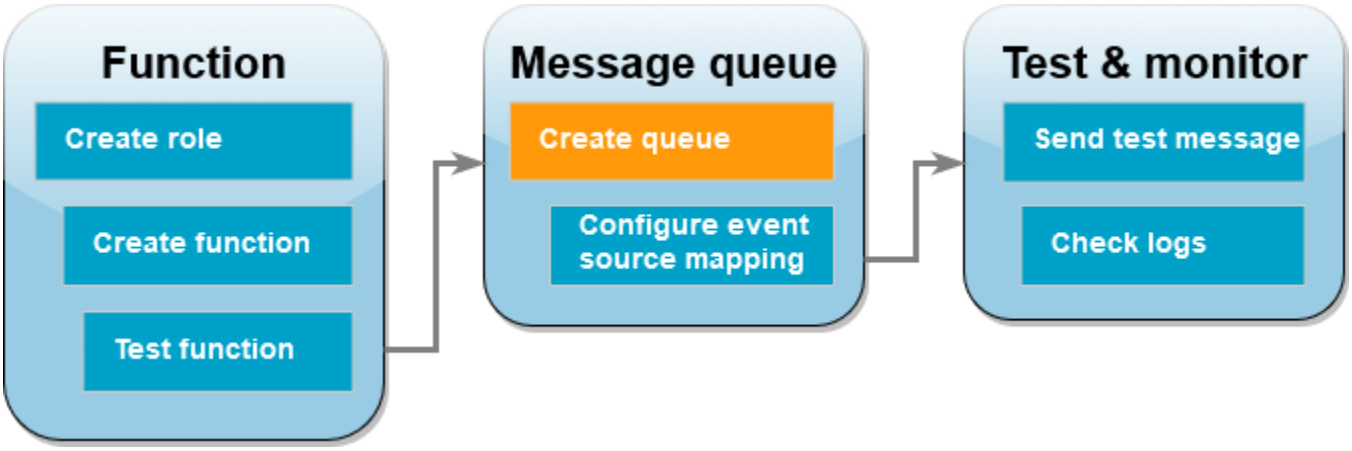
```
aws lambda invoke --function-name ProcessSQSRecord --payload file://input.json out
 --log-type Tail \
--query 'LogResult' --output text --cli-binary-format raw-in-base64-out | base64 --
decode
```

   L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

3. Individua il log `INFO` nella risposta. È qui che la funzione Lambda registra il corpo del messaggio. I log visualizzati dovrebbero essere di questo tipo:

```
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
```

Creazione di una coda Amazon SQS



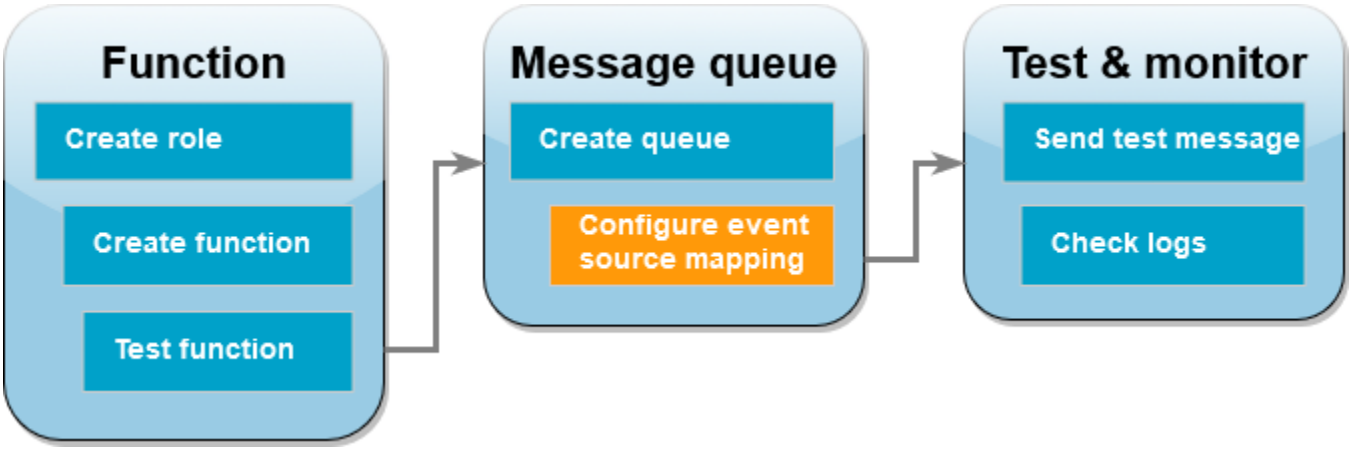
Creare una coda Amazon SQS che la funzione Lambda può utilizzare come origine eventi. La funzione Lambda e la coda Amazon SQS devono trovarsi nella stessa Regione AWS.

Come creare una coda

1. Apri la console Amazon SQS.
2. Scegliere Crea coda.
3. Inserisci un nome per la coda. Lascia tutte le altre proprietà sui valori predefiniti.
4. Scegliere Crea coda.

Dopo aver creato la coda, prendi nota del suo ARN. Questa operazione è necessaria nella fase successiva quando si associa la coda alla funzione Lambda.

Configurazione dell'origine eventi



Collega la coda Amazon SQS alla tua funzione Lambda creando una mappatura dell'origine degli eventi. La mappatura dell'origine degli eventi legge la coda di Amazon SQS e richiama la funzione Lambda quando viene aggiunto un nuovo messaggio.

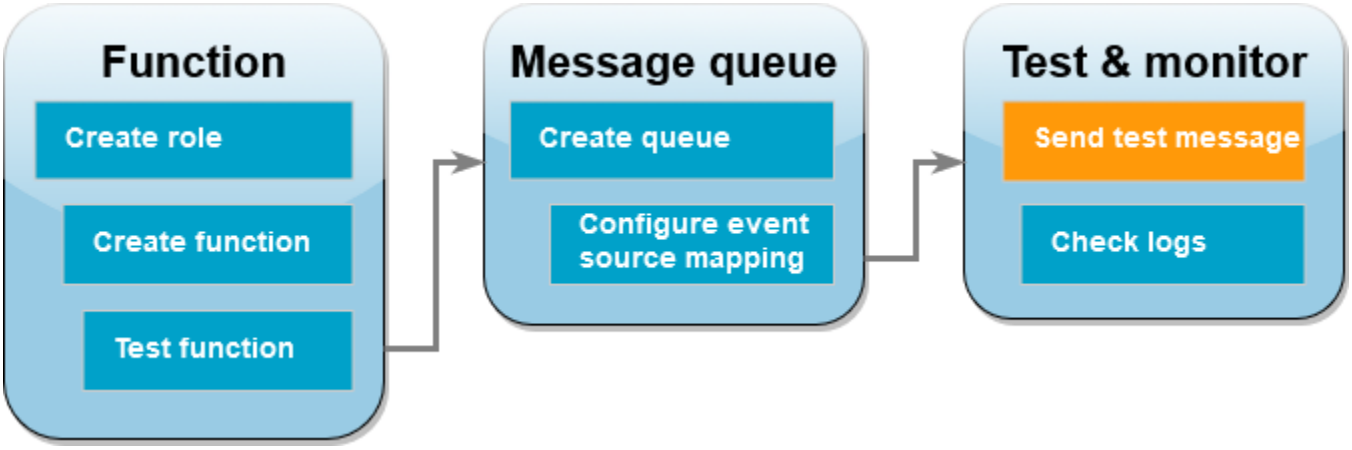
Per creare una mappatura tra la coda Amazon SQS e la funzione Lambda, usa il comando. create-event-source-mapping AWS CLI Esempio:

```
aws lambda create-event-source-mapping --function-name ProcessSQSRecord  --batch-size
 10 \
--event-source-arn arn:aws:sqs:us-east-1:111122223333:my-queue
```

Per ottenere un elenco delle mappature delle sorgenti degli eventi, usa il comando. list-event-source-mappings Esempio:

```
aws lambda list-event-source-mappings --function-name ProcessSQSRecord
```

Invio di un messaggio di test

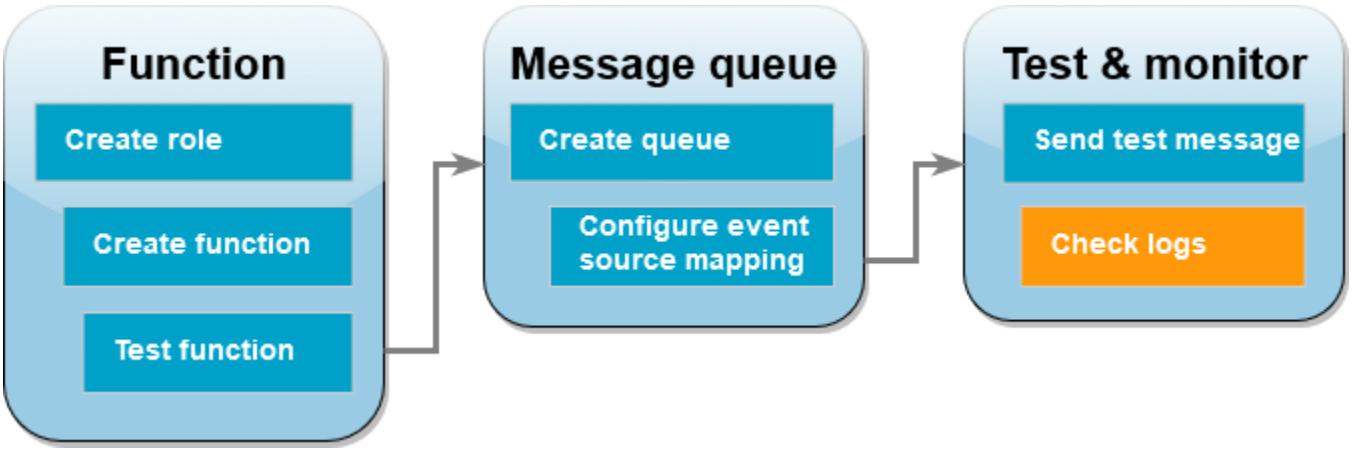


Invio di un messaggio Amazon SQS alla funzione Lambda

1. Apri la console Amazon SQS.
2. Scegli la coda creata in precedenza.
3. Scegli Invia e ricevi messaggi.
4. Nella sezione Corpo del messaggio, inserisci un messaggio di test, ad esempio "questo è un messaggio di prova".
5. Scegliere Invia messaggio.

Lambda esegue il polling della coda per gli aggiornamenti. Quando c'è un nuovo messaggio, Lambda richiama la tua funzione con questi nuovi dati di evento dalla coda. Se il gestore della funzione conclude senza eccezioni, Lambda considera il messaggio elaborato correttamente e inizia a leggere nuovi messaggi nella coda. Dopo aver elaborato correttamente un messaggio, Lambda lo elimina automaticamente dalla coda. Se il gestore genera un'eccezione, Lambda considera il batch dei messaggi come non correttamente elaborato e Lambda richiama la funzione con lo stesso batch di messaggi.

Controlla i log CloudWatch



Verifica della corretta elaborazione del messaggio da parte della funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la SQSRecord funzione Process.
3. Scegli Monitor (Monitoraggio).
4. Scegli Visualizza CloudWatch registri.
5. Nella CloudWatch console, scegli il flusso di log per la funzione.
6. Individua il log `INFO`. È qui che la funzione Lambda registra il corpo del messaggio. Dovresti vedere il messaggio che hai inviato dalla coda Amazon SQS. Esempio:

```
2023-09-11T22:49:12.730Z b0c41e9c-0556-5a8b-af83-43e59efeec71 INFO Processed
 message this is a test message.
```

Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili a tuo Account AWS carico.

Come eliminare il ruolo di esecuzione

1. Aprire la pagina Ruoli della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

Per eliminare la funzione Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Digita **`confirm`** nel campo di immissione testo e scegli Delete (Elimina).

Per eliminare la coda Amazon SQS

1. Accedi Console di gestione AWS e apri la console Amazon SQS all'indirizzo. https://console.aws.amazon.com/sqs/
2. Selezionare la coda creata.
3. Scegli Elimina.
4. Inserisci **`confirm`** nel campo di immissione del testo.
5. Scegli Delete (Elimina).

Tutorial: utilizzo di una coda Amazon SQS tra più account come origine eventi

In questo tutorial, crei una funzione Lambda che consuma i messaggi da una coda Amazon Simple Queue Service (Amazon SQS) in un account diverso. AWS Questo tutorial include due AWS account: l'account A si riferisce all'account che contiene la funzione Lambda e l'account B si riferisce all'account che contiene la coda Amazon SQS.

Prerequisiti

Installa il AWS Command Line Interface

Se non l'hai ancora installato AWS Command Line Interface, segui i passaggi indicati in Installazione o aggiornamento della versione più recente di AWS CLI per installarlo.

Per eseguire i comandi nel tutorial, sono necessari un terminale a riga di comando o una shell (interprete di comandi). In Linux e macOS, utilizza la shell (interprete di comandi) e il gestore pacchetti preferiti.

> **Note**
>
> Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere una versione integrata su Windows di Ubuntu e Bash, installa il sottosistema Windows per Linux.

Creazione del ruolo di esecuzione (account A)

Nell'Account A, crea un ruolo di esecuzione che autorizzi la funzione ad accedere alle AWS risorse richieste.

Creazione di un ruolo di esecuzione

1. Apri la pagina Ruoli nella console AWS Identity and Access Management (IAM).
2. Scegliere Create role (Crea ruolo).
3. Creare un ruolo con le seguenti proprietà.
   - Entità attendibile – AWS Lambda
   - Autorizzazioni: AWSLambdaSQSQueueExecutionRole
   - Nome ruolo – **`cross-account-lambda-sqs-role`**

La AWSLambdaSQSQueueExecutionRolepolicy dispone delle autorizzazioni necessarie alla funzione per leggere elementi da Amazon SQS e scrivere log su Amazon Logs. CloudWatch

Creazione della funzione (account A)

In Account A, crea una funzione Lambda che elabora i messaggi Amazon SQS. La funzione Lambda e la coda Amazon SQS devono trovarsi nella stessa Regione AWS.

Il seguente esempio di codice Node.js scrive ogni messaggio in un log in Logs. CloudWatch

Example index.mjs

```
export const handler = async function(event, context) {
  event.Records.forEach(record => {
    const { body } = record;
    console.log(body);
```

```
  });
  return {};
}
```

Creazione della funzione

> **Note**
>
> Seguendo questi passaggi viene creata una funzione Node.js. Per le altre lingue i passaggi sono simili, ma alcuni dettagli sono diversi.

1. Salva l'esempio di codice come un file denominato `index.mjs`.
2. Crea un pacchetto di implementazione.

   ```
   zip function.zip index.mjs
   ```

3. Create la funzione utilizzando il comando `create-function` AWS Command Line Interface (AWS CLI). Sostituisci `arn:aws:iam::111122223333:role/cross-account-lambda-sqs-role` con l'ARN del ruolo di esecuzione creato in precedenza.

   ```
   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
   ```

Testa la funzione (Account A)

Nell'Account A, verifica manualmente la tua funzione Lambda utilizzando il `invoke` AWS CLI comando e un evento Amazon SQS di esempio.

Se il gestore termina normalmente senza eccezioni, Lambda considera il messaggio come elaborato correttamente e inizia a leggere nuovi messaggi nella coda. Dopo aver elaborato correttamente un messaggio, Lambda lo elimina automaticamente dalla coda. Se il gestore genera un'eccezione, Lambda considera il batch dei messaggi come non correttamente elaborato e Lambda richiama la funzione con lo stesso batch di messaggi.

1. Salva il seguente JSON come un file denominato `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"
        }
    ]
}
```

Il precedente JSON simula un evento che Amazon SQS potrebbe inviare alla tua funzione Lambda, dove `"body"` contiene il messaggio effettivo dalla coda.

2. Eseguire il seguente comando `invoke` AWS CLI .

```
aws lambda invoke --function-name CrossAccountSQSExample \
--cli-binary-format raw-in-base64-out \
--payload file://input.txt outputfile.txt
```

L'cli-binary-formatopzione è obbligatoria se utilizzi la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

3. Verificare l'output nel file `outputfile.txt`.

Creazione di una coda Amazon SQS (account B)

In Account B, crea una coda Amazon SQS che la funzione Lambda in Account A può utilizzare come origine eventi. La funzione Lambda e la coda Amazon SQS devono trovarsi nella stessa Regione AWS.

Come creare una coda

1. Apri la console Amazon SQS.
2. Scegliere Crea coda.
3. Crea una coda con le seguenti proprietà.
   - Type (Tipo): Standard
   - Nome: LambdaCrossAccountQueue
   - Configuration (Configurazione): mantieni le impostazioni predefinite.
   - Access policy (Policy di accesso): scegli Advanced (Avanzata). Incolla la seguente politica JSON. Sostituisci i valori seguenti:
     - `111122223333`: Account AWS ID per l'account A
     - `444455556666`: Account AWS ID per l'account 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"
        }
    ]
}
```

     Questa policy concede al ruolo di esecuzione Lambda nell'Account A le autorizzazioni per utilizzare i messaggi provenienti da questa coda Amazon SQS.

4. Dopo aver creato la coda, registra il suo Amazon Resource Name (ARN). Questa operazione è necessaria nella fase successiva quando si associa la coda alla funzione Lambda.

Configurazione dell'origine eventi (account A)

Nell'Account A, crea una mappatura dell'origine degli eventi tra la coda Amazon SQS nell'Account B e la tua funzione Lambda eseguendo il comando seguente. `create-event-source-mapping` AWS CLI Sostituisci `arn:aws:sqs:us-east-1:444455556666:LambdaCrossAccountQueue` con l'ARN della coda Amazon SQS creata nel passaggio precedente.

```
aws lambda create-event-source-mapping --function-name CrossAccountSQSExample --batch-size 10 \
--event-source-arn arn:aws:sqs:us-east-1:444455556666:LambdaCrossAccountQueue
```

È possibile ottenere un elenco delle mappature delle fonti eventi eseguendo il comando riportato di seguito.

```
aws lambda list-event-source-mappings --function-name CrossAccountSQSExample \
--event-source-arn arn:aws:sqs:us-east-1:444455556666:LambdaCrossAccountQueue
```

Eseguire il test della configurazione

A questo punto è possibile eseguire il test della configurazione come indicato di seguito:

1. Nell'Account B, apri Console Amazon SQS.
2. Scegli LambdaCrossAccountQueue, che hai creato in precedenza.
3. Scegli Invia e ricevi messaggi.
4. In Corpo del messaggio, inserisci un messaggio di prova.
5. Scegliere Send Message (Invia messaggio).

La funzione Lambda nell'Account A dovrebbe ricevere il messaggio. Lambda continuerà a eseguire il polling della coda per gli aggiornamenti. Quando c'è un nuovo messaggio, Lambda richiama la tua funzione con questi nuovi dati di evento dalla coda. La tua funzione viene eseguita e crea registri in Amazon CloudWatch. Puoi visualizzare i log nella console CloudWatch .

Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili a tuo carico. Account AWS

Nell'Account A elimina il ruolo di esecuzione e la funzione Lambda.

Come eliminare il ruolo di esecuzione

1. Aprire la pagina Ruoli della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

Per eliminare la funzione Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Digita **`confirm`** nel campo di immissione testo e scegli Delete (Elimina).

Nell'Account B elimina la coda Amazon SQS.

Per eliminare la coda Amazon SQS

1. Accedi Console di gestione AWS e apri la console Amazon SQS all'indirizzo. https://console.aws.amazon.com/sqs/
2. Selezionare la coda creata.
3. Scegli Elimina.
4. Inserisci **`confirm`** nel campo di immissione del testo.
5. Scegli Delete (Elimina).

Orchestrazione delle funzioni con le Step Functions

Le funzioni Lambda che gestiscono più attività, implementano la logica di ripetizione dei tentativi o contengono la logica di ramificazione sono anti-pattern. Consigliamo invece di scrivere funzioni Lambda che eseguano singole attività e di utilizzarle AWS Step Functions per orchestrare i flussi di lavoro delle applicazioni.

Ad esempio, l'elaborazione di un ordine potrebbe richiedere la convalida dei dettagli dell'ordine, il controllo dei livelli di inventario, l'elaborazione del pagamento e la generazione di una fattura. Scrivi funzioni Lambda separate per ogni attività e usa Step Functions per gestire il flusso di lavoro. Step Functions coordina il flusso di dati tra le funzioni e gestisce gli errori in ogni fase. Questa separazione semplifica la visualizzazione, la modifica e la gestione dei flussi di lavoro man mano che diventano più complessi.

Quando usare Step Functions con Lambda

I seguenti scenari sono buoni esempi di quando utilizzare Step Functions per orchestrare applicazioni basate su Lambda.

- Elaborazione sequenziale
- Gestione complessa degli errori
- Flussi di lavoro condizionali e approvazioni umane
- Elaborazione parallela

Elaborazione sequenziale

L'elaborazione sequenziale si verifica quando un'attività deve essere completata prima che possa iniziare l'attività successiva. Ad esempio, in un sistema di elaborazione degli ordini, l'elaborazione dei pagamenti non può iniziare fino al completamento della convalida dell'ordine e la generazione della fattura deve attendere la conferma del pagamento. Scrivi funzioni Lambda separate per ogni attività e usa Step Functions per gestire la sequenza e gestire il flusso di dati tra le funzioni.

Esempio anti-modello

Una singola funzione Lambda gestisce l'intero flusso di lavoro di elaborazione degli ordini tramite:

- Richiamo di altre funzioni Lambda in sequenza
- Analisi e convalida delle risposte di ogni funzione

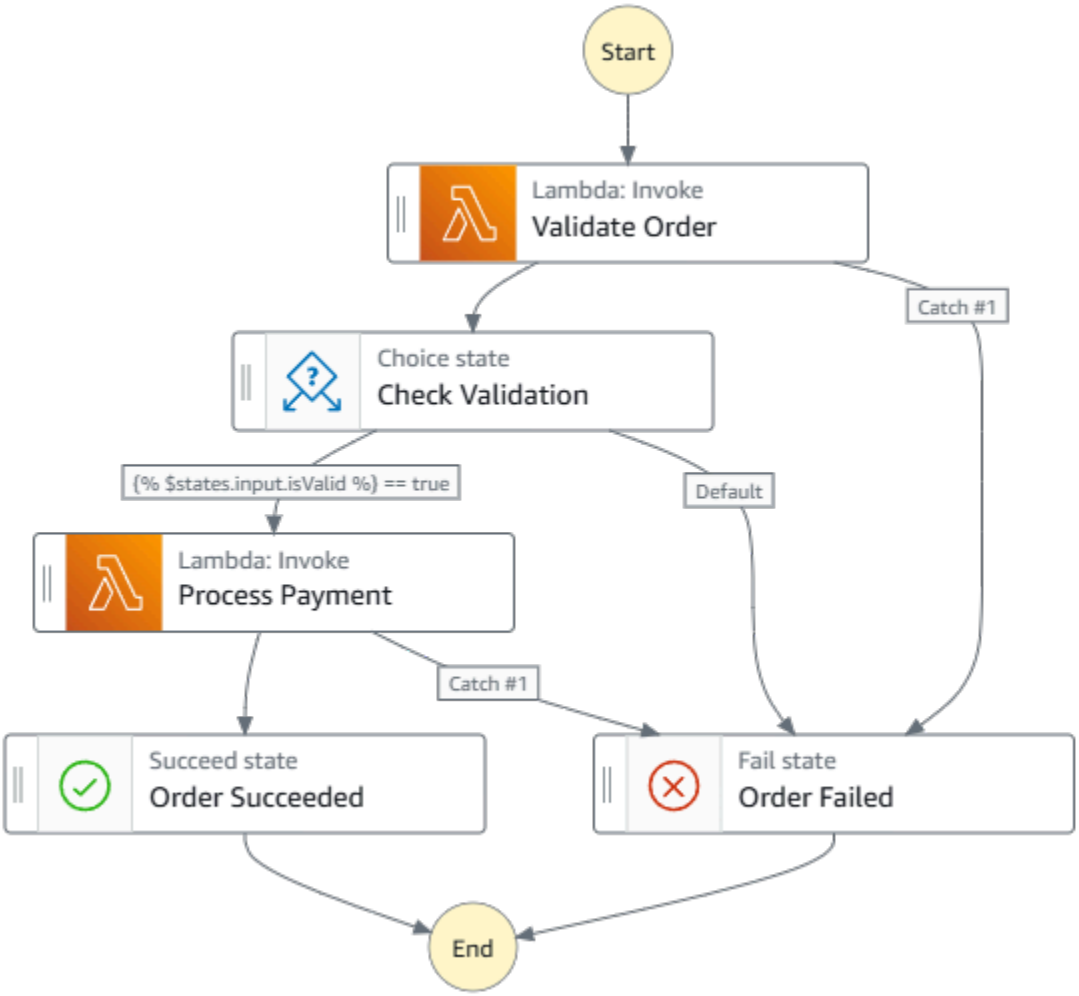
- Implementazione della logica di ripristino e della gestione degli errori
- Gestione del flusso di dati tra le funzioni

(Approccio consigliato)

Utilizza due funzioni Lambda: una per convalidare l'ordine e una per elaborare il pagamento. Step Functions coordina queste funzioni tramite:

- Esecuzione delle attività nella sequenza corretta
- Passaggio di dati tra funzioni
- Implementazione della gestione degli errori in ogni fase
- L'utilizzo di Choice indica che solo gli ordini validi procedono al pagamento

Example Grafico del flusso di lavoro



Gestione complessa degli errori

Mentre Lambda offre funzionalità di ripetizione dei tentativi per chiamate asincrone e mappature delle sorgenti degli eventi, Step Functions offre una gestione degli errori più sofisticata per flussi di lavoro complessi. È possibile configurare nuovi tentativi automatici con backoff esponenziale e impostare politiche di nuovo tentativo diverse per diversi tipi di errori. Quando i tentativi sono esauriti, utilizza per indirizzare gli errori `Catch` verso uno stato di fallback. Ciò è particolarmente utile quando è necessaria una gestione degli errori a livello di flusso di lavoro che coordini più funzioni e servizi.

Per ulteriori informazioni sulla gestione degli errori della funzione Lambda in una macchina a stati, vedere Gestione degli errori in The AWS Step Functions Workshop.

Esempio anti-modello

Una singola funzione Lambda gestisce tutto quanto segue:

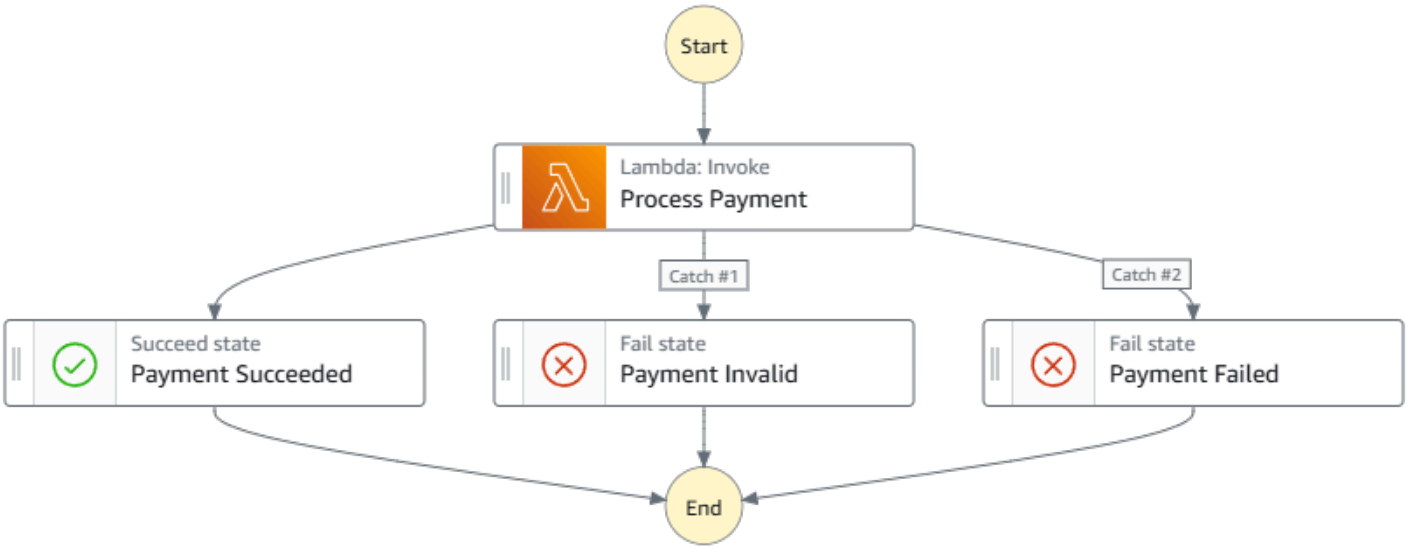
- Tentativi di chiamare un servizio di elaborazione dei pagamenti
- Se il servizio di pagamento non è disponibile, la funzione attende e riprova più tardi.
- Implementa un backoff esponenziale personalizzato per il tempo di attesa
- Dopo che tutti i tentativi sono falliti, catch l'errore e scegli un altro flusso

(Approccio consigliato)

Utilizza una singola funzione Lambda incentrata esclusivamente sull'elaborazione dei pagamenti. Step Functions gestisce la gestione degli errori tramite:

- Riprovare automaticamente le attività non riuscite con periodi di backoff configurabili
- Applicazione di politiche di riprova diverse in base ai tipi di errore
- Instradamento di diversi tipi di errori verso stati di fallback appropriati
- Mantenimento dello stato e della cronologia della gestione degli errori

Example Grafico del flusso di lavoro



Flussi di lavoro condizionali e approvazioni umane

Utilizza lo stato Step Functions Choice per indirizzare i flussi di lavoro in base all'output della funzione e il suffisso WaitForTaskToken per sospendere i flussi di lavoro per le decisioni umane. Ad esempio, per elaborare una richiesta di aumento del limite di credito, utilizza una funzione Lambda per valutare i fattori di rischio. Quindi, utilizza Step Functions per indirizzare le richieste ad alto rischio all'approvazione manuale e le richieste a basso rischio all'approvazione automatica.

Per implementare un flusso di lavoro di esempio che utilizza un modello di integrazione dei token di attività di callback, consulta Callback with Task Token in The Workshop. AWS Step Functions

Esempio anti-modello

Una singola funzione Lambda gestisce un flusso di lavoro di approvazione complesso mediante:

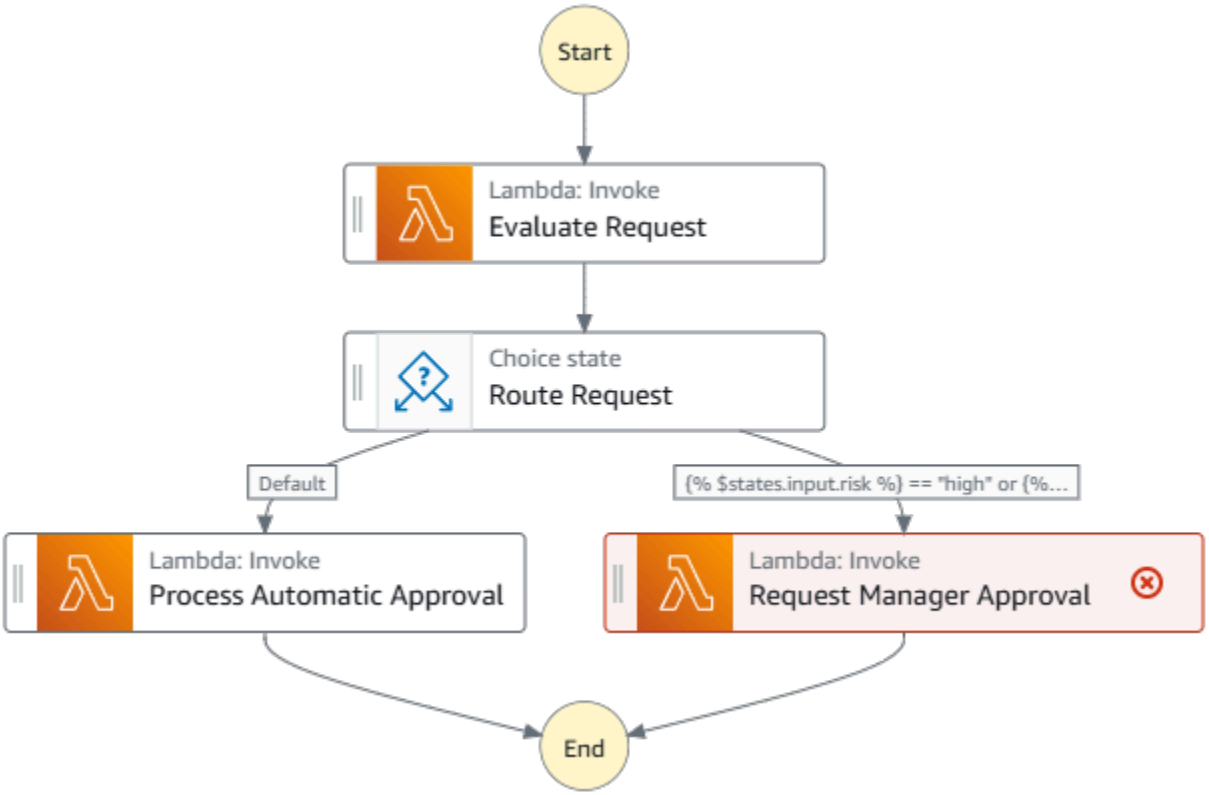
- Implementazione della logica condizionale annidata per valutare le richieste di credito
- Richiamo di diverse funzioni di approvazione in base agli importi delle richieste
- Gestione di più percorsi di approvazione e punti decisionali
- Monitoraggio dello stato delle approvazioni in sospeso
- Implementazione del timeout e della logica di notifica per le approvazioni

(Approccio consigliato)

Utilizza tre funzioni Lambda: una per valutare il rischio di ogni richiesta, una per approvare le richieste a basso rischio e una per indirizzare le richieste ad alto rischio a un manager per la revisione. Step Functions gestisce il flusso di lavoro tramite:

- Utilizzo degli stati Choice per indirizzare le richieste in base all'importo e al livello di rischio
- Sospensione dell'esecuzione in attesa dell'approvazione umana
- Gestione dei timeout per le approvazioni in sospeso
- Fornire visibilità sullo stato corrente di ogni richiesta

Example Grafico del flusso di lavoro



Elaborazione parallela

Step Functions offre tre modi per gestire l'elaborazione parallela:

- Lo stato Parallel esegue più rami del flusso di lavoro contemporaneamente. Usalo quando devi eseguire diverse funzioni in parallelo, come la generazione di miniature durante l'estrazione dei metadati delle immagini.
- Lo stato Inline Map elabora matrici di dati con un massimo di 40 iterazioni simultanee. Utilizzatelo per set di dati di piccole e medie dimensioni in cui è necessario eseguire la stessa operazione su ciascun elemento.
- Lo stato Distributed Map gestisce l'elaborazione parallela su larga scala con un massimo di 10.000 esecuzioni simultanee, supportando sia gli array JSON che le origini dati Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3). Usalo quando elabori set di dati di grandi dimensioni o quando hai bisogno di una maggiore concorrenza.

Esempio anti-modello

Una singola funzione Lambda tenta di gestire l'elaborazione parallela mediante:

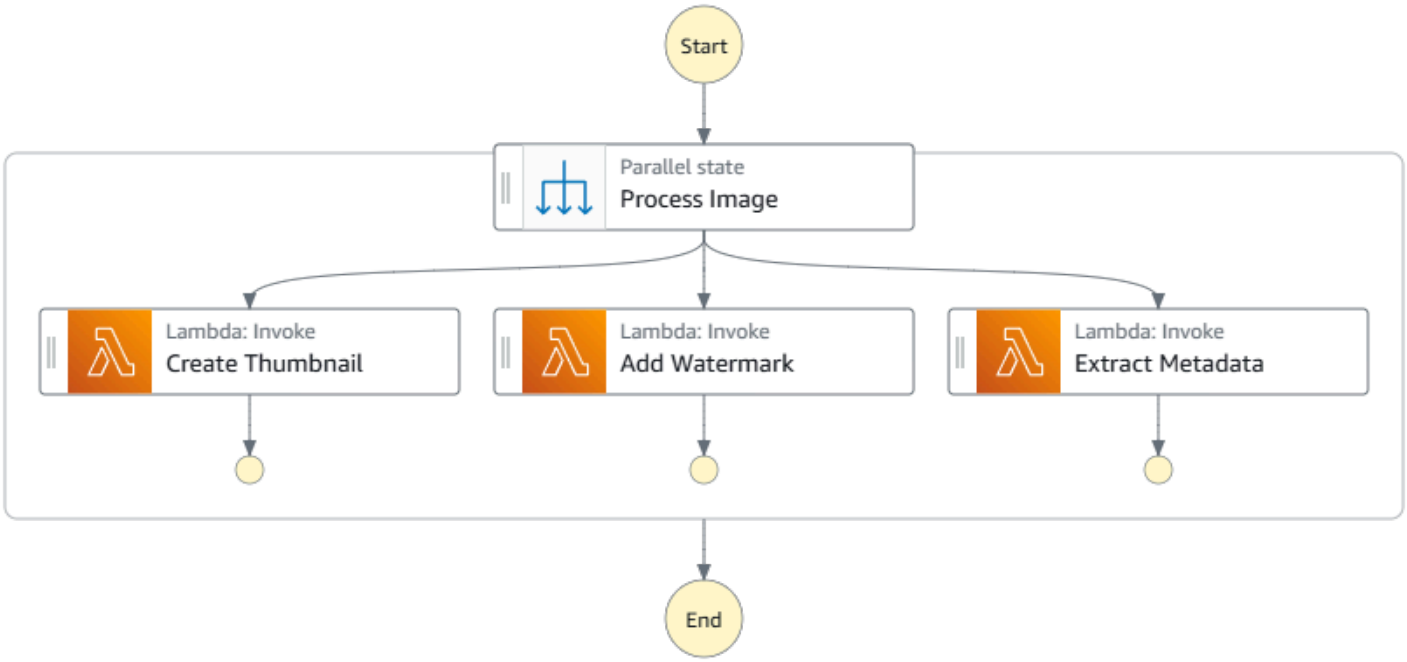
- Richiamo simultaneo di più funzioni di elaborazione delle immagini
- Implementazione di una logica di esecuzione parallela personalizzata
- Gestione dei timeout e degli errori per ogni attività parallela
- Raccolta e aggregazione dei risultati di tutte le funzioni

(Approccio consigliato)

Usa tre funzioni Lambda: una per creare un'immagine in miniatura, una per aggiungere una filigrana e una per estrarre i metadati. Step Functions gestisce queste funzioni tramite:

- Esecuzione simultanea di tutte le funzioni utilizzando lo stato Parallel
- Raccolta dei risultati da ciascuna funzione in un array ordinato
- Gestione dei timeout e degli errori in tutte le esecuzioni parallele
- Procedere solo quando tutti i rami paralleli sono completi

Example Grafico del flusso di lavoro



Quando non usare Step Functions con Lambda

Non tutte le applicazioni basate su Lambda traggono vantaggio dall'utilizzo di Step Functions. Considerate questi scenari quando scegliete l'architettura dell'applicazione.

- Un'applicazione semplice
- Elaborazione dati complessa
- Carichi di lavoro a elevati requisiti di lettura

Un'applicazione semplice

Per le applicazioni che non richiedono un'orchestrazione complessa, l'utilizzo di Step Functions potrebbe aggiungere complessità non necessarie. Ad esempio, se stai semplicemente elaborando messaggi da una coda Amazon SQS o rispondendo a eventi Amazon EventBridge, puoi configurare questi servizi per richiamare direttamente le tue funzioni Lambda. Allo stesso modo, se l'applicazione è composta solo da una o due funzioni Lambda con una gestione semplice degli errori, l'invocazione diretta Lambda o le architetture basate sugli eventi potrebbero essere più semplici da implementare e gestire.

Elaborazione dati complessa

Puoi utilizzare lo stato Step Functions Distributed Map per elaborare contemporaneamente set di dati Amazon S3 di grandi dimensioni con funzioni Lambda. Ciò è efficace per molti carichi di lavoro paralleli su larga scala, inclusa l'elaborazione di dati semistrutturati come file JSON o CSV. Tuttavia, per trasformazioni di dati più complesse o analisi avanzate, prendi in considerazione queste alternative:

- Pipeline di trasformazione dei dati: utilizzate AWS Glue per lavori ETL che elaborano dati strutturati o semistrutturati provenienti da più fonti. AWS Glueè particolarmente utile quando sono necessarie funzionalità integrate di gestione del catalogo di dati e dello schema.
- Analisi dei dati: usa Amazon EMR per analisi dei dati su scala petabyte, soprattutto quando hai bisogno di strumenti Apache Hadoop ecosistemici o per carichi di lavoro di machine learning che superano i limiti di memoria di Lambda.

Carichi di lavoro a elevati requisiti di lettura

Sebbene Step Functions sia in grado di orchestrare attività che richiedono un uso intensivo della CPU, le funzioni Lambda potrebbero non essere adatte a questi carichi di lavoro a causa delle risorse limitate della CPU. Per le operazioni ad alta intensità di calcolo all'interno dei flussi di lavoro, prendi in considerazione queste alternative:

- Orchestrazione dei container: usa Step Functions per gestire le attività di Amazon Elastic Container Service (Amazon ECS) per risorse di elaborazione più coerenti e scalabili.
- Elaborazione in batch: integrazione AWS Batch con Step Functions per la gestione di processi batch ad alta intensità di calcolo che richiedono un utilizzo prolungato della CPU.

Richiamo di una funzione Lambda con eventi batch Amazon S3

È possibile utilizzare le operazioni batch di Amazon S3 per richiamare una funzione Lambda su un set di oggetti Amazon S3 di grandi dimensioni. Amazon S3 tiene traccia dello stato di avanzamento delle operazioni batch, invia notifiche e memorizza un report di completamento che mostra lo stato di ogni operazione.

Per eseguire un'operazione batch, è possibile creare un lavoro di operazioni batch Amazon S3. Quando si crea il lavoro, si fornisce un file manifest (l'elenco degli oggetti) e si configura l'operazione da eseguire su tali oggetti.

All'avvio del lavoro batch, Amazon S3 richiama la funzione Lambda in modo sincrono per ogni oggetto nel manifest. Il parametro evento include i nomi del bucket e dell'oggetto.

Nell'esempio seguente viene illustrato l'evento che Amazon S3 invia alla funzione Lambda per un oggetto denominato customerImage1.jpg nel bucket amzn-s3-demo-bucket.

Example Evento di richiesta 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"
        }
    ]
}
```

La funzione Lambda deve restituire un oggetto JSON con i campi come mostrato nell'esempio seguente. È possibile copiare `invocationId` e `taskId` dal parametro evento. È possibile restituire una stringa in `resultString`. Amazon S3 salva i valori `resultString` nel report di completamento.

Example Risposta alla richiesta batch Amazon S3

```
{
  "invocationSchemaVersion": "1.0",
  "treatMissingKeysAs" : "PermanentFailure",
  "invocationId" : "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",
  "results": [
    {
      "taskId": "dGFza2lkZ29lc2hlcmUK",
      "resultCode": "Succeeded",
      "resultString": "[\"Alice\", \"Bob\"]"
    }
  ]
}
```

Chiamata di funzioni Lambda dalle operazioni in batch Amazon S3

È possibile richiamare la funzione Lambda con una funzione ARN non qualificata o qualificata. Se si desidera utilizzare la stessa versione di funzione per l'intero lavoro batch, configurare una versione di funzione specifica nel parametro `FunctionARN` quando si crea il lavoro. Se si configura un alias o il qualificatore $LATEST, il lavoro batch inizia immediatamente a chiamare la nuova versione della funzione se l'alias o $LATEST viene aggiornato durante l'esecuzione del lavoro.

Si noti che non è possibile riutilizzare una funzione basata su eventi Amazon S3 esistente per le operazioni batch. Questo perché l'operazione batch Amazon S3 passa un parametro di evento diverso alla funzione Lambda e si aspetta un messaggio di ritorno con una struttura JSON specifica.

Nella policy basata sulle risorse creata per il lavoro batch Amazon S3, assicurarsi di impostare l'autorizzazione per il lavoro per richiamare la funzione Lambda.

Nel ruolo di esecuzione per la funzione, impostare una policy di attendibilità Amazon S3 per assumere il ruolo quando viene eseguita la funzione.

Se la funzione utilizza SDK di AWS per gestire le risorse Amazon S3, è necessario aggiungere le autorizzazioni Amazon S3 nel ruolo di esecuzione.

Quando il lavoro viene eseguito, Amazon S3 avvia più istanze di funzione per elaborare gli oggetti Amazon S3 in parallelo, fino al limite di simultaneità della funzione. Amazon S3 limita l'aumento iniziale delle istanze per evitare costi eccessivi per i lavori più piccoli.

Se la funzione Lambda restituisce un codice di risposta `TemporaryFailure`, Amazon S3 riattiva l'operazione.

Per ulteriori informazioni sulla gestione delle operazioni in batch Amazon S3, consulta Gestione dei processi di operazioni in batch nella Guida per gli sviluppatori di Amazon S3.

Per un esempio di come utilizzare una funzione Lambda nelle operazioni batch di Amazon S3, consulta Invocare una funzione Lambda da operazioni batch di Amazon S3 nella Guida per gli sviluppatori di Amazon S3.

Richiamo di funzioni Lambda mediante notifiche Amazon SNS

Si può utilizzare una funzione Lambda per elaborare le notifiche Amazon Simple Notification Service (Amazon SNS). Amazon SNS supporta le funzioni Lambda come destinazione per i messaggi inviati a un argomento. Puoi sottoscrivere la funzione ad argomenti nello stesso account o in altri account AWS. Per la procedura guidata dettagliata, consulta the section called “Tutorial”.

Lambda supporta i trigger SNS solo per argomenti SNS standard. Gli argomenti FIFO non sono supportati.

Lambda elabora i messaggi SNS in modo asincrono mettendoli in coda e gestendo i nuovi tentativi. Se Amazon SNS non è in grado di raggiungere Lambda o il messaggio viene rifiutato, Amazon SNS riprova a intervalli crescenti per diverse ore. Per ulteriori informazioni, consulta Affidabilità nelle domande frequenti su Amazon SNS.

> **Warning**
>
> Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per evitare potenziali problemi legati agli eventi duplicati, ti consigliamo vivamente di rendere idempotente il codice della funzione. Per ulteriori informazioni, consulta Come posso rendere idempotente la mia funzione Lambda nel Knowledge Center AWS.

Utilità di idempotenza di Powertools per Lambda AWS

L'utilità di idempotenza di Powertools for AWS Lambda rende le funzioni Lambda idempotenti. È disponibile per Python, TypeScript, Java e.NET. Per ulteriori informazioni, vedere l'utilità Idempotency nella documentazione di Powertools for AWS Lambda (Python), l'utilità Idempotency nella documentazione di Powertools for Lambda (TypeScript), l'utilità Idempotency nella documentazione di Powertools AWS for Lambda (Java) e l'utilità Idempotency nella documentazione di Powertools AWS for Lambda (.NET). AWS

Argomenti

- Aggiunta di un trigger di argomento Amazon SNS per una funzione Lambda utilizzando la console
- Aggiunta manuale di un trigger di argomento Amazon SNS per una funzione Lambda
- Esempio di forma evento SNS
- Tutorial: Utilizzo AWS Lambda con Amazon Simple Notification Service

Aggiunta di un trigger di argomento Amazon SNS per una funzione Lambda utilizzando la console

Per aggiungere un argomento SNS come trigger per una funzione Lambda, il modo più semplice è utilizzare la console Lambda. Quando aggiungi il trigger tramite la console, Lambda configura automaticamente le autorizzazioni e gli abbonamenti necessari per iniziare a ricevere eventi dall'argomento SNS.

Per aggiungere un argomento SNS come trigger per una funzione Lambda (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli il nome della funzione per la quale si desidera aggiungere il trigger.
3. Scegli Configurazione, quindi scegli Trigger.
4. Selezionare Add trigger (Aggiungi trigger).
5. In Configurazione del trigger, nell'elenco a discesa, scegli SNS.
6. Per Argomento SNS, scegli l'argomento SNS da sottoscrivere.

Aggiunta manuale di un trigger di argomento Amazon SNS per una funzione Lambda

Per configurare manualmente un trigger SNS per una funzione Lambda, è necessario completare i seguenti passaggi:

- Definisci una policy basata sulle risorse per la tua funzione per consentire a SNS di richiamarla.
- Sottoscrivere la funzione Lambda all'argomento Amazon SNS.

> **Note**
>
> Se l'argomento SNS e la funzione Lambda si trovano in account AWS diversi, è inoltre necessario concedere autorizzazioni aggiuntive per consentire le sottoscrizioni tra account all'argomento SNS. Per maggiori informazioni, consulta Concedere l'autorizzazioni tra account per la sottoscrizione di Amazon SNS.

Puoi usare la AWS Command Line Interface (AWS CLI) per completare entrambi questi passaggi. Innanzitutto, per definire una policy basata sulle risorse per una funzione Lambda che consenta le

chiamate SNS, usa il comando AWS CLI seguente. Assicurati di sostituire il valore di `--function-name` con il nome della funzione Lambda e il valore di `--source-arn` con l'ARN dell'argomento SNS.

```
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
```

Per sottoscrivere la tua funzione all'argomento SNS, usa il comando AWS CLI seguente. Assicurati di sostituire il valore di `--topic-arn` con l'ARN dell'argomento SNS e il valore di `--notification-endpoint` con l'ARN della funzione Lambda.

```
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
```

Esempio di forma evento SNS

Amazon SNS richiama la funzione in modo asincrono con un evento che contiene un messaggio e dei metadati.

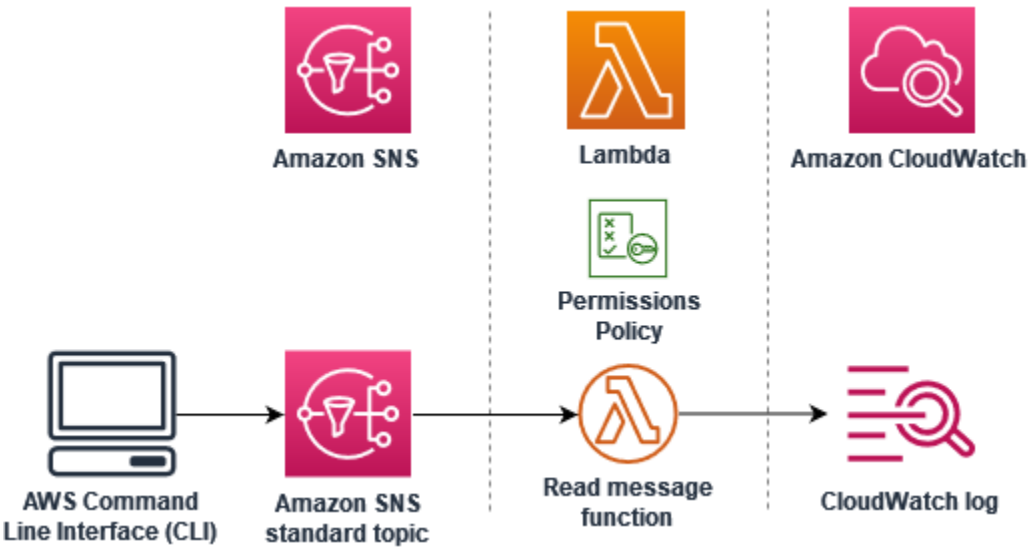
Example Evento messaggio di Amazon SNS

```
{
  "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: Utilizzo AWS Lambda con Amazon Simple Notification Service

In questo tutorial, utilizzi una funzione Lambda in un'unica funzione Account AWS per iscriverti a un argomento di Amazon Simple Notification Service (Amazon SNS) in un altro argomento. Account AWS Quando pubblichi messaggi sul tuo argomento Amazon SNS, la funzione Lambda legge il contenuto del messaggio e lo invia ad Amazon Logs. CloudWatch Per completare questo tutorial, usa il (). AWS Command Line Interface AWS CLI



Per completare questo tutorial, esegui i passaggi riportati:

- Nell'account A, crea un argomento Amazon SNS.
- Nell'account B, crea una funzione Lambda che leggerà i messaggi dall'argomento.
- Nell'account B, crea una sottoscrizione all'argomento.
- Pubblica messaggi sull'argomento Amazon SNS nell'account A e conferma che la funzione Lambda nell'account B li invii nei log. CloudWatch

Completando questi passaggi, imparerai come configurare un argomento Amazon SNS per richiamare una funzione Lambda. Imparerai anche come creare una policy AWS Identity and Access Management (IAM) che autorizzi una risorsa in un'altra Account AWS a richiamare Lambda.

Nel tutorial, vengono utilizzati due Account AWS separati. I AWS CLI comandi illustrano ciò utilizzando due profili denominati chiamati `accountA` `eaccountB`, ciascuno configurato per l'uso con un altro. Account AWS Per informazioni su come configurare l'utilizzo AWS CLI di profili diversi, consulta Impostazioni dei file di configurazione e credenziali nella Guida per l'AWS Command Line Interface utente della versione 2. Assicurati di configurare lo stesso valore predefinito Regione AWS per entrambi i profili.

Se i AWS CLI profili creati per i due Account AWS utilizzano nomi diversi o se utilizzi il profilo predefinito e un profilo denominato, modifica i AWS CLI comandi nei passaggi seguenti in base alle esigenze.

Prerequisiti

Installa il AWS Command Line Interface

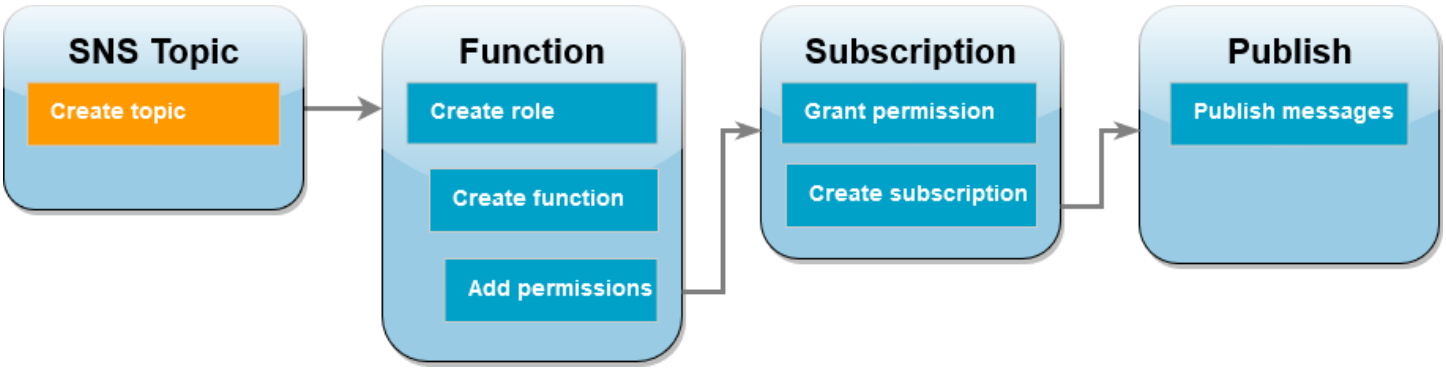
Se non l'hai ancora installato AWS Command Line Interface, segui i passaggi indicati in Installazione o aggiornamento della versione più recente di AWS CLI per installarlo.

Per eseguire i comandi nel tutorial, sono necessari un terminale a riga di comando o una shell (interprete di comandi). In Linux e macOS, utilizza la shell (interprete di comandi) e il gestore pacchetti preferiti.

> **Note**
>
> Su Windows, alcuni comandi della CLI Bash utilizzati comunemente con Lambda (ad esempio, `zip`) non sono supportati dai terminali integrati del sistema operativo. Per ottenere

una versione integrata su Windows di Ubuntu e Bash, installa il sottosistema Windows per Linux.

Creare un argomento Amazon SNS (account A)



Per creare l'argomento

- Nell'account A, crea un argomento standard di Amazon SNS utilizzando il seguente AWS CLI comando.

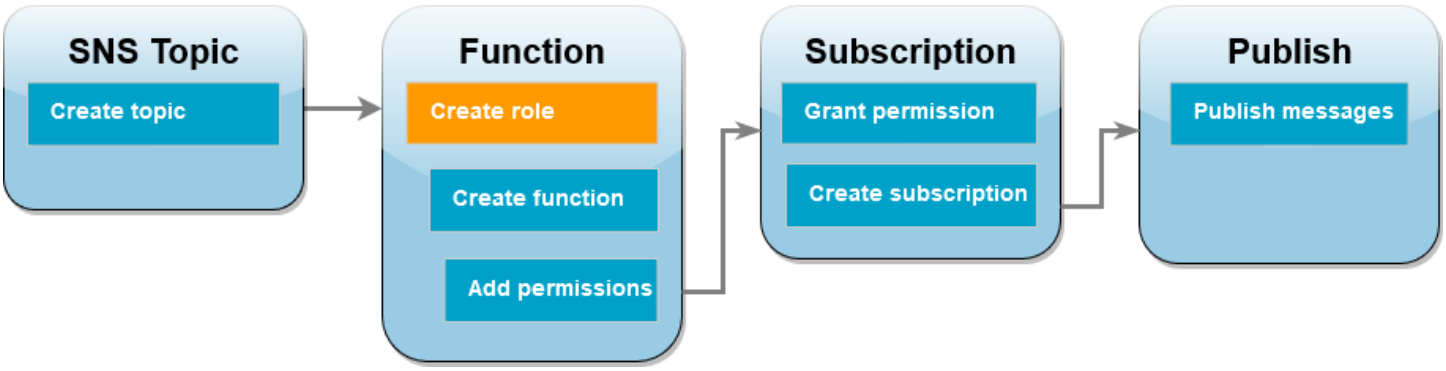
  ```
  aws sns create-topic --name sns-topic-for-lambda --profile accountA
  ```

  Verrà visualizzato un output simile al seguente.

  ```
  {
      "TopicArn": "arn:aws:sns:us-west-2:123456789012:sns-topic-for-lambda"
  }
  ```

  Prendi nota del nome della risorsa Amazon (ARN) dell'argomento. Sarà necessario in seguito nel tutorial, quando aggiungerai autorizzazioni alla funzione Lambda per effettuare la sottoscrizione all'argomento.

Creazione di un ruolo di esecuzione della funzione (account B)

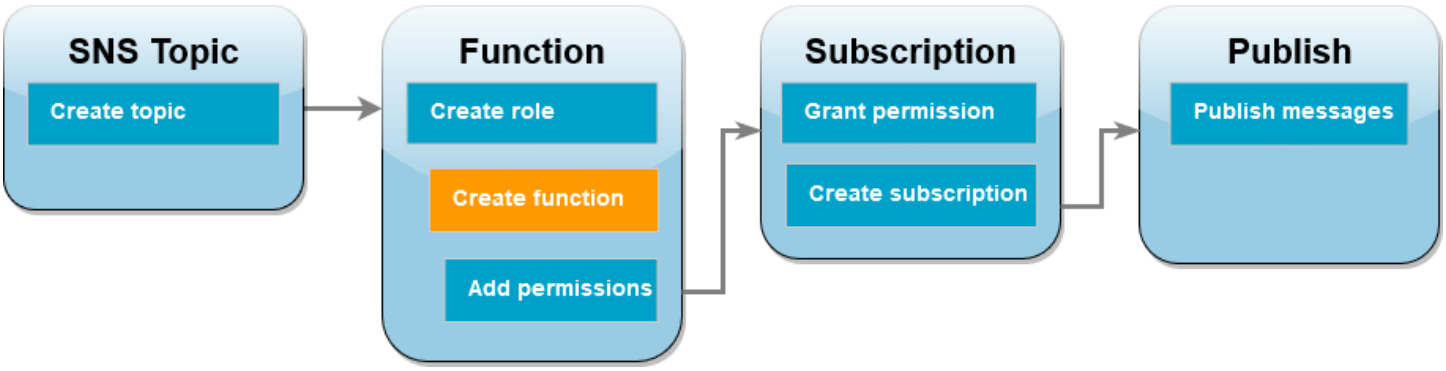


Un ruolo di esecuzione è un ruolo IAM che concede a una funzione Lambda l'autorizzazione all' Servizi AWS accesso e alle risorse. Prima di creare la funzione nell'account B, crei un ruolo che fornisce alla funzione le autorizzazioni di base per scrivere i log nei registri. CloudWatch Aggiungeremo le autorizzazioni per la lettura dal tuo argomento Amazon SNS in un passaggio successivo.

Creazione di un ruolo di esecuzione

1. Nell'account B apri la pagina dei ruoli nella console IAM.
2. Scegli Crea ruolo.
3. Per Tipo di entità attendibile, scegli Servizio AWS .
4. In Caso d'uso, scegli Lambda.
5. Scegli Next (Successivo).
6. Aggiungi una policy di autorizzazioni di base al ruolo completando le seguenti operazioni:
   a. Nella casella di ricerca Policy di autorizzazione, inserisci **`AWSLambdaBasicExecutionRole`**.
   b. Scegli Next (Successivo).
7. Completa la creazione del ruolo effettuando le seguenti operazioni:
   a. In Dettagli ruolo, immetti **`lambda-sns-role`** per Nome ruolo.
   b. Scegli Crea ruolo.

Creare una funzione Lambda (account B)



Crea una funzione Lambda che elabora i messaggi Amazon SNS. Il codice funzione registra il contenuto dei messaggi di ogni record in Amazon CloudWatch Logs.

Questo tutorial utilizza il runtime Node.js 24, ma abbiamo anche fornito codice di esempio in altri linguaggi di runtime. Per visualizzare il codice per il runtime che ti interessa, seleziona la scheda corrispondente nella casella seguente. Il JavaScript codice che utilizzerai in questo passaggio si trova nel primo esempio mostrato nella JavaScriptscheda.

.NET

SDK per .NET

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento SNS con JavaScript Lambda utilizzando.

```
// 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;
  }
}
```

Consumo di un evento SNS con TypeScript Lambda utilizzando.

```
// 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite 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
}
```

Creazione della funzione

1. Crea una directory per il progetto, quindi passa a quella directory.

   ```
   mkdir sns-tutorial
   cd sns-tutorial
   ```

2. Copia il JavaScript codice di esempio in un nuovo file denominato`index.js`.

3. Crea un pacchetto di implementazione utilizzando il seguente comando `zip`.

```
zip function.zip index.js
```

4. Esegui il AWS CLI comando seguente per creare la tua funzione Lambda nell'account B.

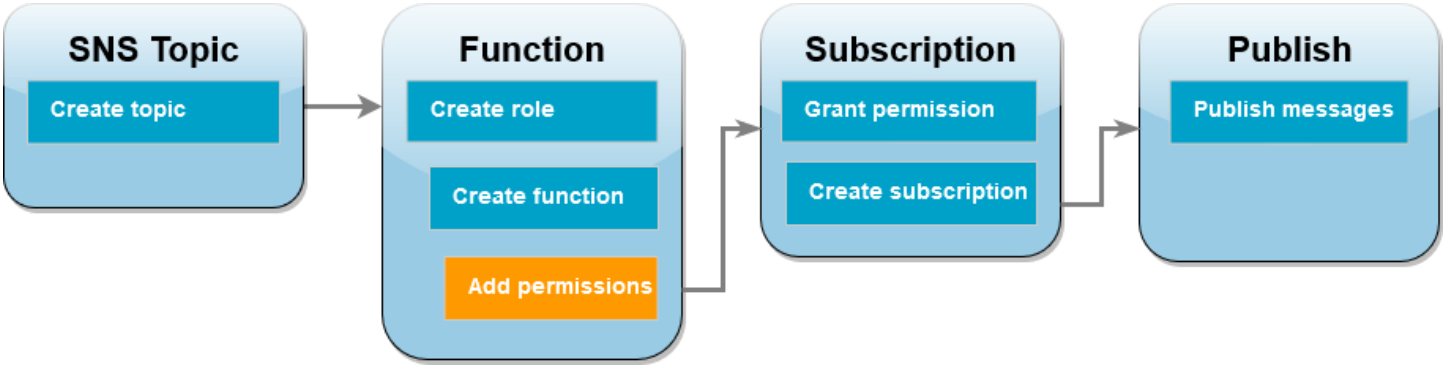
```
aws lambda create-function --function-name Function-With-SNS \
    --zip-file fileb://function.zip --handler index.handler --runtime nodejs24.x \
    --role arn:aws:iam::<AccountB_ID>:role/lambda-sns-role  \
    --timeout 60 --profile accountB
```

   Verrà visualizzato un output simile al seguente.

```
{
    "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. Registra il nome della risorsa Amazon (ARN) della funzione. Sarà necessario in seguito nel tutorial, quando aggiungerai autorizzazioni per consentire ad Amazon SNS di richiamare la funzione.

Aggiunta di autorizzazioni alla funzione (account B)



Perché Amazon SNS richiami la tua funzione, è necessario concedergli l'autorizzazione in una istruzione su una policy basata sulle risorse. Si aggiunge questa dichiarazione utilizzando il AWS CLI `add-permission` comando.

Concessione dell'autorizzazione di Amazon SNS per richiamare la tua funzione

- Nell'account B, esegui il AWS CLI comando seguente utilizzando l'ARN per l'argomento Amazon SNS che hai registrato in precedenza.

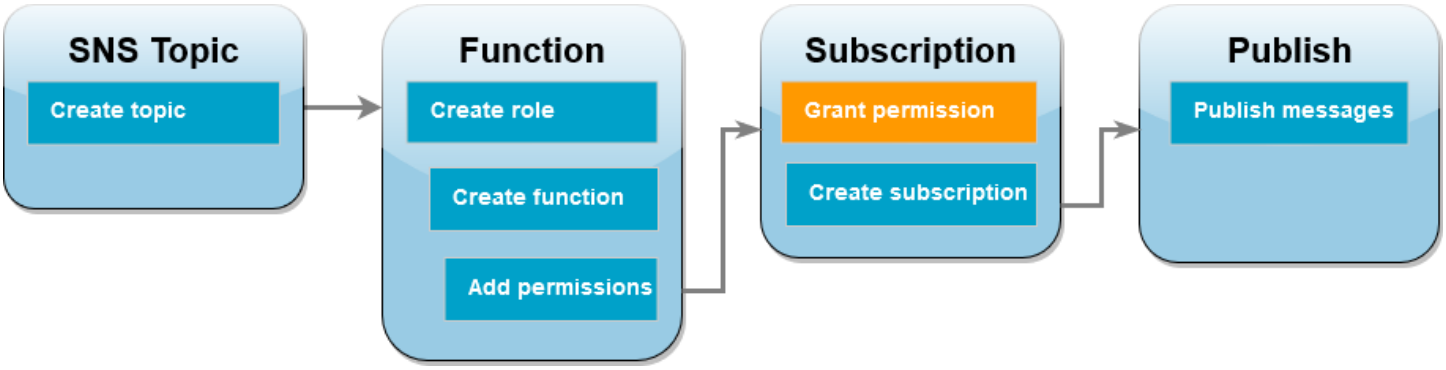
```
aws lambda add-permission --function-name Function-With-SNS \
    --source-arn arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-lambda \
    --statement-id function-with-sns --action "lambda:InvokeFunction" \
    --principal sns.amazonaws.com --profile accountB
```

  Verrà visualizzato un output simile al seguente.

```
{
    "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**
>
> Se l'account con l'argomento Amazon SNS è ospitato in un opt-in Regione AWS, devi specificare la regione nell'area principale. Ad esempio, se stai lavorando con un argomento Amazon SNS nella regione Asia Pacifico (Hong Kong), per il principale devi specificare `sns.ap-east-1.amazonaws.com` invece di `sns.amazonaws.com`.

Concessione dell'autorizzazione tra più account per la sottoscrizione ad Amazon SNS (account A)



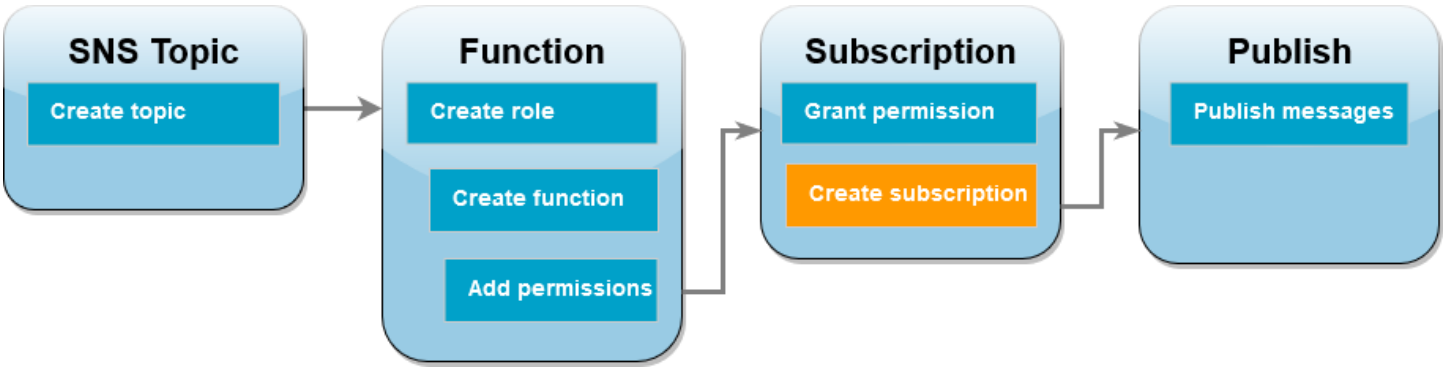
Perché la funzione Lambda nell'account B sottoscriva l'argomento Amazon SNS che hai creato nell'account A, è necessario concedere l'autorizzazione per l'account B in modo da sottoscrivere il tuo argomento. Concedi questa autorizzazione utilizzando il AWS CLI `add-permission` comando.

Concessione dell'autorizzazione per consentire all'account B la sottoscrizione all'argomento

- Nell'account A, esegui il seguente AWS CLI comando. Usa l'ARN per l'argomento Amazon SNS registrato in precedenza.

```
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
```

Creare una sottoscrizione (account B)



Nell'account B, ora sottoscrivi la tua funzione Lambda all'argomento Amazon SNS che hai creato all'inizio del tutorial nell'account A. Quando viene inviato un messaggio a questo argomento

(`sns-topic-for-lambda`), Amazon SNS richiama la funzione Lambda `Function-With-SNS` nell'account B.

Creazione di una sottoscrizione

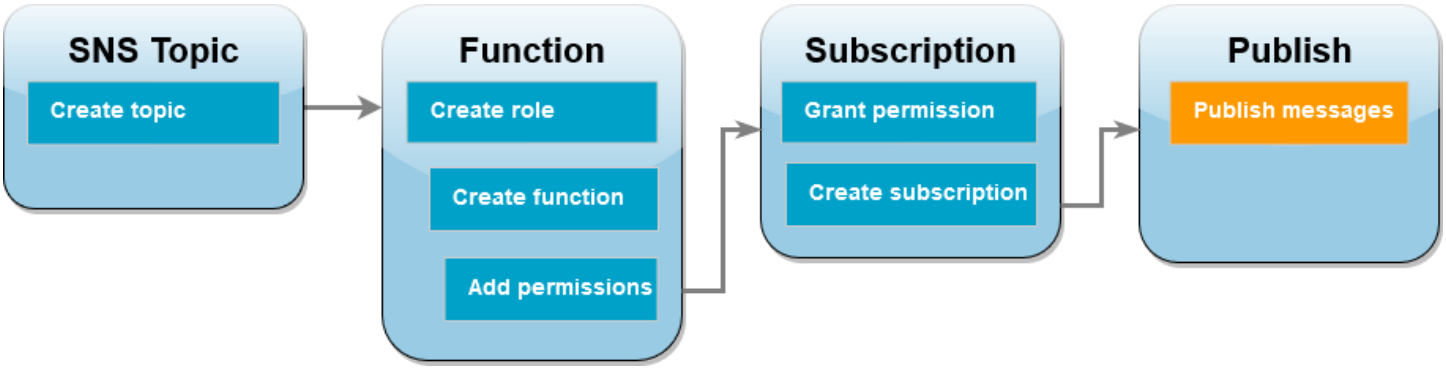
- Nell'account B, esegui il AWS CLI comando seguente. Usa l'area predefinita in cui hai creato l'argomento e la funzione ARNs for your topic e Lambda.

```
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
```

Verrà visualizzato un output simile al seguente.

```
{
    "SubscriptionArn": "arn:aws:sns:us-east-1:<AccountA_ID>:sns-topic-for-
lambda:5d906xxxx-7c8x-45dx-a9dx-0484e31c98xx"
}
```

Pubblicazione di messaggi sull'argomento (account A e account B)



Ora che la tua funzione Lambda nell'account B ha sottoscritto l'argomento Amazon SNS nell'account A, è il momento di testare la configurazione pubblicando messaggi sull'argomento. Per confermare che Amazon SNS ha richiamato la funzione Lambda, utilizza CloudWatch Logs per visualizzare l'output della funzione.

Pubblicazione di un messaggio sull' argomento e visualizzazione dell'output della tua funzione

1. Digita `Hello World` in un file di testo e salvalo come `message.txt`.
2. Dalla stessa directory in cui hai salvato il file di testo, esegui il seguente AWS CLI comando nell'account A. Usa l'ARN per il tuo argomento.

   ```
   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
   ```

   In questo modo verrà restituito un ID messaggio con un identificatore univoco, indicando che il messaggio è stato accettato da Amazon SNS. Amazon SNS prova quindi a consegnare il messaggio a tutti i sottoscrittori dell'argomento. Per confermare che Amazon SNS ha richiamato la funzione Lambda, usa CloudWatch Logs per visualizzare l'output della funzione:
3. Nell'account B, apri la pagina dei gruppi di log della CloudWatch console Amazon.
4. Scegli il nome del gruppo di log per la funzione (`/aws/lambda/Function-With-SNS`).
5. Scegli il flusso di log più recente.
6. Se la funzione è stata richiamata correttamente, vedrai un output simile al seguente che mostra il contenuto del messaggio pubblicato sull'argomento.

   ```
   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
   ```

Pulizia delle risorse

Ora è possibile eliminare le risorse create per questo tutorial, a meno che non si voglia conservarle. Eliminando AWS le risorse che non utilizzi più, eviti addebiti inutili ai tuoi Account AWS.

Nell'Account A, elimina l'argomento Amazon SNS.

Per eliminare l'argomento Amazon SNS

1. Aprire la pagina Topics (Argomenti) nella console Amazon SNS.
2. Selezionare l'argomento creato.
3. Scegli Elimina.
4. Inserisci **`delete me`** nel campo di immissione del testo.

5. Scegli Elimina.

Nell'Account B, elimina il ruolo di esecuzione, la funzione Lambda e la sottoscrizione Amazon SNS.

Come eliminare il ruolo di esecuzione

1. Aprire la pagina Ruoli della console IAM.
2. Selezionare il ruolo di esecuzione creato.
3. Scegliere Elimina.
4. Inserisci il nome del ruolo nel campo di immissione testo e seleziona Delete (Elimina).

Per eliminare la funzione Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Digita **`confirm`** nel campo di immissione testo e scegli Delete (Elimina).

Per eliminare la sottoscrizione Amazon SNS

1. Aprire la pagina Subscriptions (Sottoscrizioni) nella console Amazon SNS.
2. Selezionare la sottoscrizione creata.
3. Scegli Delete (Elimina), poi Delete (Elimina).

Gestione delle autorizzazioni in AWS Lambda

Puoi usare AWS Identity and Access Management (IAM) per gestire le autorizzazioni in. AWS Lambda Esistono due categorie principali di autorizzazioni da considerare quando si lavora con le funzioni Lambda:

- Autorizzazioni necessarie alle funzioni Lambda per eseguire azioni API e accedere ad altre risorse AWS
- Autorizzazioni necessarie ad altri AWS utenti ed entità per accedere alle tue funzioni Lambda

Le funzioni Lambda spesso devono accedere ad altre AWS risorse ed eseguire varie operazioni API su tali risorse. Ad esempio, è possibile che tu disponga di una funzione Lambda che risponde a un evento aggiornando le voci in un database Amazon DynamoDB. In questo caso, la funzione necessita delle autorizzazioni per accedere al database, nonché delle autorizzazioni per inserire o aggiornare elementi in quel database.

Definisci le autorizzazioni di cui ha bisogno la tua funzione Lambda in un ruolo IAM speciale chiamato ruolo di esecuzione. In questo ruolo, puoi allegare una policy che definisce tutte le autorizzazioni necessarie alla tua funzione per accedere ad altre AWS risorse e leggere le fonti degli eventi. Ogni funzione Lambda deve avere un ruolo di esecuzione. Come minimo, il ruolo di esecuzione deve avere accesso ad Amazon CloudWatch perché le funzioni Lambda accedono ai CloudWatch log per impostazione predefinita. Puoi collegare la policy gestita da `AWSLambdaBasicExecutionRole` al tuo ruolo di esecuzione per soddisfare questo requisito.

Per concedere Account AWS ad altre organizzazioni e servizi le autorizzazioni per accedere alle tue risorse Lambda, hai alcune opzioni:

- È possibile utilizzare le policy basate sull'identità per concedere agli utenti l'accesso alle risorse Lambda. Le policy basate su identità possono essere applicate direttamente agli utenti o ai gruppi e ruoli associati a un utente.
- Puoi utilizzare policy basate sulle risorse per concedere ad altri account e Servizi AWS autorizzazioni per accedere alle tue risorse Lambda. Quando un utente prova ad accedere a una risorsa Lambda, Lambda considera sia le policy basate su identità dell'utente che la policy basata sulla risorsa della risorsa. Quando un AWS servizio come Amazon Simple Storage Service (Amazon S3) richiama la funzione Lambda, Lambda considera solo la politica basata sulle risorse.

- Puoi utilizzare un modello controllo degli accessi basato su attributi (ABAC), per controllare l'accesso alle funzioni Lambda. Con ABAC, puoi collegare i tag a una funzione Lambda, inviarli in determinate richieste API o collegarli al principale IAM da cui proviene la richiesta. Per controllare gli accessi alla funzione, puoi invece specificare gli stessi tag nell'elemento di condizione di una policy IAM.

In generale AWS, è consigliabile concedere solo le autorizzazioni necessarie per eseguire un'attività (autorizzazioni con privilegi minimi). Per implementarlo in Lambda, consigliamo di iniziare con una policy gestita da AWS. Puoi utilizzare queste policy gestite così come sono o come punto di partenza per scrivere le tue policy più restrittive.

Per aiutarti a ottimizzare le autorizzazioni per l'accesso con privilegi minimi, Lambda fornisce alcune condizioni aggiuntive che puoi includere nelle tue policy. Per ulteriori informazioni, consulta the section called “Risorse e condizioni”.

Per ulteriori informazioni su IAM, consulta la Guida per l'utente di IAM.

Definizione delle autorizzazioni della funzione Lambda con un ruolo di esecuzione

Il ruolo di esecuzione di una funzione Lambda è un ruolo AWS Identity and Access Management (IAM) che concede alla funzione l'autorizzazione all'accesso Servizi AWS e alle risorse. Ad esempio, potresti creare un ruolo di esecuzione con l'autorizzazione a inviare log ad Amazon CloudWatch e caricare AWS X-Ray dati di traccia su. Questa pagina fornisce informazioni su come creare, visualizzare e gestire il ruolo di esecuzione di una funzione Lambda.

Lambda assume automaticamente il tuo ruolo di esecuzione quando richiami la tua funzione. Dovresti evitare di chiamare `sts:AssumeRole` manualmente per assumere il ruolo di esecuzione nel codice della funzione. Se il tuo caso d'uso richiede che il ruolo venga assunto autonomamente, dovrai includere il ruolo stesso come principale attendibile nella policy di attendibilità del ruolo. Per ulteriori informazioni su come modificare una policy di attendibilità del ruolo, consulta Modifica di una policy di attendibilità del ruolo (console) nella Guida per l'utente di IAM.

Affinché Lambda possa assumere correttamente il ruolo di esecuzione, la policy di attendibilità del ruolo deve specificare il principale del servizio Lambda (`lambda.amazonaws.com`) come servizio attendibile.

Argomenti

- Creazione di un ruolo di esecuzione nella console di IAM
- Creazione e gestione dei ruoli con AWS CLI
- Garantisci l'accesso minimo ai privilegi per il tuo ruolo di esecuzione Lambda
- Visualizzazione e aggiornamento delle autorizzazioni nel ruolo di esecuzione
- Utilizzo delle politiche AWS gestite nel ruolo di esecuzione
- Utilizzo dell'ARN della funzione di origine per controllare il comportamento di accesso alla funzione

Creazione di un ruolo di esecuzione nella console di IAM

Per impostazione predefinita, Lambda crea un ruolo di esecuzione con autorizzazioni minime quando si crea una funzione nella console Lambda. In particolare, questo ruolo di esecuzione include la policy `AWSLambdaBasicExecutionRole` gestita, che fornisce alla tua funzione le autorizzazioni di base per registrare gli eventi su Amazon CloudWatch Logs.

Per eseguire attività più significative, le funzioni in genere richiedono autorizzazioni aggiuntive. Ad esempio, è possibile che tu disponga di una funzione Lambda che risponde a un evento aggiornando le voci in un database Amazon DynamoDB. È possibile creare un ruolo di esecuzione con le autorizzazioni necessarie utilizzando la console IAM.

Per creare un ruolo di esecuzione nella console IAM

1. Aprire la pagina Roles (Ruoli) nella console IAM.
2. Scegli Crea ruolo.
3. In Tipo di entità attendibile, scegli Servizio AWS .
4. In Use case (Caso d'uso), scegli Lambda.
5. Scegli Next (Successivo).
6. Seleziona le politiche AWS gestite che desideri associare al tuo ruolo. Ad esempio, se la tua funzione deve accedere a DynamoDB, seleziona la policy gestita da DBExecutionDynamo AWSLambda Role.
7. Scegli Next (Successivo).
8. Immettere un Role name (Nome ruolo), quindi selezionare Create role (Crea ruolo).

Per istruzioni dettagliate, consulta Creazione di un ruolo per un AWS servizio (console) nella Guida per l'utente IAM.

Dopo aver creato il ruolo di esecuzione, collegalo alla funzione. Quando si crea una funzione nella console Lambda, potrai collegare alla funzione qualsiasi ruolo di esecuzione creato in precedenza. Se desideri associare un nuovo ruolo di esecuzione a una funzione esistente, segui i passaggi indicati in the section called “Aggiornamento del ruolo di esecuzione di una funzione”.

Creazione e gestione dei ruoli con AWS CLI

Per creare un ruolo di esecuzione con AWS Command Line Interface (AWS CLI), utilizzate il create-role comando. Quando utilizzi questo comando, puoi specificare la policy di attendibilità in linea. La policy di attendibilità di un ruolo garantisce ai principali specificati l'autorizzazione per assumere tale ruolo. Nell'esempio seguente concedi al principale del servizio Lambda l'autorizzazione per assumere il ruolo. I requisiti per l'escape delle virgolette nella stringa JSON variano in base alla shell.

```
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"}]}'
```

Puoi inoltre definire la policy di attendibilità per il ruolo utilizzando un file JSON separato. Nell'esempio seguente, `trust-policy.json` è un file che si trova nella directory attuale.

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
```

Utilizza il comando attach-policy-to-role per aggiungere le autorizzazioni al ruolo. Il seguente comando aggiunge la policy gestita da `AWSLambdaBasicExecutionRole` al ruolo di esecuzione `lambda-ex`.

```
aws iam attach-role-policy --role-name lambda-ex --policy-arn arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole
```

Dopo aver creato il ruolo di esecuzione, collegalo alla funzione. Quando si crea una funzione nella console Lambda, potrai collegare alla funzione qualsiasi ruolo di esecuzione creato in precedenza. Se desideri associare un nuovo ruolo di esecuzione a una funzione esistente, segui i passaggi indicati in the section called “Aggiornamento del ruolo di esecuzione di una funzione”.

Garantisci l'accesso minimo ai privilegi per il tuo ruolo di esecuzione Lambda

Quando si crea per la prima volta un ruolo IAM per la funzione Lambda durante la fase di sviluppo, a volte è possibile concedere altre autorizzazioni oltre a quanto richiesto. Prima di pubblicare la funzione nell'ambiente di produzione, una best practice consiste nel modificare la policy in modo da includere solo le autorizzazioni richieste. Per ulteriori informazioni, consulta Applicazione delle autorizzazioni del privilegio minimo nella Guida per l'utente di IAM.

Utilizzare IAM Access Analyzer per identificare le autorizzazioni necessarie per la policy del ruolo di esecuzione IAM. IAM Access Analyzer esamina i tuoi AWS CloudTrail log nell'intervallo di date specificato e genera un modello di policy con solo le autorizzazioni utilizzate dalla funzione in quel periodo. È possibile utilizzare il modello per creare una policy gestita con autorizzazioni granulari e quindi collegarla al ruolo IAM. In questo modo, concedi solo le autorizzazioni necessarie al ruolo per interagire con le AWS risorse per il tuo caso d'uso specifico.

Per ulteriori informazioni, consulta Generazione di policy basate sull'attività di accesso nella Guida per l'utente di IAM.

Visualizzazione e aggiornamento delle autorizzazioni nel ruolo di esecuzione

Questo argomento spiega come visualizzare e aggiornare il ruolo di esecuzione della funzione.

Argomenti

- Visualizzazione del ruolo di esecuzione di una funzione
- Aggiornamento del ruolo di esecuzione di una funzione

Visualizzazione del ruolo di esecuzione di una funzione

Per visualizzare il ruolo di esecuzione di una funzione, usa la console Lambda.

Per visualizzare il ruolo di esecuzione di una funzione (console)

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Scegli Configuration (Configurazione), quindi Permissions (Autorizzazioni).

4. In Ruolo di esecuzione, puoi visualizzare il ruolo attualmente utilizzato come ruolo di esecuzione della funzione. Per comodità, puoi visualizzare tutte le risorse e le azioni a cui la funzione può accedere nella sezione Riepilogo delle risorse. Puoi anche scegliere un servizio dal menu a discesa per visualizzare tutte le autorizzazioni relative a tale servizio.

Aggiornamento del ruolo di esecuzione di una funzione

È possibile aggiungere o rimuovere le autorizzazioni da un ruolo di esecuzione della funzione in qualsiasi momento, oppure configurare la funzione per l'utilizzo di un altro ruolo. Se la funzione richiede l'accesso a qualsiasi altro servizio o risorsa, è necessario aggiungere le autorizzazioni necessarie al ruolo di esecuzione.

Quando aggiungi le autorizzazioni alla funzione, aggiorni anche il relativo codice o la configurazione. Questa operazione forza l'arresto e la sostituzione delle istanze della funzione in esecuzione che hanno credenziali obsolete.

Per aggiornare il ruolo di esecuzione di una funzione, usa la console Lambda.

Per aggiornare il ruolo di esecuzione della funzione (console)

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Scegliere il nome della funzione.
3. Scegli Configuration (Configurazione), quindi Permissions (Autorizzazioni).
4. In Ruolo di esecuzione, scegli Modifica.
5. Se desideri aggiornare la tua funzione in modo da utilizzare un ruolo diverso come ruolo di esecuzione, scegli il nuovo ruolo nel menu a discesa sotto Ruolo esistente.

   > **Note**
   >
   > Se desideri aggiornare le autorizzazioni all'interno di un ruolo di esecuzione esistente, puoi farlo solo nella console AWS Identity and Access Management (IAM).

   Se desideri creare un nuovo ruolo da utilizzare come ruolo di esecuzione, scegli Crea un nuovo ruolo dai modelli di AWS policy in Ruolo di esecuzione. Quindi, inserisci un nome per il tuo nuovo ruolo in Nome ruolo e specifica le politicy che desideri collegare al nuovo ruolo in Modelli di policy.

6. Scegli Save (Salva).

Utilizzo delle politiche AWS gestite nel ruolo di esecuzione

Le seguenti politiche AWS gestite forniscono le autorizzazioni necessarie per utilizzare le funzionalità Lambda.

Modifica	Descrizione	Data
AWSLambdaMSKExecutionRuolo: Lambda ha aggiunto l'autorizzazione kafka:DescribeCluster V2 a questa policy.	`AWSLambdaMSKExecutionRole` concede le autorizzazioni per leggere e accedere ai record da un cluster Amazon Managed Streaming for Apache Kafka (Amazon MSK), gestire interfacce di ENIs rete elastiche () e scrivere nei log. CloudWatch	17 giugno 2022
AWSLambdaBasicExecutionRole— Lambda ha iniziato a tracciare le modifiche a questa politica.	`AWSLambdaBasicExecutionRole` concede le autorizzazioni per caricare i log su CloudWatch.	14 febbraio 2022
AWSLambdaDynamoDBExecution Role — Lambda ha iniziato a tenere traccia delle modifiche a questa politica.	`AWSLambdaDynamoDBExecutionRole` concede le autorizzazioni per leggere i record da un flusso Amazon DynamoDB e scrivere nei log. CloudWatch	14 febbraio 2022
AWSLambdaKinesisExecutionRole— Lambda ha iniziato a tracciare le modifiche a questa politica.	`AWSLambdaKinesisExecutionRole` concede le autorizzazioni per leggere eventi da un flusso di dati	14 febbraio 2022

Modifica	Descrizione	Data
	Amazon Kinesis e scrivere su Logs. CloudWatch	
AWSLambdaMSKExecutionRuolo: Lambda ha iniziato a tenere traccia delle modifiche a questa politica.	`AWSLambdaMSKExecutionRole` concede le autorizzazioni per leggere e accedere ai record da un cluster Amazon Managed Streaming for Apache Kafka (Amazon MSK), gestire interfacce di ENIs rete elastiche () e scrivere nei log. CloudWatch	14 febbraio 2022
AWSLambdaSQSQueueExecutionRole— Lambda ha iniziato a tracciare le modifiche a questa politica.	`AWSLambdaSQSQueueExecutionRole` concede le autorizzazioni per leggere un messaggio da una coda Amazon Simple Queue Service (Amazon SQS) e scrivere nei log. CloudWatch	14 febbraio 2022
AWSLambdaVPCAccessExecutionRole— Lambda ha iniziato a tracciare le modifiche a questa politica.	`AWSLambdaVPCAccessExecutionRole` concede le autorizzazioni per la gestione all' ENIs interno di un Amazon VPC e la scrittura nei log. CloudWatch	14 febbraio 2022
AWSXRayDaemonWriteAccess— Lambda ha iniziato a tracciare le modifiche a questa politica.	`AWSXRayDaemonWriteAccess` concede le autorizzazioni per caricare i dati non elaborati su X-Ray.	14 febbraio 2022

Modifica	Descrizione	Data
CloudWatchLambdaInsightsExecutionRolePolicy— Lambda ha iniziato a tracciare le modifiche a questa politica.	`CloudWatchLambdaInsightsExecutionRolePolicy` concede le autorizzazioni per scrivere metriche di runtime a CloudWatch Lambda Insights.	14 febbraio 2022
AmazonS3 — ObjectLambdaExecutionRolePolicy Lambda ha iniziato a tracciare le modifiche a questa politica.	`AmazonS3ObjectLambdaExecutionRolePolicy` concede le autorizzazioni per interagire con l'oggetto Lambda di Amazon Simple Storage Service (Amazon S3) e per scrivere su Logs. CloudWatch	14 febbraio 2022

Per alcune funzionalità, la console Lambda tenta di aggiungere autorizzazioni mancanti al ruolo di esecuzione in una policy gestita dal cliente. Queste policy possono diventare numerose. Aggiungere le policy gestite AWS pertinenti al ruolo di esecuzione prima di abilitare le funzionalità per evitare la creazione di policy aggiuntive.

Quando si utilizza una mappatura origine eventi per invocare la funzione, Lambda utilizza il ruolo di esecuzione per leggere i dati dell'evento. Ad esempio, la mappatura dell'origine eventi per Kinesis legge gli eventi provenienti da un flusso di dati e li invia alla funzione in batch.

Quando un servizio assume un ruolo nel tuo account, puoi includere le chiavi di contesto delle condizioni globali `aws:SourceAccount` e `aws:SourceArn` nella policy di attendibilità del ruolo per limitare l'accesso al ruolo solo alle richieste generate dalle risorse previste. Per ulteriori informazioni consulta Prevenzione del problema "confused deputy" tra servizi per AWS Security Token Service.

Oltre alle policy AWS gestite, la console Lambda fornisce modelli per la creazione di policy personalizzate con autorizzazioni per casi d'uso aggiuntivi. Quando si crea una funzione nella console Lambda, è possibile scegliere di creare un nuovo ruolo di esecuzione con le autorizzazioni da uno o più modelli. Questi modelli vengono applicati automaticamente al momento della creazione

di una funzione da un blueprint, oppure quando si configurano le opzioni che richiedono l'accesso ad altri servizi. Modelli di esempio sono disponibili nell'archivio di questa guida. GitHub

Utilizzo dell'ARN della funzione di origine per controllare il comportamento di accesso alla funzione

È normale che il codice della funzione Lambda effettui richieste API ad altri Servizi AWS. Per effettuare queste richieste, Lambda genera un insieme effimero di credenziali assumendo il ruolo di esecuzione della tua funzione. Queste credenziali sono disponibili come variabili d'ambiente durante l'invocazione della funzione. Quando si lavora con AWS SDKs, non è necessario fornire le credenziali per l'SDK direttamente nel codice. Per impostazione predefinita, la catena di fornitori di credenziali controlla in sequenza ogni punto in cui è possibile impostare le credenziali e seleziona la prima disponibile, in genere le variabili d'ambiente (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` e `AWS_SESSION_TOKEN`).

Lambda inserisce la funzione di origine ARN nel contesto delle credenziali se la richiesta è una richiesta AWS API proveniente dall'ambiente di esecuzione. Lambda inserisce anche l'ARN della funzione di origine per le seguenti richieste API AWS che Lambda effettua per tuo conto al di fuori dell'ambiente di esecuzione:

Servizio	Azione	Motivo
CloudWatch Registri	`CreateLogGroup` , `CreateLogStream` , `PutLogEvents`	Per archiviare i log in un gruppo di CloudWatch log Logs
X-Ray	`PutTraceSegments`	Invio dei dati di traccia a X-Ray
Amazon EFS	`ClientMount`	Connessione di una funzione a un file system Amazon Elastic File System (Amazon EFS)

Le altre chiamate AWS API che Lambda effettua al di fuori dell'ambiente di esecuzione per tuo conto utilizzando lo stesso ruolo di esecuzione non contengono la funzione di origine ARN. Esempi di tali chiamate API al di fuori dell'ambiente di esecuzione includono:

- Chiama a AWS Key Management Service (AWS KMS) per crittografare e decrittografare automaticamente le variabili di ambiente.
- Chiamate ad Amazon Elastic Compute Cloud (Amazon EC2) per creare interfacce di rete elastiche (ENIs) per una funzione abilitata al VPC.
- Chiamate a Servizi AWS, come Amazon Simple Queue Service (Amazon SQS), per la lettura da un'origine di eventi configurata come mappatura delle sorgenti di eventi.

Con l'ARN della funzione di origine nel contesto delle credenziali puoi verificare se una chiamata alla tua risorsa proviene dal codice di una funzione Lambda specifica. Per verificarlo, usa la chiave di condizione `lambda:SourceFunctionArn` in una policy basata sulle identità IAM o una policy di controllo dei servizi (SCP).

> **Note**
>
> Non puoi utilizzare la chiave di condizione `lambda:SourceFunctionArn` nelle policy basate sulle risorse.

Implementando questa condizione nelle tue policy basate sull'identità SCPs, puoi oppure implementare controlli di sicurezza per le azioni API che il tuo codice funzionale esegue su altri. Servizi AWS Questo ha alcune funzioni di sicurezza chiave, come aiutarti a identificare l'origine di una perdita di credenziali.

> **Note**
>
> La chiave di condizione `lambda:SourceFunctionArn` è diversa dalle chiavi di condizione `lambda:FunctionArn` e `aws:SourceArn`. La chiave di condizione `lambda:FunctionArn` si applica solo a mappature di origine eventi e aiuta a definire quali funzioni può richiamare l'origine eventi. La chiave di `aws:SourceArn` condizione si applica solo alle politiche in cui la funzione Lambda è la risorsa di destinazione e aiuta a definire quali altre Servizi AWS risorse possono richiamare quella funzione. La chiave di `lambda:SourceFunctionArn` condizione può essere applicata a qualsiasi policy o SCP basata sull'identità per definire le funzioni Lambda specifiche che dispongono delle autorizzazioni per effettuare chiamate API specifiche ad altre risorse. AWS

Per utilizzare `lambda:SourceFunctionArn` nella tua policy, includila come condizione con uno qualsiasi degli operatori di condizione ARN. Il valore della chiave deve essere un ARN valido.

Ad esempio, supponiamo che il codice della funzione Lambda effettui una chiamata `s3:PutObject` che si rivolge a un determinato bucket Amazon S3. Potresti volere che solo una funzione Lambda specifica abbia accesso `s3:PutObject` a quel bucket. In questo caso, il ruolo di esecuzione della funzione deve avere una policy collegata simile alla seguente:

Example policy che concede l'accesso a una funzione Lambda specifica a una risorsa 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"
                }
            }
        }
    ]
}
```

Questa policy consente l'accesso `s3:PutObject` solo se l'origine è la funzione Lambda con ARN `arn:aws:lambda:us-east-1:123456789012:function:source_lambda`. La policy non consente l'accesso `s3:PutObject` ad alcun'altra identità chiamante. Ciò è vero anche se una funzione o entità diversa effettua una chiamata `s3:PutObject` con lo stesso ruolo di esecuzione.

> **Note**
>
> La chiave di condizione `lambda:SourceFunctionARN` non supporta gli alias delle funzioni o le versioni delle funzioni Lambda. Se utilizzi l'ARN per una particolare versione o alias di una funzione, la tua funzione non avrà l'autorizzazione per eseguire l'azione specificata.

Assicurarsi di utilizzare l'ARN non completo per la funzione senza un suffisso di versione o alias.

Puoi anche usare in. `lambda:SourceFunctionArn` SCPs Ad esempio, supponiamo di voler limitare l'accesso al bucket al codice di una singola funzione Lambda o a chiamate da un cloud privato virtuale (VPC) Amazon specifico. L'SCP seguente illustra questo scenario.

Example politica che nega l'accesso ad Amazon S3 in condizioni specifiche

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"
                }
            }
        }
```

```
    ]
}
```

Questa policy nega tutte le azioni S3 a meno che non provengano da una specifica funzione Lambda con ARN `arn:aws:lambda:*:123456789012:function:source_lambda` o a meno che non provengano dal VPC specificato. L'operatore `StringNotEqualsIfExists` dice a IAM di elaborare questa condizione solo se la chiave `aws:SourceVpc` è presente nella richiesta. Allo stesso modo, IAM considera l'operatore `ArnNotEqualsIfExists` solo se esiste `lambda:SourceFunctionArn`.

Concedere ad altre AWS entità l'accesso alle tue funzioni Lambda

Per concedere Account AWS ad altre organizzazioni e servizi le autorizzazioni per accedere alle tue risorse Lambda, hai alcune opzioni:

- È possibile utilizzare le policy basate sull'identità per concedere agli utenti l'accesso alle risorse Lambda. Le policy basate su identità possono essere applicate direttamente agli utenti o ai gruppi e ruoli associati a un utente.
- Puoi utilizzare policy basate sulle risorse per concedere ad altri account e Servizi AWS autorizzazioni per accedere alle tue risorse Lambda. Quando un utente prova ad accedere a una risorsa Lambda, Lambda considera sia le policy basate su identità dell'utente che la policy basata sulla risorsa della risorsa. Quando un AWS servizio come Amazon Simple Storage Service (Amazon S3) richiama la funzione Lambda, Lambda considera solo la politica basata sulle risorse.
- Puoi utilizzare un modello controllo degli accessi basato su attributi (ABAC), per controllare l'accesso alle funzioni Lambda. Con ABAC, puoi collegare i tag a una funzione Lambda, inviarli in determinate richieste API o collegarli al principale IAM da cui proviene la richiesta. Per controllare gli accessi alla funzione, puoi invece specificare gli stessi tag nell'elemento di condizione di una policy IAM.

Per aiutarti a ottimizzare le autorizzazioni per l'accesso con privilegi minimi, Lambda fornisce alcune condizioni aggiuntive che puoi includere nelle tue policy. Per ulteriori informazioni, consulta the section called “Risorse e condizioni”.

Policy IAM basate sull'identità per Lambda

Puoi utilizzare le policy basate sull'identità in AWS Identity and Access Management (IAM) per concedere agli utenti del tuo account l'accesso a Lambda. Le policy basate sulle identità possono essere applicate a utenti, gruppi di utenti o ruoli. È inoltre possibile concedere le autorizzazioni a utenti in un altro account in modo che assumano un ruolo nel proprio account ed eseguano l'accesso alle risorse Lambda.

Lambda fornisce policy AWS gestite che garantiscono l'accesso alle azioni dell'API Lambda e, in alcuni casi, l'accesso ad altri AWS servizi utilizzati per sviluppare e gestire le risorse Lambda. Lambda aggiorna le policy gestite in base alle necessità, per assicurare che gli utenti possano accedere a nuove caratteristiche quando queste vengono rilasciate.

- AWSLambda_FullAccess— Garantisce l'accesso completo alle azioni Lambda e AWS ad altri servizi utilizzati per sviluppare e gestire le risorse Lambda.
- AWSLambda_ReadOnlyAccess— Garantisce l'accesso in sola lettura alle risorse Lambda.
- AWSLambdaRuolo: concede le autorizzazioni per richiamare le funzioni Lambda.

AWS le politiche gestite concedono l'autorizzazione alle azioni API senza limitare le funzioni o i livelli Lambda che un utente può modificare. Per un controllo ancora più accurato, è possibile creare le proprie policy che limitano l'ambito di applicazione delle autorizzazioni di un utente.

Argomenti

- Concedere agli utenti l'accesso a una funzione Lambda
- Concedere agli utenti l'accesso a un livello Lambda

Concedere agli utenti l'accesso a una funzione Lambda

Utilizza le policy basate sull'identità per consentire a utenti, gruppi di utenti o ruoli di eseguire operazioni sulle funzioni Lambda.

> **Note**
>
> Per una funzione definita come immagine di container, l'autorizzazione utente per accedere all'immagine deve essere configurata in Amazon Elastic Container Registry (Amazon ECR). Per un esempio, consulta le policy del repository di Amazon ECR.

Di seguito viene illustrato un esempio di policy di autorizzazione con ambito di applicazione limitato. Ciò consente a un utente di creare e gestire le funzioni Lambda denominate con un prefisso designato (`intern-`) e configurate con un ruolo di esecuzione designato.

Example Policy di sviluppo delle funzioni

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-*"
        }
    ]
}
```

Le autorizzazioni nella policy sono organizzate in dichiarazioni in base alle risorse e le condizioni che supportano.

- `ReadOnlyPermissions` – La console Lambda utilizza queste autorizzazioni quando l'utente sfoglia e visualizza le funzioni. Esse non supportano i modelli di risorse o condizioni.

```
        "Action": [
            "lambda:GetAccountSettings",
            "lambda:GetEventSourceMapping",
            "lambda:GetFunction",
            "lambda:GetFunctionConfiguration",
            "lambda:GetFunctionCodeSigningConfig",
            "lambda:GetFunctionConcurrency",
            "lambda:ListEventSourceMappings",
            "lambda:ListFunctions",
            "lambda:ListTags",
            "iam:ListRoles"
        ],
        "Resource": "*"
```

- `DevelopFunctions`: utilizza qualsiasi operazione che opera su funzioni con prefisso `intern-`, eccetto `AddPermission` e `PutFunctionConcurrency`. `AddPermission` modifica la policy basata sulle risorse sulla funzione e può avere implicazioni in termini di sicurezza. `PutFunctionConcurrency` riserva la capacità di dimensionamento per una funzione e può prendere la capacità da altre funzioni.

```
        "NotAction": [
            "lambda:AddPermission",
            "lambda:PutFunctionConcurrency"
        ],
        "Resource": "arn:aws:lambda:*:*:function:intern-*"
```

- `DevelopEventSourceMappings`: gestisci le mappature di origine eventi sulle funzioni con prefisso `intern-`. Queste operazioni operano su mappature di origine eventi, ma è possibile limitarle per funzione con una condizione.

```
        "Action": [
            "lambda:DeleteEventSourceMapping",
            "lambda:UpdateEventSourceMapping",
            "lambda:CreateEventSourceMapping"
        ],
        "Resource": "*",
        "Condition": {
            "StringLike": {
                "lambda:FunctionArn": "arn:aws:lambda:*:*:function:intern-*"
            }
        }
```

- `PassExecutionRole` – Visualizzare e trasferire solo un ruolo denominato `intern-lambda-execution-role`, che deve essere creato e gestito da un utente con le autorizzazioni IAM. `PassRole` viene utilizzato quando si assegna un ruolo di esecuzione a una funzione.

```
        "Action": [
            "iam:ListRolePolicies",
            "iam:ListAttachedRolePolicies",
            "iam:GetRole",
            "iam:GetRolePolicy",
            "iam:PassRole",
```

```
            "iam:SimulatePrincipalPolicy"
        ],
        "Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
```

- `ViewLogs`— Utilizzate CloudWatch Logs per visualizzare i log delle funzioni il cui prefisso è. `intern-`

```
        "Action": [
            "logs:*"
        ],
        "Resource": "arn:aws:logs:*:*:log-group:/aws/lambda/intern-*"
```

Questa policy consente a un utente di iniziare con Lambda, senza mettere le risorse di altri utenti a rischio. Non consente a un utente di configurare una funzione da attivare o chiamare altri AWS servizi, il che richiede autorizzazioni IAM più ampie. Inoltre, non include l'autorizzazione a servizi che non supportano politiche ad ambito limitato, come CloudWatch X-Ray. Utilizzare le policy di sola lettura per questi servizi per fornire all'utente l'accesso ai parametri e ai dati di traccia.

Quando configuri i trigger per la tua funzione, devi accedere per utilizzare il AWS servizio che richiama la tua funzione. Ad esempio, per configurare un trigger Amazon S3, è necessario disporre dell'autorizzazione per utilizzare le operazioni Amazon S3 che gestiscono le notifiche dei bucket. Molte di queste autorizzazioni sono incluse nella politica gestita. AWSLambda_FullAccess

Concedere agli utenti l'accesso a un livello Lambda

Utilizza le policy basate sull'identità per consentire a utenti, gruppi di utenti o ruoli di eseguire operazioni sui livelli Lambda. La seguente policy concede un'autorizzazione utente per creare i livelli e utilizzarli con le funzioni. I modelli di risorse consentono all'utente di lavorare in qualsiasi Regione AWS versione del layer, a condizione che il nome del layer inizi con `test-`.

Example Policy per lo sviluppo dei livelli

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-*:*"
        }
    ]
}
```

È anche possibile implementare l'utilizzo di livelli durante la creazione e la configurazione della funzione con la condizione `lambda:Layer`. Ad esempio, è possibile evitare che gli utenti utilizzino livelli pubblicati da altri account. La policy seguente aggiunge una condizione per le operazioni `CreateFunction` e `UpdateFunctionConfiguration` per richiedere che i livelli specificati provengano dall'account `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:*:*"
                ]
            }
        }
    }
]
}
```

Per garantire che la condizione venga applicata, verificare che nessun'altra dichiarazioni conceda l'autorizzazione all'utente per tali azioni.

Visualizzazione delle policy IAM basate sulle risorse in Lambda

Lambda supporta le policy di autorizzazioni basate sulle risorse per le funzioni e i livelli Lambda. Puoi utilizzare le policy basate sulle risorse per concedere l'accesso ad altri account AWS, organizzazioni o servizi. Le policy basate sulle risorse si applicano a una singola funzione, versione, alias o versione del livello.

Console

Per visualizzare le policy basate sulle risorse di una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Scegliere Configuration (Configurazione), quindi Permissions (Autorizzazioni).
4. Scorrere verso il basso fino a Policy basata su risorse, quindi scegliere View Policy Document (Visualizza documento policy). La politica basata sulle risorse mostra le autorizzazioni applicate quando un altro account o servizio tenta di accedere alla funzione. AWS L'esempio seguente mostra un'istruzione che consente ad Amazon S3 di richiamare una funzione denominata `my-function` per un bucket denominato `amzn-s3-demo-bucket` nell'account `123456789012`.

   Example policy basata su risorse

```
{
    "Version":"2012-10-17",
    "Id": "default",
    "Statement": [
```

```
        {
            "Sid": "lambda-allow-s3-my-function",
            "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:::amzn-s3-demo-bucket"
              }
            }
        }
    ]
}
```

AWS CLI

Per visualizzare una policy basata sulle risorse di una funzione, utilizzare il comando `get-policy`.

```
aws lambda get-policy \
  --function-name my-function \
  --output text
```

Verrà visualizzato l'output seguente:

```
{"Version":"2012-10-17",       "Id":"default","Statement":
[{"Sid":"sns","Effect":"Allow","Principal":
{"Service":"s3.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambd
east-2:123456789012:function:my-function","Condition":{"ArnLike":
{"AWS:SourceArn":"arn:aws:sns:us-east-2:123456789012:lambda*"}}}]}
```

Per le versioni e gli alias, aggiungere il numero di versione o un alias al nome della funzione.

```
aws lambda get-policy --function-name my-function:PROD
```

Per rimuovere le autorizzazioni dalla funzione, utilizzare `remove-permission`.

```
aws lambda remove-permission \
  --function-name example \
  --statement-id sns
```

Utilizzare il comando `get-layer-version-policy` per visualizzare le autorizzazioni su un layer.

```
aws lambda get-layer-version-policy \
  --layer-name my-layer \
  --version-number 3 \
  --output text
```

Verrà visualizzato l'output seguente:

```
b0cd9796-d4eb-4564-939f-de7fe0b42236    {"Sid":"engineering-
org","Effect":"Allow","Principal":"*","Action":"lambda:GetLayerVersion","Resource":"arn:aws:
west-2:123456789012:layer:my-layer:3","Condition":{"StringEquals":
{"aws:PrincipalOrgID":"o-t194hfs8cz"}}}"
```

Utilizzare `remove-layer-version-permission` per rimuovere le istruzioni dalla policy.

```
aws lambda remove-layer-version-permission --layer-name my-layer --version-number 3
 --statement-id engineering-org
```

Operazioni API supportate

Le seguenti operazioni dell'API Lambda supportano le policy basate sulle risorse:

- CreateAlias
- DeleteAlias
- DeleteFunction
- DeleteFunctionConcurrency
- DeleteFunctionEventInvokeConfig

- DeleteProvisionedConcurrencyConfig
- GetAlias
- GetFunction
- GetFunctionConcurrency
- GetFunctionConfiguration
- GetFunctionEventInvokeConfig
- GetPolicy
- GetProvisionedConcurrencyConfig
- Invoke
- InvokeFunctionUrl(solo autorizzazione)
- ListAliases
- ListFunctionEventInvokeConfigs
- ListProvisionedConcurrencyConfigs
- ListTags
- ListVersionsByFunction
- PublishVersion
- PutFunctionConcurrency
- PutFunctionEventInvokeConfig
- PutProvisionedConcurrencyConfig
- TagResource
- UntagResource
- UpdateAlias
- UpdateFunctionCode
- UpdateFunctionEventInvokeConfig

Concessione dell'accesso alla funzione Lambda a Servizi AWS

Quando utilizzi un AWS servizio per richiamare la tua funzione, concedi l'autorizzazione in una dichiarazione su una politica basata sulle risorse. È possibile applicare l'istruzione all'intera funzione oppure limitare l'istruzione a una singola versione o alias.

> **Note**
>
> Quando si aggiunge un trigger a una funzione con la console Lambda, la console aggiorna la policy basata sulle risorse della funzione per consentire al servizio di invocarla. Per concedere le autorizzazioni ad altri account o servizi che non sono disponibili nella console Lambda, puoi utilizzare l'interfaccia a riga di comando di AWS CLI.

Aggiungi una istruzione con il comando add-permission. La dichiarazione più semplice della policy basata sulla risorsa consente a un servizio di invocare una funzione. Il comando seguente concede ad Amazon Simple Notification Service l'autorizzazione per richiamare una funzione denominata `my-function`.

```
aws lambda add-permission \
  --function-name my-function \
  --action lambda:InvokeFunction \
  --statement-id sns \
  --principal sns.amazonaws.com \
  --output text
```

Verrà visualizzato l'output seguente:

```
{"Sid":"sns","Effect":"Allow","Principal":
{"Service":"sns.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-
east-2:123456789012:function:my-function"}
```

In questo modo Amazon SNS richiama l'operazione API Invoke per la funzione, ma non limita l'argomento Amazon SNS che attiva la chiamata. Per garantire che la tua funzione venga richiamata solo da una risorsa specifica, specificare l'Amazon Resource Name (ARN) della risorsa con l'opzione `source-arn`. Il comando seguente consente solo a Amazon SNS di richiamare la funzione per gli abbonamenti a un argomento denominato `my-topic`.

```
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
```

Alcuni servizi possono invocare le funzioni in altri account. Se si specifica un ARN origine che ha il proprio ID account, non è un problema. Per Amazon S3, tuttavia, l'origine è un bucket il cui ARN non dispone di un ID account. È possibile eliminare il bucket e un altro account può creare un bucket con lo stesso nome. Utilizza l'opzione `source-account` con il tuo ID account per avere la certezza che solo le risorse nel proprio account possano richiamare la funzione.

```
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
```

Concedere l'accesso alle funzioni a un'organizzazione

Per concedere un'autorizzazione a un'organizzazione in AWS Organizations, specifica l'ID dell'organizzazione come `principal-org-id`. Il seguente comando add-permission garantisce l'accesso all'invocazione a tutti gli utenti dell'organizzazione `o-a1b2c3d4e5f`.

```
aws lambda add-permission \
  --function-name example \
  --statement-id PrincipalOrgIDExample \
  --action lambda:InvokeFunction \
  --principal * \
  --principal-org-id o-a1b2c3d4e5f
```

> **Note**
>
> In questo comando, `Principal` è `*`. Ciò significa che tutti gli utenti dell'organizzazione `o-a1b2c3d4e5f` ottengono le autorizzazioni di invocazione delle funzioni. Se specifichi un ruolo Account AWS o come`Principal`, solo quel principale ottiene i permessi di invocazione della funzione, ma solo se fa anche parte dell'organizzazione. `o-a1b2c3d4e5f`

Questo comando crea una policy basata sulle risorse simile alla seguente:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "PrincipalOrgIDExample",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-east-2:123456789012:function:example",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-a1b2c3d4e5f"
                }
            }
        }
    ]
}
```

Per ulteriori informazioni, consulta aws: PrincipalOrg ID nella guida per l'utente IAM.

Concessione dell'accesso alla funziona Lambda ad altri account

Per condividere una funzione con un'altra Account AWS, aggiungi una dichiarazione di autorizzazione per più account alla politica basata sulle risorse della funzione. Esegui il comando add-permission e specifica l'ID account come `principal`. L'esempio seguente concede l'autorizzazione `111122223333` a un account per invocare `my-function` con l'alias `prod`.

```
aws lambda add-permission \
  --function-name my-function:prod \
  --statement-id xaccount \
  --action lambda:InvokeFunction \
  --principal 111122223333 \
  --output text
```

Verrà visualizzato l'output seguente:

```
{"Sid":"xaccount","Effect":"Allow","Principal":
{"AWS":"arn:aws:iam::111122223333:root"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:l
east-1:123456789012:function:my-function"}
```

La policy basata sulle risorse concede l'autorizzazione all'altro account per accedere alla funzione, ma non consente agli utenti in tale account di superare le autorizzazioni. Gli utenti nell'altro account devono disporre delle autorizzazioni utente corrispondenti per utilizzare l'API Lambda.

Per limitare l'accesso a un utente, un gruppo o un ruolo in un altro account, specificare l'ARN completo dell'identità come principale. Ad esempio, `arn:aws:iam::123456789012:user/developer`.

L'alias limita quale versione l'altro account può invocare. Richiede che l'altro account includa l'alias nell'ARN della funzione.

```
aws lambda invoke \
  --function-name arn:aws:lambda:us-east-2:123456789012:function:my-function:prod out
```

Verrà visualizzato l'output seguente:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "1"
}
```

Il proprietario della funzione può quindi aggiornare l'alias per puntare a una nuova versione senza che il chiamante debba cambiare il modo in cui invoca la funzione. In questo modo l'altro account non deve modificare il codice per utilizzare la nuova versione e dispone solo dell'autorizzazione per richiamare la versione della funzione associata all'alias.

È possibile concedere l'accesso tra account per qualsiasi operazione API che opera su una funzione esistente. Ad esempio, è possibile concedere l'accesso a un account `lambda:ListAliases` per ottenere un elenco di alias, oppure `lambda:GetFunction` per scaricare il codice funzione. Aggiungere ogni autorizzazione separatamente oppure utilizzare `lambda:*` per concedere l'accesso a tutte le azioni per la funzione specificata.

Per concedere ad altri account l'autorizzazione per più funzioni o per operazioni che non operano su una funzione, utilizzare ruoli IAM.

Concessione dell'accesso ai livelli Lambda ad altri account

Per condividere un layer con un altro Account AWS, aggiungi una dichiarazione di autorizzazione per più account alla policy basata sulle risorse del layer. Esegui il add-layer-version-permissioncomando e specifica l'ID dell'account come. `principal` In ogni istruzione, puoi concedere l'autorizzazione a un singolo account, a tutti gli account o a un'organizzazione in AWS Organizations.

L'esempio seguente concede all'account 111122223333 l'accesso alla versione 2 del livello `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
```

Verrà visualizzato un output simile al seguente:

```
{"Sid":"xaccount","Effect":"Allow","Principal":
{"AWS":"arn:aws:iam::111122223333:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:
east-1:123456789012:layer:bash-runtime:2"}
```

Le autorizzazioni si applicano solo a un'unica versione di un livello. Ripeti la procedura ogni volta che crei la nuova versione di un livello.

Per concedere le autorizzazioni a tutti gli account in un'organizzazione AWS Organizations, è possibile utilizzare l'opzione `organization-id`. L'esempio seguente concede a tutti gli account dell'organizzazione l'autorizzazione `o-t194hfs8cz` per utilizzare la versione 3 di `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
```

Verrà visualizzato l’output seguente:

```
{"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"}}}"
```

Per concedere l'autorizzazione a più account o organizzazioni, devi aggiungere più istruzioni.

Utilizzo del controllo degli accessi basato sugli attributi in Lambda

Con il controllo degli accessi basato su attributi (ABAC), puoi usare i tag per controllare l'accesso alle risorse Lambda. Puoi allegare tag a determinate risorse Lambda, allegarli a determinate richieste API o collegarli al principale AWS Identity and Access Management (IAM) che effettua la richiesta. Per ulteriori informazioni su come AWS concedere l'accesso basato sugli attributi, consulta Controlling access to AWS resources using tags nella IAM User Guide.

Puoi utilizzare ABAC per concedere il privilegio minimo senza specificare un nome della risorsa Amazon (ARN) o uno schema ARN nella policy IAM. Per controllare gli accessi puoi invece specificare un tag nell'elemento di condizione di una policy IAM. La scalabilità è più facile con ABAC perché non è necessario aggiornare le policy IAM quando vengono create nuove risorse. Invece, per controllare l'accesso aggiungi i tag alle nuove risorse.

In Lambda, i tag funzionano sulle seguenti risorse:

- Funzioni: per ulteriori informazioni sul tagging delle funzioni, consulta the section called “Tag”.
- Configurazioni di firma del codice: per ulteriori informazioni sul tagging delle configurazioni di firma del codice, consulta the section called “Tag di configurazione della firma del codice”.
- Strumenti di mappatura dell'origine degli eventi: per ulteriori informazioni sul tagging degli strumenti di mappatura dell'origine degli eventi, consulta the section called “Tag dello strumento di mappatura dell'origine degli eventi”.

I tag non sono supportati per i livelli.

È possibile utilizzare le seguenti chiavi di condizione per scrivere regole di policy IAM basate sui tag:

- aws: ResourceTag /tag-key: controlla l'accesso in base ai tag allegati a una risorsa Lambda.
- aws: RequestTag /tag-key: richiede che i tag siano presenti in una richiesta, ad esempio quando si crea una nuova funzione.

- aws: PrincipalTag /tag-key : Controlla cosa può fare il principale IAM (la persona che effettua la richiesta) in base ai tag associati al suo utente o ruolo IAM.
- aws:TagKeys: Controlla se è possibile utilizzare chiavi di tag specifiche in una richiesta.

Puoi specificare solo le condizioni per le azioni che le supportano. Per un elenco delle operazioni supportate da ogni operazione Lambda, consulta Operazioni, risorse e chiavi di condizione per AWS Lambda in Service Authorization Reference. Per il supporto di aws: ResourceTag /tag-key, consulta «Tipi di risorse definiti da». AWS Lambda Per aws: RequestTag /tag-key e aws: TagKeys support, consulta «Azioni definite da». AWS Lambda

Argomenti

- Proteggere le funzioni con un tag

Proteggere le funzioni con un tag

La seguente procedura mostra un modo per impostare le autorizzazioni per le funzioni tramite ABAC. In questo scenario di esempio, creerai quattro policy di autorizzazione IAM. Quindi, collegherai queste policy a un nuovo ruolo IAM. Infine, creerai un utente IAM e gli assegnerai l'autorizzazione per assumere il nuovo ruolo.

Argomenti

- Prerequisiti
- Fase 1: richiesta di tag sulle nuove funzioni
- Fase 2: Consentire azioni basate su tag collegati a una funzione Lambda e al principale IAM
- Fase 3: Concessione delle autorizzazioni di elenco
- Fase 4: Concessione delle autorizzazioni IAM
- Fase 5: Creazione del ruolo IAM
- Fase 6: Creazione dell'utente IAM
- Fase 7: Test delle autorizzazioni
- Fase 8: eliminare le risorse

Prerequisiti

Assicurati di avere un ruolo di esecuzione Lambda. Userai questo ruolo quando concedi le autorizzazioni IAM e crei una funzione Lambda.

Fase 1: richiesta di tag sulle nuove funzioni

Quando utilizzi ABAC con Lambda, è consigliabile richiedere che tutte le funzioni abbiano i tag. Questo aiuta a garantire che le policy di autorizzazione ABAC funzionino come previsto.

Crea una policy IAM simile a quella del seguente esempio. Questa policy utilizza le chiavi aws: RequestTag /tag-key, aws: ResourceTag /tag-key e aws: TagKeys condition per richiedere che le nuove funzioni e il principale IAM che le crea abbiano entrambi il tag. `project` Il modificatore `ForAllValues` assicura che `project` sia l'unico tag consentito. Se non includi il modificatore `ForAllValues`, gli utenti potranno aggiungere altri tag alla funzione purché passino anche loro `project`.

Example : richiesta di tag sulle nuove funzioni

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"
        }
      }
    }
  }
```

Fase 2: Consentire azioni basate su tag collegati a una funzione Lambda e al principale IAM

Crea una seconda policy IAM utilizzando la chiave di condizione aws: ResourceTag /tag-key per richiedere che il tag del principale corrisponda al tag associato alla funzione. La seguente policy di

esempio consente ai principali con tag `project` di richiamare le funzioni con il tag `project`. Se una funzione ha altri tag, l'azione è negata.

Example : Richiesta di tag corrispondenti sulla funzione e sul principale 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}"
          }
        }
      }
    ]
  }
```

Fase 3: Concessione delle autorizzazioni di elenco

Crea una policy che consenta al principale di elencare le funzioni Lambda e i ruoli IAM. Ciò consente al principale di vedere tutte le funzioni Lambda e i ruoli IAM sulla console e quando si chiamano le operazioni API.

Example : Concessione delle autorizzazioni di elenco per Lambda e IAM

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
```

```
        "Sid": "AllResourcesLambdaNoTags",
        "Effect": "Allow",
        "Action": [
          "lambda:GetAccountSettings",
          "lambda:ListFunctions",
          "iam:ListRoles"
        ],
        "Resource": "*"
      }
    ]
  }
```

Fase 4: Concessione delle autorizzazioni IAM

Crea una policy che consenta iam:. PassRole Questa autorizzazione è necessaria quando si assegna un ruolo di esecuzione a una funzione. Nella seguente policy di esempio, sostituisci l'ARN di esempio con l'ARN del tuo ruolo di esecuzione Lambda.

> **Note**
>
> Non utilizzare la chiave di condizione `ResourceTag` in una policy con l'operazione `iam:PassRole`. Non puoi utilizzare il tag su un ruolo IAM per controllare l'accesso a chi può passare tale ruolo. Per ulteriori informazioni sulle autorizzazioni necessarie per passare un ruolo a un servizio, vedere Concessione a un utente delle autorizzazioni per passare un ruolo a un servizio. AWS

Example : Concessione dell'autorizzazione per inviare il ruolo di esecuzione

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
      {
        "Sid": "VisualEditor0",
        "Effect": "Allow",
        "Action": [
          "iam:PassRole"
        ],
```

```
        "Resource": "arn:aws:iam::111122223333:role/lambda-ex"
      }
    ]
  }
```

Fase 5: Creazione del ruolo IAM

Una best practice consiste nell'utilizzare i ruoli per delegare le autorizzazioni. Crea un ruolo IAM chiamato `abac-project-role`:

- In Fase 1: Selezione dell'entità attendibile, scegli Account AWS , quindi Questo account.
- In Fase 2: Aggiunta delle autorizzazioni, collega le quattro policy IAM create nei passaggi precedenti.
- In Fase 3: Assegnazione di un nome, revisione e creazione, scegli Add tag (Aggiungi tag). In Chiave, inserire `project`. Non immettere un valore.

Fase 6: Creazione dell'utente IAM

Crea un utente IAM chiamato `abac-test-user`. Nella sezione Set permissions (Imposta autorizzazioni), seleziona Attach existing policies directly (Collega direttamente le policy esistenti), quindi Create policy (Crea policy). Immetti la seguente definizione di policy. Sostituisci *111122223333* con il tuo ID account AWS. Questa policy consente a `abac-test-user` di assumere il ruolo `abac-project-role`.

Example : Consentire all'utente IAM di assumere il ruolo ABAC

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::111122223333:role/abac-project-role"
    }
  }
```

Fase 7: Test delle autorizzazioni

1. Accedi alla console come. AWS `abac-test-user` Per ulteriori informazioni, consulta Accesso come utente IAM.
2. Passare al ruolo `abac-project-role`. Per ulteriori informazioni, consulta Cambio di un ruolo (console).
3. Crea una funzione Lambda:
   - In Permissions (Autorizzazioni), scegli Change default execution role (Modifica ruolo di esecuzione predefinito), quindi per Execution role (Ruolo di esecuzione), scegli Use an existing role (Utilizza un ruolo esistente). Scegli lo stesso ruolo di esecuzione che hai usato in Fase 4: Concessione delle autorizzazioni IAM.
   - In Advanced settings (Impostazioni avanzate), seleziona Enable tags (Abilita tag) quindi scegli Add new tag (Aggiungi nuovo tag). In Chiave, inserire `project`. Non immettere un valore.
4. Esegui il test della funzione.
5. Crea una seconda funzione Lambda e aggiungi un tag diverso, ad esempio `environment`. Questa operazione dovrebbe non riuscire perché la policy ABAC creata in Fase 1: richiesta di tag sulle nuove funzioni consente al principale di creare solo funzioni con tag `project`.
6. Crea una terza funzione senza tag. Questa operazione dovrebbe non riuscire perché la policy ABAC creata in Fase 1: richiesta di tag sulle nuove funzioni non consente al principale di creare funzioni senza tag.

Questa strategia di autorizzazione consente di controllare l'accesso senza creare nuove policy per ogni nuovo utente. Per concedere l'accesso a nuovi utenti, è sufficiente concedere loro l'autorizzazione per assumere il ruolo che corrisponde al progetto assegnato.

Fase 8: eliminare le risorse

Per eliminare il ruolo IAM

1. Aprire la pagina Ruoli della console IAM.
2. Seleziona il ruolo creato nella fase 5.
3. Scegli Elimina.
4. Per confermare l'eliminazione, inserisci il nome del ruolo nel campo di immissione del testo.
5. Scegli Elimina.

Per eliminare l'utente IAM

1. Apri la pagina Utenti nella console IAM.
2. Seleziona l'utente IAM creato nella fase 6.
3. Scegli Elimina.
4. Per confermare l'eliminazione, inserisci il nome utente nel campo di immissione del testo.
5. Scegli Elimina utente.

Per eliminare la funzione Lambda

1. Aprire la pagina Functions (Funzioni) della console Lambda.
2. Selezionare la funzione creata.
3. Scegliere Operazioni, Elimina.
4. Inserisci **`confirm`** nel campo di immissione del testo, quindi scegli Elimina.

Ottimizzazione delle sezioni Risorse e Condizioni delle policy

Puoi limitare l'ambito delle autorizzazioni di un utente specificando le risorse e le condizioni in una policy AWS Identity and Access Management (IAM). Ogni operazione in una policy supporta una combinazione di tipi di risorse e condizioni che varia a seconda del comportamento dell'operazione.

Ogni dichiarazione di policy IAM concede l'autorizzazione a un'operazione eseguita su una risorsa. Quando l'operazione non agisce su una risorsa designata oppure quando concedi l'autorizzazione per eseguire l'operazione su tutte le risorse, il valore della risorsa nella policy è un carattere jolly (`*`). Per molte operazioni, puoi limitare le risorse che un utente può modificare specificando il nome della risorsa Amazon (ARN) di una risorsa o uno schema ARN che corrisponde a più risorse.

Per tipo di risorsa, la struttura generale di come limitare l'ambito di un'azione è la seguente:

- Funzioni: le operazioni alla base di una funzione possono essere limitate a una funzione specifica in base alla funzione, alla versione o all'ARN dell'alias.
- Strumenti di mappatura dell'origine degli eventi: le azioni possono essere limitate a risorse specifiche dello strumento di mappatura dell'origine degli eventi tramite ARN. Gli strumenti di mappatura dell'origine degli eventi sono sempre associati a una funzione. È inoltre possibile utilizzare la condizione `lambda:FunctionArn` per limitare le operazioni in base alla funzione associata.

- Livelli: le operazioni che riguardano l'utilizzo e le autorizzazioni relative ai livelli agiscono su una versione di un livello.
- Configurazione della firma del codice: le azioni possono essere limitate a risorse di configurazione di firma del codice specifiche tramite ARN.
- Tag: utilizza condizioni standard per i tag. Per ulteriori informazioni, consulta the section called “Controllo dell'accesso basato sugli attributi”.

Per limitare le autorizzazioni in base alla risorsa, specifica la risorsa in base all'ARN.

Formato ARN della risorsa Lambda

- Funzione – `arn:aws:lambda:us-west-2:123456789012:function:my-function`
- Versione funzione – `arn:aws:lambda:us-west-2:123456789012:function:my-function:1`
- Alias funzione – `arn:aws:lambda:us-west-2:123456789012:function:my-function:TEST`
- Mappatura delle origini eventi – `arn:aws:lambda:us-west-2:123456789012:event-source-mapping:fa123456-14a1-4fd2-9fec-83de64ad683de6d47`
- Livello – `arn:aws:lambda:us-west-2:123456789012:layer:my-layer`
- Versione livello – `arn:aws:lambda:us-west-2:123456789012:layer:my-layer:1`
- Configurazione della firma del codice: `arn:aws:lambda:us-west-2:123456789012:code-signing-config:my-csc`

Ad esempio, la seguente politica consente a un utente di Account AWS `123456789012` richiamare una funzione denominata `my-function` nella regione Stati Uniti occidentali (Oregon) AWS .

Example Invocare la policy di una funzione

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Invoke",
```

```
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-
function"
        }
    ]
}
```

Questo è un caso speciale in cui l'identificatore dell'operazione (`lambda:InvokeFunction`) differisce dall'operazione API (Invoke). Per altre operazioni, l'identificatore dell'operazione è il nome dell'operazione preceduto dal prefisso `lambda:`.

Sections

- Informazioni sulla sezione Condizione nelle policy
- Funzioni di riferimento nella sezione Risorse delle policy
- Comportamenti di azioni e funzioni IAM supportati

Informazioni sulla sezione Condizione nelle policy

Le condizioni sono un elemento opzionale della policy che applica una logica aggiuntiva per stabilire se è consentita un'operazione. Oltre alle condizioni comuni supportate da tutte le operazioni, Lambda definisce i tipi di condizione che è possibile utilizzare per limitare i valori dei parametri aggiuntivi su alcune operazioni.

Ad esempio, la condizione `lambda:Principal` consente di limitare il servizio o un account al quale un utente può concedere l'accesso a una chiamata su una policy basata sulle risorse della funzione. La policy seguente consente a un utente di concedere l'autorizzazione agli argomenti di Amazon Simple Notification Service (Amazon SNS) per richiamare una funzione denominata `test`.

Example Gestione delle autorizzazioni di una policy della funzione

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"
                }
            }
        }
    ]
}
```

La condizione richiede che il principale sia Amazon SNS e non un altro servizio o account. Il modello di risorsa richiede che il nome della funzione sia `test` e includa un numero di versione o un alias. Ad esempio, `test:v1`.

Per ulteriori informazioni sulle risorse e le condizioni per Lambda e altri AWS servizi, consulta Azioni, risorse e chiavi di condizione per AWS i servizi nel Service Authorization Reference.

Funzioni di riferimento nella sezione Risorse delle policy

Puoi fare riferimento a una funzione Lambda in un rendiconto sulla policy utilizzando un Amazon Resource Name (ARN). Il formato dell'ARN di una funzione dipende dal fatto che si stia facendo riferimento all'intera funzione (non qualificato) o a una versione di funzione o a un alias (qualificato).

Quando effettuano chiamate API Lambda, gli utenti possono specificare una versione o un alias passando una versione ARN o un alias ARN nel parametro o impostando un valore GetFunction`FunctionName`nel parametro. GetFunction`Qualifier` Lambda prende decisioni di autorizzazione confrontando l'elemento risorsa nella policy IAM con `FunctionName` e `Qualifier` passati nelle chiamate API. Se c'è un errore, Lambda nega la richiesta.

Sia che si stia consentendo o negando un'operazione sulla funzione, è necessario utilizzare i tipi ARN della funzione corretti nell'istruzione di policy per ottenere i risultati previsti. Ad esempio, se la policy fa riferimento all'ARN non qualificato, Lambda accetta le richieste che fanno riferimento all'ARN non qualificato ma nega le richieste che fanno riferimento a un ARN qualificato.

> **Note**
>
> Non puoi utilizzare un carattere jolly (*) per la corrispondenza con l'ID account. Per informazioni sulla sintassi accettata, consulta Documentazione di riferimento sulle policy JSON per IAM nella Guida per l'utente di IAM.

Example permettere la chiamata di un ARN non qualificato

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction"
        }
    ]
}
```

Se la policy fa riferimento a uno specifico ARN qualificato, Lambda accetta le richieste che fanno riferimento a quell'ARN ma nega le richieste che fanno riferimento all'ARN non qualificato o a un ARN qualificato diverso, ad esempio `myFunction:2`.

Example permettere la chiamata di un ARN qualificato specifico

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction:1"
```

```
        }
    ]
}
```

Se tua policy fa riferimento a un ARN qualificato utilizzando `:*`, Lambda accetta qualsiasi ARN qualificato ma nega le richieste che fanno riferimento all'ARN non qualificato.

Example permettere la chiamata di un ARN qualificato

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction:*"
        }
    ]
}
```

Se la tua policy fa riferimento a qualsiasi ARN che utilizza `*`, Lambda accetta qualsiasi ARN qualificato o non qualificato.

Example permettere la chiamata di un ARN qualificato o non qualificato

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction*"
        }
```

```
    ]
}
```

Comportamenti di azioni e funzioni IAM supportati

Le azioni definiscono cosa può essere consentito tramite le policy IAM. Per un elenco delle operazioni supportate in Lambda, consulta Operazioni, risorse e chiavi di condizione per AWS Lambda in Service Authorization Reference. Nella maggior parte dei casi, quando un'azione IAM consente un'operazione API Lambda, il nome dell'azione IAM è lo stesso dell'operazione API Lambda, con le seguenti eccezioni:

Azione API	Azione IAM
Invoke	`lambda:InvokeFunction`
GetLayerVersion GetLayerVersionByArn	`lambda:GetLayerVersion`

Oltre alle risorse e alle condizioni definite nel Service Authorization Reference, Lambda supporta le seguenti risorse e condizioni per determinate azioni. Molte di queste sono correlate alle funzioni di riferimento nella sezione delle risorse delle policy. Le operazioni alla base di una funzione possono essere limitate a una funzione specifica in base alla funzione, alla versione o all'ARN dell'alias, come descritto nella tabella riportata di seguito.

Azione	Risorsa	Condizione
AddPermission RemovePermission Invoke (Autorizzazione:) `lambda:InvokeFunction`	Versione funzione Alias funzione	N/D
UpdateFunctionConfiguration	N/D	`lambda:CodeSigningConfigArn`
CreateFunctionUrlConfig	Alias funzione	N/D

Azione	Risorsa	Condizione
DeleteFunctionUrlConfig GetFunctionUrlConfig UpdateFunctionUrlConfig		

Sicurezza in AWS Lambda

La sicurezza del cloud AWS è la massima priorità. In qualità di AWS cliente, puoi beneficiare di un data center e di un'architettura di rete progettati per soddisfare i requisiti delle organizzazioni più sensibili alla sicurezza.

La sicurezza è una responsabilità condivisa tra AWS te e te. Il modello di responsabilità condivisa descrive questo come sicurezza del cloud e sicurezza nel cloud:

- Sicurezza del cloud: AWS è responsabile della protezione dell'infrastruttura che funziona Servizi AWS nel AWS cloud. AWS ti fornisce anche servizi che puoi utilizzare in modo sicuro. I revisori di terze parti testano e verificano regolarmente l'efficacia della sicurezza come parte dei programmi di conformitàAWS. Per maggiori informazioni sui programmi di conformità applicabili AWS Lambda, consulta Servizi AWS la sezione Scope by Compliance Program.
- Sicurezza nel cloud: la responsabilità dell'utente è determinata dal servizio AWS utilizzato. L'utente è anche responsabile di altri fattori, tra cui la riservatezza dei dati, i requisiti della propria azienda e le leggi e normative vigenti.

Questa documentazione consente di comprendere come applicare il modello di responsabilità condivisa quando si usa Lambda. I seguenti argomenti illustrano come configurare Lambda per soddisfare gli obiettivi di sicurezza e conformità. Imparerai anche a usarne altri Servizi AWS che ti aiutano a monitorare e proteggere le tue risorse Lambda.

Per maggiori informazioni sull'applicazione dei principi di sicurezza alle applicazioni Lambda, consulta Sicurezza in Serverless Land.

Argomenti

- Protezione dei dati in AWS Lambda
- Utilizzo di ruoli collegati ai servizi per Lambda
- Identity and Access Management per AWS Lambda
- Creare una strategia di governance per le funzioni e i livelli Lambda
- Convalida della conformità per AWS Lambda
- Resilienza nell' AWS Lambda
- Sicurezza dell'infrastruttura in AWS Lambda
- Protezione dei carichi di lavoro con endpoint pubblici

- Utilizzo della firma del codice per verificare l'integrità del codice con Lambda

Protezione dei dati in AWS Lambda

Il modello di responsabilità condivisa di AWS si applica alla protezione dei dati in AWS Lambda. Come descritto in questo modello, AWSè responsabile della protezione dell'infrastruttura globale che esegue tutto l'Cloud AWS. L'utente è responsabile del controllo dei contenuti ospitati su questa infrastruttura. L'utente è inoltre responsabile della configurazione della protezione e delle attività di gestione per i Servizi AWS utilizzati. Per maggiori informazioni sulla privacy dei dati, consultare le Domande frequenti sulla privacy dei dati. Per informazioni sulla protezione dei dati in Europa, consulta il post del blog relativo al Modello di responsabilità condivisa AWSe GDPR nel Blog sulla sicurezza AWS.

Per garantire la protezione dei dati, ti suggeriamo di proteggere le credenziali Account AWSe di configurare i singoli utenti con AWS IAM Identity Centero AWS Identity and Access Management(IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- Utilizza SSL/TLS per comunicare con le risorse AWS. È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Configurare l'API e la registrazione dei log delle attività degli utenti con AWS CloudTrail. Per informazioni sull'utilizzo dei percorsi CloudTrail per acquisire le attività AWS, consulta Utilizzo dei percorsi CloudTrail nella Guida per l'utente di AWS CloudTrail.
- Utilizza le soluzioni di crittografia AWS, insieme a tutti i controlli di sicurezza di default all'interno dei Servizi AWS.
- Utilizzare i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.
- Se necessiti di moduli crittografici convalidati FIPS 140-3 quando accedi ad AWS attraverso un'interfaccia a riga di comando o un'API, utilizza un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il Federal Information Processing Standard (FIPS) 140-3.

Ti consigliamo di non inserire mai informazioni riservate o sensibili, ad esempio gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero, ad esempio nel campo Nome. Ciò include quando si lavora con Lambda o altri Servizi AWS utilizzando la console, l'API, la AWS CLI o gli SDK AWS. I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per i la fatturazione o i log di diagnostica. Quando si fornisce un URL a un server esterno,

suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la richiesta al server.

Sezioni

- Crittografia in transito
- Crittografia dei dati a riposo per AWS Lambda

Crittografia in transito

Gli endpoint API di Lambda supportano solo connessioni protette tramite HTTPS. Quando si gestiscono le risorse Lambda attraverso la Console di gestione AWS, l'SDK AWS o l'API Lambda, tutte le comunicazioni sono crittografate con Transport Layer Security (TLS). Per un elenco completo degli endpoint API, consultare Regioni ed endpoint AWS in Riferimenti generali di AWS.

Quando si connette la funzione a un file system, Lambda utilizza Crittografia in transito per tutte le connessioni. Per ulteriori informazioni, consulta Crittografia dati in Amazon EFS nella Amazon Elastic File System User Guide.

Quando si utilizzano le variabili di ambiente, è possibile abilitare gli helper della crittografia della console per utilizzare il file crittografato lato client per proteggere le variabili di ambiente in transito. Per ulteriori informazioni, consulta Protezione delle variabili di ambiente Lambda.

Crittografia dei dati a riposo per AWS Lambda

Lambda fornisce sempre la crittografia a riposo per le seguenti risorse utilizzando un Chiave di proprietà di AWS o un Chiave gestita da AWS:

- Variabili di ambiente
- I file che vengono caricati in Lambda, inclusi i pacchetti di implementazione e gli archivi dei livelli
- Oggetti dei criteri del filtro dello strumento di mappatura dell'origine degli eventi

Facoltativamente, puoi configurare Lambda per utilizzare una chiave gestita dal cliente per crittografare le variabili di ambiente, pacchetti di implementazione .zip e gli oggetti dei criteri di filtro.

Amazon CloudWatch Logs e AWS X-Ray, inoltre, crittografano i dati per impostazione predefinita e possono essere configurati per l'utilizzo di una chiave gestita dal cliente. Per ulteriori informazioni, consulta Crittografare i dati di log in CloudWatch Logs e Protezione dei dati in AWS X-Ray.

Monitoraggio delle chiavi di crittografia per Lambda

Quando utilizzi una chiave gestita dal cliente AWS KMS con Lambda, puoi usare AWS CloudTrail. Di seguito sono riportati eventi CloudTrail per chiamate `Decrypt`, `DescribeKey` e `GenerateDataKey` effettuate da Lambda per accedere ai dati crittografati dalla chiave gestita dal cliente.

Decrypt

Se hai utilizzato una chiave gestita dal cliente AWS KMS per crittografare l'oggetto dei criteri di filtro, Lambda invia una richiesta `Decrypt` per tuo conto quando provi ad accedervi in testo semplice (ad esempio, da una chiamata `ListEventSourceMappings`). L'evento di esempio seguente registra l'operazione `Decrypt`:

```
{
    "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

Se hai utilizzato una chiave gestita dal cliente AWS KMS per crittografare l'oggetto dei criteri di filtro, Lambda invia una richiesta `DescribeKey` per tuo conto quando provi ad accedervi (ad esempio, da una chiamata `GetEventSourceMapping`). L'evento di esempio seguente registra l'operazione `DescribeKey`:

```
{
    "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

Quando utilizzi una chiave gestita dal cliente AWS KMS per crittografare l'oggetto dei criteri di filtro in una chiamata `CreateEventSourceMapping` o `UpdateEventSourceMapping`, Lambda invia una richiesta `GenerateDataKey` per tuo conto per generare una chiave di dati per crittografare i criteri di filtro (crittografia a busta). L'evento di esempio seguente registra l'operazione `GenerateDataKey`:

```
{
    "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"
}
```


Utilizzo di ruoli collegati ai servizi per Lambda

Lambda utilizza ruoli collegati ai servizi AWS Identity and Access Management (IAM). Un ruolo collegato ai servizi è un tipo unico di ruolo IAM collegato direttamente a Lambda. I ruoli collegati ai servizi sono predefiniti da Lambda e includono le autorizzazioni richieste dal servizio per chiamare altri servizi per tuo conto. AWS

Lambda definisce le autorizzazioni dei suoi ruoli collegati ai servizi e solo Lambda può assumerne i ruoli. Le autorizzazioni definite includono la policy di attendibilità e la policy delle autorizzazioni che non può essere allegata a nessun'altra entità IAM.

È possibile eliminare un ruolo collegato al servizio solo dopo avere eliminato le risorse correlate. Questo protegge le tue risorse Lambda perché non puoi rimuovere inavvertitamente l'autorizzazione ad accedere alle risorse.

Per informazioni su altri servizi che supportano i ruoli collegati ai servizi, consulta i AWS servizi che funzionano con IAM e cerca i servizi con Sì nella colonna Ruoli collegati ai servizi. Scegli Sì in corrispondenza di un link per visualizzare la documentazione relativa al ruolo collegato al servizio per tale servizio.

Autorizzazioni di ruolo collegate ai servizi per Lambda

Lambda utilizza il ruolo collegato al servizio denominato. AWSServiceRoleForLambda Ai fini dell'assunzione del ruolo, il ruolo collegato al servizio considera attendibili i seguenti servizi:

- `lambda.amazonaws.com`

La politica di autorizzazione dei ruoli denominata AWSLambda ServiceRolePolicy consente a Lambda di completare le seguenti azioni sulle risorse specificate:

- Azione: `ec2:TerminateInstances` attiva `arn:aws:ec2:*:*:instance/*` con la condizione che è uguale `ec2:ManagedResourceOperator` `scaler.lambda.amazonaws.com`
- Azione: `ec2:DescribeInstanceStatus` e `ec2:DescribeInstances` su `*`

Per consentire a utenti, gruppi o ruoli di creare, modificare o eliminare un ruolo orientato ai servizi, devi configurare le autorizzazioni. Per ulteriori informazioni, consulta Autorizzazioni del ruolo collegato ai servizi nella Guida per l'utente IAM.

Per gli aggiornamenti delle policy gestite, consulta Politiche gestite da Lambda.

Creazione di un ruolo collegato ai servizi per Lambda

Non hai bisogno di creare manualmente un ruolo collegato ai servizi. Quando crei un provider di capacità Lambda nella Console di gestione AWS, o nell' AWS API AWS CLI, Lambda crea automaticamente il ruolo collegato al servizio.

Se elimini questo ruolo collegato al servizio, è possibile ricrearlo seguendo lo stesso processo utilizzato per ricreare il ruolo nell'account. Quando crei un provider di capacità Lambda, Lambda crea nuovamente il ruolo collegato ai servizi per te.

Puoi anche utilizzare la console IAM per creare un ruolo collegato al servizio con lo use case. AWSServiceRoleForLambda Nella AWS CLI o nell' AWS API, crea un ruolo collegato al servizio con il nome del servizio. `lambda.amazonaws.com` Per ulteriori informazioni, consulta Creazione di un ruolo collegato ai servizi nella Guida per l'utente IAM. Se elimini il ruolo collegato ai servizi, è possibile utilizzare lo stesso processo per crearlo nuovamente.

Modifica di un ruolo collegato al servizio per Lambda

Lambda non consente di modificare il ruolo collegato al AWSService RoleForLambda servizio. Dopo avere creato un ruolo collegato al servizio, non sarà possibile modificarne il nome perché varie entità potrebbero farvi riferimento. È possibile tuttavia modificarne la descrizione utilizzando IAM. Per ulteriori informazioni, consulta Modifica di un ruolo collegato al servizio nella Guida per l'utente di IAM.

Eliminazione di un ruolo collegato al servizio per Lambda

Se non è più necessario utilizzare una funzionalità o un servizio che richiede un ruolo collegato al servizio, ti consigliamo di eliminare il ruolo. In questo modo non sarà più presente un'entità non utilizzata che non viene monitorata e gestita attivamente. Tuttavia, è necessario effettuare la pulizia delle risorse associate al ruolo collegato al servizio prima di poterlo eliminare manualmente.

> **Note**
>
> Se il servizio Lambda utilizza il ruolo quando si tenta di eliminare le risorse, l'eliminazione potrebbe non riuscire. In questo caso, attendi alcuni minuti e quindi ripeti l'operazione.

Per eliminare le risorse Lambda utilizzate da AWSService RoleForLambda

1. Rimuovi tutti i provider di capacità Lambda dal tuo account. Puoi farlo utilizzando la console Lambda, la CLI o l'API.
2. Verifica che nessun provider di capacità Lambda rimanga nel tuo account prima di tentare di eliminare il ruolo collegato al servizio.

Per eliminare manualmente il ruolo collegato ai servizi mediante IAM

Utilizza la console IAM AWS CLI, l'o l' AWS API per eliminare il ruolo collegato al servizio. AWSService RoleForLambda Per ulteriori informazioni, consulta Eliminazione del ruolo collegato ai servizi nella Guida per l'utente IAM.

Regioni supportate per i ruoli collegati ai servizi Lambda

Lambda non supporta l'utilizzo di ruoli collegati ai servizi in tutte le regioni in cui il servizio è disponibile. AWSServiceRoleForLambda è supportato nelle seguenti regioni.

Nome della Regione	Identità della Regione	Support in Lambda
Stati Uniti orientali (Virginia settentrionale)	us-east-1	Sì
Stati Uniti orientali (Ohio)	us-east-2	Sì
Stati Uniti occidentali (California settentrionale)	us-west-1	Sì
Stati Uniti occidentali (Oregon)	us-west-2	Sì
Africa (Città del Capo)	af-south-1	No
Asia Pacific (Hong Kong)	ap-east-1	Sì
Asia Pacifico (Giacarta)	ap-southeast-3	Sì
Asia Pacifico (Bangkok)	ap-southeast-7	Sì
Asia Pacifico (Mumbai)	ap-south-1	Sì
Asia Pacifico (Osaka)	ap-northeast-3	No
Asia Pacific (Seoul)	ap-northeast-2	No
Asia Pacifico (Singapore)	ap-southeast-1	Sì
Asia Pacifico (Sydney)	ap-southeast-2	Sì
Asia Pacifico (Tokyo)	ap-northeast-1	Sì

Nome della Regione	Identità della Regione	Support in Lambda
Canada (Centrale)	ca-central-1	No
Europa (Francoforte)	eu-central-1	Sì
Europa (Irlanda)	eu-west-1	Sì
Europa (Londra)	eu-west-2	Sì
Europe (Milan)	eu-south-1	No
Europa (Parigi)	eu-west-3	No
Europa (Stoccolma)	eu-north-1	No
Medio Oriente (Bahrein)	me-south-1	No
Medio Oriente (Emirati Arabi Uniti)	me-central-1	No
Sud America (San Paolo)	sa-east-1	No
AWS GovCloud (Stati Uniti orientali)	us-gov-east-1	No
AWS GovCloud (Stati Uniti occidentali)	us-gov-west-1	No

Identity and Access Management per AWS Lambda

AWS Identity and Access Management (IAM) è un Servizio AWS che consente agli amministratori di controllare in modo sicuro l'accesso alle risorse AWS. Gli amministratori IAM controllano chi può essere autenticato (connesso) e autorizzato (avere le autorizzazioni) a utilizzare le risorse Lambda. IAM è un Servizio AWS il cui uso non comporta costi aggiuntivi.

Argomenti

- Destinatari
- Autenticazione con identità
- Gestione dell'accesso con l'utilizzo delle policy
- Funzionamento di AWS Lambda con IAM

- Esempi di policy basate su identità per AWS Lambda
- AWS politiche gestite per AWS Lambda
- Risoluzione dei problemi di identità e accesso in AWS Lambda

Destinatari

Il modo in cui utilizzi AWS Identity and Access Management (IAM) varia in base al tuo ruolo:

- Utente del servizio: richiedi le autorizzazioni all'amministratore se non riesci ad accedere alle funzionalità (vedi Risoluzione dei problemi di identità e accesso in AWS Lambda)
- Amministratore del servizio: determina l'accesso degli utenti e invia le richieste di autorizzazione (vedi Funzionamento di AWS Lambda con IAM)
- Amministratore IAM: scrivi policy per gestire l'accesso (vedi Esempi di policy basate su identità per AWS Lambda)

Autenticazione con identità

L'autenticazione è la procedura di accesso ad AWScon le credenziali di identità. Devi essere autenticato come Utente root dell'account AWS, utente IAM o assumendo un ruolo IAM.

Puoi accedere come identità federata utilizzando credenziali provenienti da una fonte di identità come AWS IAM Identity Center (IAM Identity Center), autenticazione Single Sign-On o credenziali Google/Facebook. Per maggiori informazioni sull'accesso, consultare la sezione Come accedere a Account AWS nella Guida per l'utente di Accedi ad AWS.

Per l'accesso programmatico, AWS fornisce un SDK e una CLI per firmare crittograficamente le richieste. Per ulteriori informazioni, consulta AWS Signature Version 4 per le richieste API nella Guida per l'utente IAM.

Account AWSUtente root di un

Quando crei un Account AWS, inizi con una singola identità di accesso chiamata utente root Account AWS che ha accesso completo a tutti i Servizi AWS e a tutte le risorse. Si consiglia vivamente di non utilizzare l'utente root per le attività quotidiane. Per le attività che richiedono le credenziali come utente root, consulta Attività che richiedono le credenziali dell'utente root nella Guida per l'utente IAM.

Identità federata

Come best practice, chiedi agli utenti fisici di utilizzare la federazione con un provider di identità per accedere ad Servizi AWS con credenziali temporanee.

Un'identità federata è un utente della directory aziendale, un provider di identità Web o Directory Service che accede a Servizi AWS utilizzando credenziali fornite tramite un'origine di identità. Le identità federate assumono ruoli che forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, consigliamo di utilizzare AWS IAM Identity Center. Per ulteriori informazioni, consulta Cos'è IAM Identity Center? nella Guida per l'utente di AWS IAM Identity Center.

Utenti e gruppi IAM

Un utente IAM è un'identità che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ti consigliamo di utilizzare credenziali temporanee invece di utenti IAM con credenziali a lungo termine. Per maggiori informazioni, consulta Richiedere utenti umani per usare la federazione con un provider di identità per accedere a AWS utilizzando credenziali temporanee nella Guida per l'utente IAM.

Un gruppo IAM specifica una raccolta di utenti IAM e semplifica la gestione delle autorizzazioni per gestire gruppi di utenti di grandi dimensioni. Per ulteriori informazioni, consulta Casi d'uso per utenti IAM nella Guida per l'utente IAM.

Ruoli IAM

Un ruolo IAM è un'identità con autorizzazioni specifiche che fornisce credenziali temporanee. Puoi quindi assumere un ruolo passando da un utente a un ruolo IAM (console), chiamando AWS CLI o un'operazione API AWS. Per ulteriori informazioni, consulta Metodi per assumere un ruolonella Guida per l'utente IAM.

I ruoli IAM sono utili per l'accesso degli utenti federati, le autorizzazioni utente IAM temporanee, l'accesso multi-account, l'accesso tra servizi e le applicazioni in esecuzione su Amazon EC2. Per maggiori informazioni, consultare Accesso a risorse multi-account in IAM nella Guida per l'utente IAM.

Gestione dell'accesso con l'utilizzo delle policy

Per controllare l'accesso a AWS è possibile creare policy e collegarle a identità o risorse AWS. Una policy definisce le autorizzazioni se associata a un'identità o a una risorsa. AWS valuta queste policy

quando un principale effettua una richiesta. La maggior parte delle policy viene archiviata in AWS sotto forma di documenti JSON. Per maggiori informazioni sui documenti delle policy JSON, consulta Panoramica delle policy JSON nella Guida per l'utente IAM.

Utilizzando le policy, gli amministratori specificano chi ha accesso a cosa definendo quale principale può eseguire azioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Un amministratore IAM crea le policy IAM e le aggiunge ai ruoli, che gli utenti possono quindi assumere. Le policy IAM definiscono le autorizzazioni indipendentemente dal metodo utilizzato per eseguirle.

Policy basate sull'identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile collegare a un'identità (utente, gruppo o ruolo). Tali policy controllano le operazioni autorizzate per l'identità, nonché le risorse e le condizioni in cui possono essere eseguite. Per informazioni su come creare una policy basata su identità, consultare Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente nella Guida per l'utente IAM.

Le policy basate sull'identità possono essere policy in linea (incorporate direttamente in una singola identità) o policy gestite (policy autonome collegate a più identità). Per informazioni su come scegliere tra una policy gestita o una policy inline, consultare Scelta fra policy gestite e policy inline nella Guida per l'utente IAM.

Policy basate sulle risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Gli esempi includono le policy di trust dei ruoli IAM e le policy dei bucket di Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. In una policy basata sulle risorse è obbligatorio specificare un'entità principale.

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non è possibile utilizzare le policy gestite da AWS di IAM in una policy basata sulle risorse.

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi che possono impostare il numero massimo di autorizzazioni concesse dai tipi di policy più comuni.

- Limiti delle autorizzazioni: impostano il numero massimo di autorizzazioni che una policy basata su identità ha la possibilità di concedere a un'entità IAM. Per ulteriori informazioni, consulta Limiti delle autorizzazioni per le entità IAM nella Guida per l'utente IAM.
- Policy di controllo dei servizi (SCP): specificano il numero massimo di autorizzazioni per un'organizzazione o un'unità organizzativa (OU) in AWS Organizations. Per ulteriori informazioni, consultare Policy di controllo dei servizi nella Guida per l'utente di AWS Organizations.
- Le policy di controllo delle risorse (RCP) definiscono le autorizzazioni massime disponibili per le risorse degli account. Per ulteriori informazioni, consulta Policy di controllo delle risorse (RCP) nella Guida per l'utente di AWS Organizations.
- Le policy di sessione sono policy avanzate che si passano come parametro quando si crea una sessione temporanea per un ruolo o un utente federato. Per maggiori informazioni, consultare Policy di sessione nella Guida per l'utente IAM.

Più tipi di policy

Quando più tipi di policy si applicano a una richiesta, le autorizzazioni risultanti sono più complicate da comprendere. Per informazioni su come AWSdetermina se consentire una richiesta quando sono coinvolti più tipi di policy, consultare Logica di valutazione delle policy nella Guida per l'utente IAM.

Funzionamento di AWS Lambda con IAM

Prima di utilizzare IAM per gestire l'accesso a Lambda, scopri quali funzionalità di IAM sono disponibili per l'uso con .

Funzionalità IAM	Supporto Lambda
Policy basate sull'identità	Sì
Policy basate sulle risorse	Sì
Operazioni di policy	Sì
Risorse relative alle policy	Sì
Chiavi di condizione della policy (specifica del servizio)	Sì

Funzionalità IAM	Supporto Lambda
Liste di controllo degli accessi (ACL	No
ABAC (tag nelle policy)	Parziale
Credenziali temporanee	Sì
Inoltro delle sessioni di accesso (FAS)	No
Ruoli di servizio	Sì
Ruoli collegati ai servizi	Parziale

Per ottenere un quadro generale del funzionamento di Lambda e altri servizi AWS con la maggior parte delle funzionalità di IAM, consulta Servizi AWS supportati da IAM nella Guida per l'utente IAM.

Policy basate sulle identità per Lambda

Supporta le policy basate sull'identità: sì

Le policy basate sull'identità sono documenti di policy di autorizzazione JSON che è possibile allegare a un'identità (utente, gruppo di utenti o ruolo IAM). Tali policy definiscono le operazioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consultare Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente nella Guida per l'utente IAM.

Con le policy basate sull'identità di IAM, è possibile specificare quali operazioni e risorse sono consentite o respinte, nonché le condizioni in base alle quali le operazioni sono consentite o respinte. Per informazioni su tutti gli elementi utilizzabili in una policy JSON, consulta Guida di riferimento agli elementi delle policy JSON IAM nella Guida per l'utente IAM.

Esempi di policy basate su identità per Lambda

Per visualizzare esempi di policy basate su identità Lambda, consulta Esempi di policy basate su identità per AWS Lambda.

Policy basate sulle risorse all'interno di Lambda

Supporta le policy basate sulle risorse: sì

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Esempi di policy basate sulle risorse sono le policy di attendibilità dei ruoli di IAM e le policy dei bucket di Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. Quando è collegata a una risorsa, una policy definisce le operazioni che un principale può eseguire su tale risorsa e a quali condizioni. In una policy basata sulle risorse è obbligatorio specificare un'entità principale. Le entità principali possono includere account, utenti, ruoli, utenti federati o Servizi AWS.

Per consentire l'accesso multi-account, è possibile specificare un intero account o entità IAM in un altro account come entità principale in una policy basata sulle risorse. Per ulteriori informazioni, consulta Accesso a risorse multi-account in IAM nella Guida per l'utente IAM.

È possibile associare una policy basata su risorse a una funzione o a un livello Lambda. Questa policy definisce quali principali possono eseguire azioni sulla funzione o il livello.

Per informazioni su come collegare una policy basata su risorse a una funzione o un livello, consulta Visualizzazione delle policy IAM basate sulle risorse in Lambda.

Operazioni di policy per Lambda

Supporta le operazioni di policy: si

Gli amministratori possono utilizzare le policy JSON di AWS per specificare gli accessi ai diversi elementi. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento `Action` di una policy JSON descrive le operazioni che è possibile utilizzare per consentire o negare l'accesso a un criterio. Includere le operazioni in una policy per concedere le autorizzazioni a eseguire l'operazione associata.

Per visualizzare un elenco di operazioni Lambda, consulta Operazioni definite da AWS Lambda nella Service Authorization Reference.

Le operazioni delle policy in Lambda utilizzano il seguente prefisso prima dell'operazione:

```
lambda
```

Per specificare più operazioni in una sola istruzione, occorre separarle con la virgola.

```
"Action": [
```

```
      "lambda:action1",
      "lambda:action2"
         ]
```

Per visualizzare esempi di policy basate su identità Lambda, consulta Esempi di policy basate su identità per AWS Lambda.

Risorse di policy per Lambda

Supporta le risorse relative alle policy: sì

Gli amministratori possono utilizzare le policy JSON di AWS per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento JSON `Resource` della policy specifica l'oggetto o gli oggetti ai quali si applica l'operazione. Come best practice, specificare una risorsa utilizzando il suo nome della risorsa Amazon (ARN). Per le azioni che non supportano le autorizzazioni a livello di risorsa, utilizzare un carattere jolly (*) per indicare che l'istruzione si applica a tutte le risorse.

```
"Resource": "*"
```

Per visualizzare un elenco di tipi di risorse Lambda e dei relativi ARN, consulta Tipi di risorse definite da AWS Lambda in Service Authorization Reference. Per informazioni sulle operazioni con cui è possibile specificare l'ARN di ogni risorsa, consulta la sezione Operazioni definite da AWS Lambda.

Per visualizzare esempi di policy basate su identità Lambda, consulta Esempi di policy basate su identità per AWS Lambda.

Chiavi di condizione delle policy per Lambda

Supporta le chiavi di condizione delle policy specifiche del servizio: sì

Gli amministratori possono utilizzare le policy JSON di AWS per specificare gli accessi ai diversi elementi. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento `Condition` specifica quando le istruzioni vengono eseguite in base a criteri definiti. È possibile compilare espressioni condizionali che utilizzano operatori di condizione, ad esempio

uguale a o minore di, per soddisfare la condizione nella policy con i valori nella richiesta. Per visualizzare tutte le chiavi di condizione globali di AWS, consultare Chiavi di contesto delle condizioni globali di AWS nella Guida per l'utente IAM.

Per visualizzare un elenco di chiavi di condizione Lambda, consulta Chiavi di condizione per AWS Lambda in Service Authorization Reference. Per informazioni su operazioni e risorse con cui è possibile utilizzare una chiave di condizione, consulta la sezione Operazioni definite da AWS Lambda.

Per visualizzare esempi di policy basate su identità Lambda, consulta Esempi di policy basate su identità per AWS Lambda.

ACL in Lambda

Supporta le ACL: no

Le liste di controllo degli accessi (ACL) controllano quali principali (membri, utenti o ruoli dell'account) hanno le autorizzazioni per accedere a una risorsa. Le ACL sono simili alle policy basate su risorse, sebbene non utilizzino il formato del documento di policy JSON.

ABAC con Lambda

Supporta ABAC (tag nelle policy): parzialmente

Il controllo degli accessi basato su attributi (ABAC) è una strategia di autorizzazione che definisce le autorizzazioni in base agli attributi, chiamati tag. Puoi allegare tag alle entità IAM e alle risorse AWS, quindi progettare policy ABAC per consentire operazioni quando il tag del principale corrisponde al tag sulla risorsa.

Per controllare l'accesso basato su tag, fornire informazioni sui tag nell'elemento condizione di una policy utilizzando le chiavi di condizione `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`* o `aws:TagKeys`.

Se un servizio supporta tutte e tre le chiavi di condizione per ogni tipo di risorsa, il valore per il servizio è Sì. Se un servizio supporta tutte e tre le chiavi di condizione solo per alcuni tipi di risorsa, allora il valore sarà Parziale.

Per maggiori informazioni su ABAC, consultare Definizione delle autorizzazioni con autorizzazione ABAC nella Guida per l'utente IAM. Per visualizzare un tutorial con i passaggi per l'impostazione di ABAC, consultare Utilizzo del controllo degli accessi basato su attributi (ABAC) nella Guida per l'utente IAM.

Per ulteriori informazioni sul tagging delle risorse Lambda, consulta Utilizzo del controllo degli accessi basato sugli attributi in Lambda.

Utilizzo di credenziali temporanee con Lambda

Supporta le credenziali temporanee: sì

Le credenziali temporanee forniscono l'accesso a breve termine alle risorse AWS e vengono create automaticamente quando si utilizza la federazione o si cambia ruolo. AWS consiglia di generare credenziali temporanee in modo dinamico anziché utilizzare chiavi di accesso a lungo termine. Per ulteriori informazioni, consulta Credenziali di sicurezza temporanee in IAM e Servizi AWS compatibili con IAM nelle Guida per l'utente IAM.

Inoltro delle sessioni di accesso per Lambda

Supporta l'inoltro delle sessioni di accesso (FAS): no

La tecnologia FAS utilizza le autorizzazioni dell'entità principale che effettua la chiamata a un Servizio AWS, combinate con la richiesta di un Servizio AWS, per effettuare richieste a servizi downstream. Per i dettagli delle policy relative alle richieste FAS, consultare Forward access sessions.

Ruoli di servizio per Lambda

Supporta i ruoli di servizio: sì

Un ruolo di servizio è un ruolo IAM che un servizio assume per eseguire operazioni per tuo conto. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta la sezione Create a role to delegate permissions to an Servizio AWS nella Guida per l'utente IAM.

In Lambda, un ruolo di servizio è noto come ruolo di esecuzione.

> **Warning**
>
> La modifica delle autorizzazioni per un ruolo di servizio potrebbe compromettere la funzionalità di Lambda.

Ruoli collegati ai servizi per Lambda

Supporta i ruoli legati ai servizi: Parziale

Un ruolo collegato al servizio è un tipo di ruolo di servizio che è collegato a un Servizio AWS. Il servizio può assumere il ruolo per eseguire un'azione per tuo conto. I ruoli collegati ai servizi sono visualizzati nell'account Account AWSe sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati al servizio, ma non modificarle.

Lambda non dispone di ruoli collegati al servizio, a differenza di Lambda@Edge. Per ulteriori informazioni, consulta Ruoli collegati ai servizi di Lambda@Edge nella Guida per gli sviluppatori di Amazon CloudFront.

Per ulteriori informazioni su come creare e gestire i ruoli collegati ai servizi, consulta Servizi AWS supportati da IAM. Trova un servizio nella tabella che include un `Yes` nella colonna Service-linked role (Ruolo collegato ai servizi). Scegli il collegamento Sì per visualizzare la documentazione relativa al ruolo collegato ai servizi per tale servizio.

Esempi di policy basate su identità per AWS Lambda

Per impostazione predefinita, gli utenti e i ruoli non sono autorizzati a creare o modificare le risorse Lambda. Per concedere agli utenti l'autorizzazione a eseguire operazioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM.

Per informazioni su come creare una policy basata su identità IAM utilizzando questi documenti di policy JSON di esempio, consulta Creazione di policy IAM (console) nella Guida per l'utente IAM.

Per informazioni dettagliate sulle operazioni e sui tipi di risorse definiti da Lambda, incluso il formato degli ARN per ogni tipo di risorsa, consulta Operazioni, risorse e chiavi di condizione per AWS Lambda in Service Authorization Reference.

Argomenti

- Best practice per le policy
- Uso della console Lambda
- Consentire agli utenti di visualizzare le loro autorizzazioni

Best practice per le policy

Le policy basate su identità determinano se qualcuno può creare, accedere o eliminare risorse Lambda nel tuo account. Queste operazioni possono comportare costi aggiuntivi per l'Account AWS. Quando si creano o modificano policy basate sull'identità, seguire queste linee guida e raccomandazioni:

- Nozioni di base sulle policy gestite da AWS e passaggio alle autorizzazioni con privilegio minimo - Per le informazioni di base su come concedere autorizzazioni a utenti e carichi di lavoro, utilizzare le policy gestite da AWS che concedono le autorizzazioni per molti casi d'uso comuni. Sono disponibili all'interno dell'Account AWS. Consigliamo pertanto di ridurre ulteriormente le autorizzazioni definendo policy gestite dal cliente di AWS specifiche per i propri casi d'uso. Per maggiori informazioni, consultare Policy gestite da AWS o Policy gestite da AWS per le funzioni dei processi nella Guida per l'utente IAM.
- Applicazione delle autorizzazioni con privilegio minimo - Quando si impostano le autorizzazioni con le policy IAM, concedere solo le autorizzazioni richieste per eseguire un'attività. È possibile farlo definendo le azioni che possono essere intraprese su risorse specifiche in condizioni specifiche, note anche come autorizzazioni con privilegio minimo. Per maggiori informazioni sull'utilizzo di IAM per applicare le autorizzazioni, consultare Policy e autorizzazioni in IAM nella Guida per l'utente IAM.
- Condizioni d'uso nelle policy IAM per limitare ulteriormente l'accesso - Per limitare l'accesso a operazioni e risorse è possibile aggiungere una condizione alle policy. Ad esempio, è possibile scrivere una condizione di policy per specificare che tutte le richieste devono essere inviate utilizzando SSL. È possibile inoltre utilizzare le condizioni per concedere l'accesso alle operazioni di servizio, ma solo se vengono utilizzate tramite uno specifico Servizio AWS, ad esempio CloudFormation. Per maggiori informazioni, consultare la sezione Elementi delle policy JSON di IAM: condizione nella Guida per l'utente IAM.
- Utilizzo di IAM Access Analyzer per convalidare le policy IAM e garantire autorizzazioni sicure e funzionali - IAM Access Analyzer convalida le policy nuove ed esistenti in modo che aderiscano alla sintassi della policy IAM (JSON) e alle best practice di IAM. IAM Access Analyzer offre oltre 100 controlli delle policy e consigli utili per creare policy sicure e funzionali. Per maggiori informazioni, consultare Convalida delle policy per il Sistema di analisi degli accessi IAM nella Guida per l'utente IAM.
- Richiesta dell'autenticazione a più fattori (MFA) - Se ci si trova di fronte a uno scenario che richiede utenti IAM o utenti root nell'Account AWS, attivare MFA per una maggiore sicurezza. Per richiedere la MFA quando vengono chiamate le operazioni API, aggiungere le condizioni MFA alle policy. Per maggiori informazioni, consultare Protezione dell'accesso API con MFA nella Guida per l'utente IAM.

Per maggiori informazioni sulle best practice in IAM, consultare Best practice di sicurezza in IAM nella Guida per l'utente IAM.

Uso della console Lambda

Per accedere alla console AWS Lambda è necessario disporre di un insieme di autorizzazioni minimo. Queste autorizzazioni devono consentirti di elencare e visualizzare i dettagli sulle risorse Lambda disponibili nel tuo Account AWS. Se si cra una policy basata sull'identità più restrittiva rispetto alle autorizzazioni minime richieste, la console non funzionerà nel modo previsto per le entità (utenti o ruoli) associate a tale policy.

Non è necessario concedere le autorizzazioni minime della console agli utenti che effettuano chiamate solo alla AWS CLI o all'API di AWS. Al contrario, è opportuno concedere l'accesso solo alle operazioni che corrispondono all'operazione API che stanno cercando di eseguire.

Per una policy di esempio che consente l'accesso minimo per lo sviluppo di funzioni, vedere Concedere agli utenti l'accesso a una funzione Lambda. Oltre alle API Lambda, la console Lambda utilizza altri servizi per visualizzare la configurazione di attivazione e permettere l'aggiunta di nuovi trigger. Se gli utenti utilizzano Lambda con altri servizi, devono anche accedere a tali servizi. Per informazioni dettagliate sulla configurazione di altri servizi con Lambda, consultare Richiamare Lambda con eventi di altri servizi AWS.

Consentire agli utenti di visualizzare le loro autorizzazioni

Questo esempio mostra in che modo è possibile creare una policy che consente agli utenti IAM di visualizzare le policy inline e gestite che sono cpllegate alla relativa identità utente. Questa policy include le autorizzazioni per completare questa operazione sulla console o a livello di codice utilizzando AWS CLI o l'API 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 politiche gestite per AWS Lambda

Una policy AWS gestita è una policy autonoma creata e amministrata da. AWS AWS le politiche gestite sono progettate per fornire autorizzazioni per molti casi d'uso comuni, in modo da poter iniziare ad assegnare autorizzazioni a utenti, gruppi e ruoli.

Tieni presente che le policy AWS gestite potrebbero non concedere le autorizzazioni con il privilegio minimo per i tuoi casi d'uso specifici, poiché sono disponibili per tutti i clienti. AWS Si consiglia pertanto di ridurre ulteriormente le autorizzazioni definendo policy gestite dal cliente specifiche per i propri casi d’uso.

Non è possibile modificare le autorizzazioni definite nelle politiche gestite. AWS Se AWS aggiorna le autorizzazioni definite in una politica AWS gestita, l'aggiornamento ha effetto su tutte le identità principali (utenti, gruppi e ruoli) a cui è associata la politica. AWS è più probabile che aggiorni una policy AWS gestita quando ne Servizio AWS viene lanciata una nuova o quando diventano disponibili nuove operazioni API per i servizi esistenti.

Per ulteriori informazioni, consultare Policy gestite da AWS nella Guida per l’utente di IAM.

Argomenti

- AWS politica gestita: AWSLambda_FullAccess

- AWS politica gestita: AWSLambda_ReadOnlyAccess
- AWS politica gestita: AWSLambda BasicExecutionRole
- AWS politica gestita: AWSLambda BasicDurableExecutionRolePolicy
- AWS politica gestita: Dynamo Role AWSLambda DBExecution
- AWS politica gestita: accesso AWSLambda ENIManagement
- AWS politica gestita: AWSLambdaInvocation-DynamoDB
- AWS politica gestita: AWSLambda KinesisExecutionRole
- AWS politica gestita: ruolo AWSLambda MSKExecution
- AWS politica gestita: ruolo AWSLambda
- AWS politica gestita: AWSLambda SQSQueue ExecutionRole
- AWS politica gestita: AWSLambda VPCAccess ExecutionRole
- AWS politica gestita: AWSLambda gestita EC2 ResourceOperator
- AWS politica gestita: AWSLambda ServiceRolePolicy
- Aggiornamenti Lambda alle AWS politiche gestite

AWS politica gestita: AWSLambda_FullAccess

Questa policy concede l'accesso completo a tutte le operazioni di Lambda. Concede inoltre le autorizzazioni ad altri AWS servizi utilizzati per sviluppare e gestire le risorse Lambda.

Puoi associare la policy `AWSLambda_FullAccess` ai tuoi utenti, gruppi e ruoli.

Dettagli delle autorizzazioni

Questa policy include le seguenti autorizzazioni:

- `lambda`: consente ai principali l'accesso completo a Lambda.
- `cloudformation`— Consente ai responsabili di descrivere gli AWS CloudFormation stack ed elencare le risorse in essi contenuti.
- `cloudwatch`— Consente ai responsabili di elencare i CloudWatch parametri di Amazon e ottenere dati sui parametri.
- `ec2`— Consente ai responsabili di descrivere gruppi di sicurezza, sottoreti e. VPCs
- `iam`: consente ai principali di ottenere policy, versioni delle policy, ruoli, policy dei ruoli, policy di ruolo allegate e l'elenco dei ruoli. Questa policy consente inoltre ai principali di trasferire ruoli a

Lambda. L'autorizzazione `PassRole` è necessaria quando si assegna un ruolo di esecuzione a una funzione. L'`CreateServiceLinkedRole`autorizzazione viene utilizzata durante la creazione di un ruolo collegato al servizio.

- `kms`— Consente ai responsabili di elencare gli alias e descrivere la chiave per la crittografia dei volumi.
- `logs`— Consente ai responsabili di descrivere i flussi di log, ottenere eventi di registro, filtrare gli eventi di registro e avviare e interrompere le sessioni di Live Tail.
- `states`— Consente ai responsabili di descrivere ed AWS Step Functions elencare le macchine a stati.
- `tag`: consente ai principali di ottenere risorse in base ai loro tag.
- `xray`— Consente ai responsabili di ottenere riepiloghi delle AWS X-Ray tracce e recuperare un elenco di tracce specificato dall'ID.

Per ulteriori informazioni su questa politica, inclusi il documento sulla policy JSON e le versioni delle policy, consulta la AWS Managed Policy AWSLambda_FullAccessReference Guide.

AWS politica gestita: AWSLambda_ReadOnlyAccess

Questa policy garantisce l'accesso in sola lettura alle risorse Lambda e ad altri AWS servizi utilizzati per sviluppare e gestire le risorse Lambda.

Puoi associare la policy `AWSLambda_ReadOnlyAccess` ai tuoi utenti, gruppi e ruoli.

Dettagli delle autorizzazioni

Questa policy include le seguenti autorizzazioni:

- `lambda`: consente ai principali di ottenere ed elencare tutte le risorse.
- `cloudformation`— Consente ai responsabili di descrivere ed elencare gli AWS CloudFormation stack e di elencare le risorse in essi contenuti.
- `cloudwatch`— Consente ai responsabili di elencare i CloudWatch parametri di Amazon e ottenere dati sui parametri.
- `ec2`— Consente ai responsabili di descrivere gruppi di sicurezza, sottoreti e. VPCs
- `iam`: consente ai principali di ottenere policy, versioni delle policy, ruoli, policy dei ruoli, policy di ruolo allegate e l'elenco dei ruoli.
- `kms`: consente ai principali di elencare gli alias.

- `logs`— Consente ai responsabili di descrivere i flussi di registro, ottenere eventi di registro, filtrare gli eventi di registro e avviare e interrompere le sessioni di Live Tail.
- `states`— Consente ai responsabili di descrivere ed AWS Step Functions elencare le macchine a stati.
- `tag`: consente ai principali di ottenere risorse in base ai loro tag.
- `xray`— Consente ai responsabili di ottenere riepiloghi delle AWS X-Ray tracce e recuperare un elenco di tracce specificato dall'ID.

Per ulteriori informazioni su questa politica, inclusi il documento sulla policy JSON e le versioni delle policy, consulta la AWS Managed Policy AWSLambda_ReadOnlyAccessReference Guide.

AWS politica gestita: AWSLambda BasicExecutionRole

Questa politica concede le autorizzazioni per caricare i log in Logs. CloudWatch

Puoi associare la policy `AWSLambdaBasicExecutionRole` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento sulla policy JSON e le versioni delle politiche, consulta la Managed Policy Reference AWSLambdaBasicExecutionRoleGuide.AWS

AWS politica gestita: AWSLambda BasicDurableExecutionRolePolicy

Questa policy fornisce autorizzazioni di scrittura per CloudWatch i log e read/write autorizzazioni per l'esecuzione durevole utilizzate dalle funzioni durevoli APIs di Lambda. Questa policy fornisce le autorizzazioni essenziali richieste per le funzioni durevoli Lambda, che utilizzano un' APIs esecuzione duratura per mantenere l'avanzamento e mantenere lo stato tra le chiamate di funzione.

Puoi associare la policy `AWSLambdaBasicDurableExecutionRolePolicy` ai tuoi utenti, gruppi e ruoli.

Dettagli delle autorizzazioni

Questa policy include le seguenti autorizzazioni:

- `logs`— Consente ai responsabili di creare gruppi di log e flussi di log e di scrivere eventi di log in Logs. CloudWatch
- `lambda`— Consente ai responsabili di controllare lo stato di esecuzione durevole e recuperare lo stato di esecuzione durevole per le funzioni durevoli Lambda.

Per visualizzare ulteriori dettagli sulla policy, inclusa la versione più recente del documento di policy JSON, consulta AWSLambdaBasicDurableExecutionRolePolicy nella Guida di riferimento alle policy gestite da AWS .

AWS politica gestita: Dynamo Role AWSLambda DBExecution

Questa policy concede le autorizzazioni per leggere i record da uno stream Amazon DynamoDB e scrivere su Logs. CloudWatch

Puoi associare la policy `AWSLambdaDynamoDBExecutionRole` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento e le versioni della policy JSON, consulta AWSLambdaDynamo DBExecution Role nella Managed Policy Reference Guide.AWS

AWS politica gestita: accesso AWSLambda ENIManagement

Questa policy concede le autorizzazioni per creare, descrivere ed eliminare interfacce di rete elastiche utilizzate da una funzione Lambda abilitata per VPC.

Puoi associare la policy `AWSLambdaENIManagementAccess` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento sulla policy JSON e le versioni delle policy, consulta AWSLambdaENIManagementAccess nella AWS Managed Policy Reference Guide.

AWS politica gestita: AWSLambdaInvocation-DynamoDB

Questa policy concede l'accesso in lettura ai flussi Amazon DynamoDB.

Puoi associare la policy `AWSLambdaInvocation-DynamoDB` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento sulla policy JSON e le versioni delle policy, consulta AWSLambdaInvocation-DynamoDBla AWS Managed Policy Reference Guide.

AWS politica gestita: AWSLambda KinesisExecutionRole

Questa policy concede le autorizzazioni per leggere gli eventi da un flusso di dati di Amazon Kinesis e scriverli nei log. CloudWatch

Puoi associare la policy `AWSLambdaKinesisExecutionRole` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, incluso il documento e le versioni della policy JSON, consulta la Managed Policy Reference AWSLambdaKinesisExecutionRoleGuide.AWS

AWS politica gestita: ruolo AWSLambda MSKExecution

Questa policy concede le autorizzazioni per leggere e accedere ai record da un cluster Amazon Managed Streaming for Apache Kafka, gestire interfacce di rete elastiche e scrivere su Logs. CloudWatch

Puoi associare la policy `AWSLambdaMSKExecutionRole` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento e le versioni della policy JSON, consulta AWSLambda MSKExecution Role in the Managed Policy Reference Guide.AWS

AWS politica gestita: ruolo AWSLambda

Questa policy concede le autorizzazioni per invocare le funzioni Lambda.

Puoi associare la policy `AWSLambdaRole` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento sulla policy JSON e le versioni della policy, consulta AWSLambdaRole nella AWS Managed Policy Reference Guide.

AWS politica gestita: AWSLambda SQSQueue ExecutionRole

Questa politica concede le autorizzazioni per leggere ed eliminare i messaggi da una coda di Amazon Simple Queue Service e concede le autorizzazioni di scrittura ai log. CloudWatch

Puoi associare la policy `AWSLambdaSQSQueueExecutionRole` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento e le versioni della policy JSON, consulta la Managed Policy Reference Guide. AWSLambdaSQSQueueExecutionRoleAWS

AWS politica gestita: AWSLambda VPCAccess ExecutionRole

Questa politica concede le autorizzazioni per gestire interfacce di rete elastiche all'interno di un Amazon Virtual Private Cloud e scrivere su Logs. CloudWatch

Puoi associare la policy `AWSLambdaVPCAccessExecutionRole` ai tuoi utenti, gruppi e ruoli.

Per ulteriori informazioni su questa politica, inclusi il documento e le versioni della policy JSON, consulta la Managed Policy Reference AWSLambdaVPCAccessExecutionRoleGuide.AWS

AWS politica gestita: AWSLambda gestita EC2 ResourceOperator

Questa policy consente la gestione automatizzata delle istanze di Amazon Elastic Compute Cloud per i provider di capacità Lambda. Concede le autorizzazioni al servizio scaler Lambda per eseguire operazioni sul ciclo di vita delle istanze per tuo conto.

Puoi associare la policy `AWSLambdaManagedEC2ResourceOperator` ai tuoi utenti, gruppi e ruoli.

Dettagli delle autorizzazioni

Questa policy include le seguenti autorizzazioni:

- `ec2:RunInstances`— Consente a Lambda di lanciare nuove EC2 istanze Amazon a condizione che ec2: sia uguale a scaler.lambda.amazonaws.com e limiti l'ManagedResourceOperator utilizzo dell'AMI alle sole immagini di proprietà di Amazon.
- `ec2:DescribeInstancese ec2:DescribeInstanceStatus` — Consente a Lambda di monitorare lo stato dell'istanza e recuperare le informazioni sull'istanza.
- `ec2:CreateTags`— Consente a Lambda di etichettare EC2 le risorse Amazon per scopi di gestione e identificazione.
- `ec2:DescribeAvailabilityZones`— Consente a Lambda di visualizzare le zone disponibili, ad esempio le decisioni di posizionamento.
- `ec2:DescribeCapacityReservations`— Consente a Lambda di verificare le prenotazioni di capacità per un posizionamento ottimale delle istanze.
- `ec2:DescribeInstanceTypeseec2:DescribeInstanceTypeOfferings`: consente a Lambda di esaminare i tipi di istanze disponibili e le relative offerte.
- `ec2:DescribeSubnets`— Consente a Lambda di esaminare le configurazioni delle sottoreti per la pianificazione della rete.
- `ec2:DescribeSecurityGroups`— Consente a Lambda di recuperare le informazioni sul gruppo di sicurezza per la configurazione dell'interfaccia di rete.
- `ec2:CreateNetworkInterface`— Consente a Lambda di creare interfacce di rete e gestire associazioni di sottoreti e gruppi di sicurezza.
- `ec2:AttachNetworkInterface`— Consente a Lambda di collegare interfacce di rete alle EC2 istanze Amazon con una condizione uguale a scaler.lambda.amazonaws.com. `ec2:ManagedResourceOperator`

Per ulteriori informazioni su questa politica, inclusi il documento e le versioni della policy JSON, consulta Managed in the Managed Policy Reference Guide. AWSLambda EC2 ResourceOperator AWS

AWS politica gestita: AWSLambda ServiceRolePolicy

Questa policy è associata al ruolo collegato ai servizi denominato per consentire AWSService RoleForLambda a Lambda di terminare le istanze gestite come parte dei provider di capacità Lambda.

Dettagli delle autorizzazioni

Questa policy include le seguenti autorizzazioni:

- `ec2:TerminateInstances`— Consente a Lambda di terminare le EC2 istanze a condizione che ec2: sia uguale a scaler.lambda.amazonaws.com. ManagedResourceOperator
- `ec2:DescribeInstanceStatus`e `ec2:DescribeInstances` — Consente a Lambda di descrivere EC2 le istanze.

Per ulteriori informazioni su questa politica, consulta Using service-linked roles for Lambda.

Aggiornamenti Lambda alle AWS politiche gestite

Modifica	Descrizione	Data
AWSLambdaGestito EC2 ResourceOperator: nuova politica	Lambda ha aggiunto una nuova policy gestita per abilitare la gestione automatizzata delle EC2 istanze Amazon per i fornitori di capacità Lambda, consentendo al servizio scaler di eseguire operazioni sul ciclo di vita delle istanze.	30 novembre 2025
AWSLambdaServiceRolePolicy: nuova policy	Lambda ha aggiunto una nuova policy gestita per il ruolo collegato ai servizi	30 novembre 2025

Modifica	Descrizione	Data
	per consentire a Lambda di terminare le istanze gestite come parte dei provider di capacità Lambda.	
AWSLambda_FullAccess— Cambia	Lambda ha aggiornato la `AWSLambda_FullAccess` policy per consentire le azioni `kms:DescribeKey` `eiam:CreateServiceLinkedRole` .	30 novembre 2025
AWSLambdaBasicDurableExecutionRolePolicy— Nuova politica gestita	Lambda ha rilasciato una nuova policy gestita `AWSLambdaBasicDurableExecutionRolePolicy` che fornisce autorizzazioni di scrittura per CloudWatch i log e read/write autorizzazioni per l'esecuzione durevole utilizzate APIs dalle funzioni durevoli di Lambda.	1 dicembre 2025
AWSLambda_ReadOnlyAccesse AWSLambda_FullAccess— Cambia	Lambda ha aggiornato le `AWSLambda_FullAccess` politiche `AWSLambda_ReadOnlyAccess` and per consentire le azioni `logs:StartLiveTail` and`logs:StopLiveTail` .	17 marzo 2025

Modifica	Descrizione	Data
AWSLambdaVPCAccessExecutionRole— Cambia	Lambda ha aggiornato la policy `AWSLambdaVPCAccessExecutionRole` per consentire l'azione `ec2:DescribeSubnets` .	5 gennaio 2024
AWSLambda_ReadOnlyAccess— Cambiare	Lambda ha aggiornato la `AWSLambda_ReadOnlyAccess` policy per consentire ai principali di elencare gli stack. CloudFormation	27 luglio 2023
AWS Lambda ha iniziato a tenere traccia delle modifiche	AWS Lambda ha iniziato a tenere traccia delle modifiche per le sue politiche AWS gestite.	27 luglio 2023

Risoluzione dei problemi di identità e accesso in AWS Lambda

Utilizza le seguenti informazioni per diagnosticare e risolvere i problemi comuni che possono verificarsi durante l'uso di Lambda e IAM.

Argomenti

- Non sono autorizzato/a a eseguire un'operazione in Lambda
- Non si possiede l’autorizzazione a eseguire iam:PassRole
- Voglio consentire alle persone esterne al mio Account AWS di accedere alle mie risorse Lambda

Non sono autorizzato/a a eseguire un'operazione in Lambda

Se ricevi un errore che indica che non sei autorizzato a eseguire un’operazione, le tue policy devono essere aggiornate per poter eseguire l’operazione.

L’errore di esempio seguente si verifica quando l’utente IAM `mateojackson` prova a utilizzare la console per visualizzare i dettagli relativi a una risorsa *`my-example-widget`* fittizia ma non dispone di autorizzazioni `lambda:`*`GetWidget`* fittizie.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 lambda:GetWidget on resource: my-example-widget
```

In questo caso, la policy per l'utente `mateojackson` deve essere aggiornata per consentire l'accesso alla risorsa *`my-example-widget`* utilizzando l'azione `lambda:GetWidget`.

Per ulteriore assistenza con l'accesso, contattare l'amministratore AWS. L'amministratore è la persona che ha fornito le credenziali di accesso.

Non si possiede l'autorizzazione a eseguire iam:PassRole

Se ricevi un errore che indica che non sei autorizzato a eseguire l'operazione `iam:PassRole`, devi aggiornare le policy per poter passare un ruolo a Lambda.

Alcuni Servizi AWS consentono di trasmettere un ruolo esistente a tale servizio, invece di creare un nuovo ruolo di servizio o un ruolo collegato al servizio. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per trasmettere il ruolo al servizio.

L'errore di esempio riportato di seguito si verifica quando un utente IAM denominato `marymajor` tenta di utilizzare la console per eseguire un'operazione in Lambda. Tuttavia, l'azione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per trasmettere il ruolo al servizio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In questo caso, le policy di Mary devono essere aggiornate per poter eseguire l'operazione `iam:PassRole`.

Per ulteriore assistenza con l'accesso, contattare l'amministratore AWS. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Voglio consentire alle persone esterne al mio Account AWS di accedere alle mie risorse Lambda

È possibile creare un ruolo con il quale utenti in altri account o persone esterne all'organizzazione possono accedere alle tue risorse. È possibile specificare chi è attendibile per l'assunzione del ruolo. Per servizi che supportano policy basate su risorse o liste di controllo degli accessi (ACL), utilizzare tali policy per concedere alle persone l'accesso alle proprie risorse.

Per ulteriori informazioni, consultare gli argomenti seguenti:

- Per sapere se Lambda supporta queste funzionalità, consultare Funzionamento di AWS Lambda con IAM.
- Per informazioni su come garantire l'accesso alle risorse negli Account AWS che possiedi, consultare Fornire l'accesso a un utente IAM in un altro Account AWS che si possiede nella Guida per l'utente di IAM.
- Per informazioni su come fornire l'accesso alle risorse ad Account AWS di terze parti, consultare Fornire l'accesso agli Account AWS di proprietà di terze parti nella Guida per l'utente IAM.
- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consultare Fornire l'accesso a utenti autenticati esternamente (Federazione delle identità) nella Guida per l'utente IAM.
- Per informazioni sulle differenze di utilizzo tra ruoli e policy basate su risorse per l'accesso multi-account, consulta Accesso a risorse multi-account in IAM nella Guida per l'utente IAM.

Creare una strategia di governance per le funzioni e i livelli Lambda

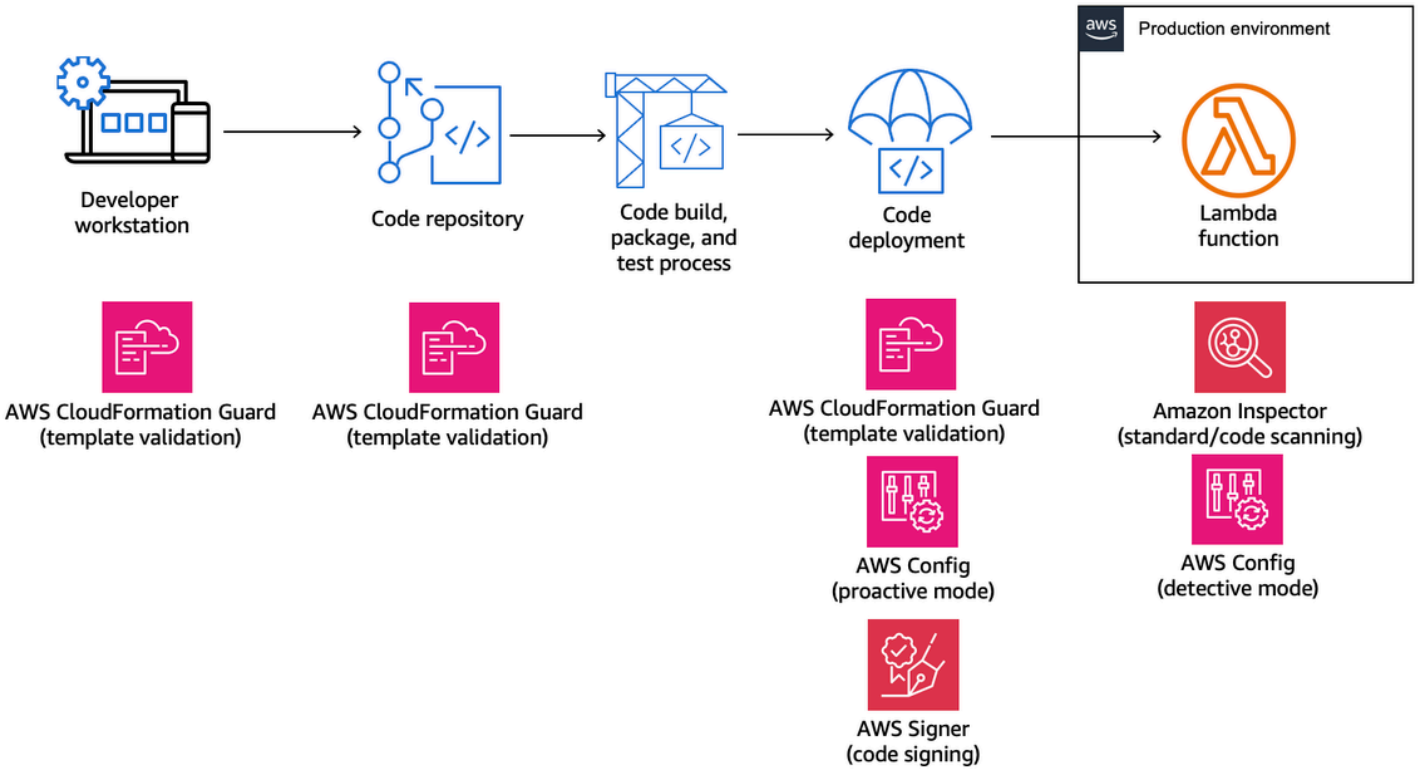
Per creare e implementare applicazioni serverless native del cloud, è necessario garantire agilità e velocità di immissione sul mercato con una governance e guardrail adeguati. Devi stabilire le priorità a livello aziendale, magari enfatizzando l'agilità come priorità assoluta o, in alternativa, sottolineando l'avversione al rischio attraverso governance, guardrail e controlli. Realisticamente, non adotterai una strategia "o/o", ma una strategia "e" che bilanci agilità e guardrail nel ciclo di vita dello sviluppo software. A prescindere dal punto del ciclo di vita dell'azienda in cui tali requisiti rientrano, è probabile che le funzionalità di governance diventino un requisito di implementazione nei processi e nelle toolchain.

Di seguito sono riportati alcuni esempi di controlli di governance che un'organizzazione potrebbe implementare per Lambda:

- Le funzioni Lambda non devono essere accessibili pubblicamente.
- Le funzioni Lambda devono essere associate a un VPC.
- Le funzioni Lambda non dovrebbero utilizzare runtime ritirati.
- Le funzioni Lambda devono essere etichettate con un set di tag obbligatori.
- I livelli Lambda non devono essere accessibili all'esterno dell'organizzazione.

- Le funzioni Lambda con un gruppo di sicurezza associato devono disporre di corrispondenti tra la funzione e il gruppo di sicurezza.
- Le funzioni Lambda con un livello associato devono utilizzare una versione approvata.
- Le variabili di ambiente Lambda devono essere crittografate a riposo con una chiave gestita dal cliente.

Il diagramma seguente è un esempio di una strategia di governance approfondita che implementa controlli e policy durante tutto il processo di sviluppo e implementazione del software:



I seguenti argomenti spiegano come implementare i controlli per lo sviluppo e l'implementazione di funzioni Lambda nell'organizzazione, sia per le startup che per le imprese. L'organizzazione potrebbe già disporre di alcuni strumenti. Gli argomenti seguenti adottano un approccio modulare a tali controlli, affinché si possano scegliere i componenti effettivamente necessari.

Argomenti

- Controlli proattivi per Lambda con AWS CloudFormation Guard
- Implementa controlli preventivi per Lambda con AWS Config
- Rilevare implementazioni e configurazioni Lambda non conformi con AWS Config
- Firma del codice Lambda con AWS Signer

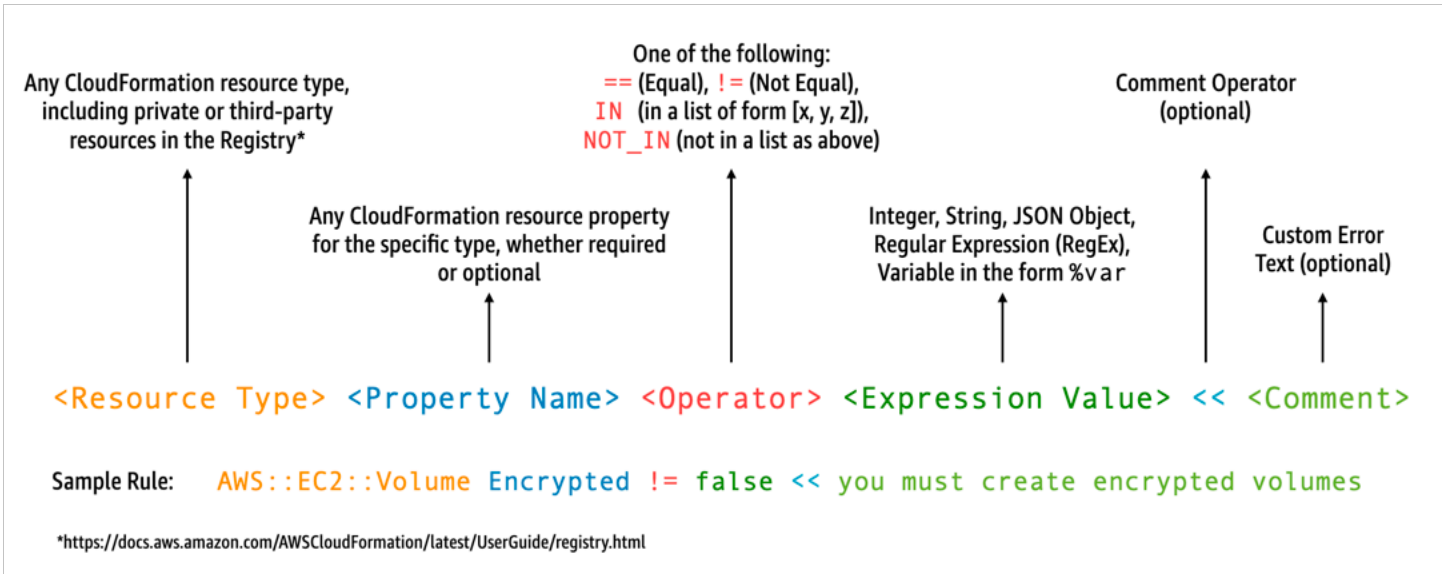
- Automatizzare le valutazioni di sicurezza per Lambda con Amazon Inspector
- Implementazione dell'osservabilità per la sicurezza e la conformità Lambda

Controlli proattivi per Lambda con AWS CloudFormation Guard

AWS CloudFormation Guard è uno strumento di valutazione policy as code open source per uso generico. Può essere utilizzato per la governance preventiva e la conformità attraverso la convalida dei modelli di infrastructure as code (IaC) e le composizioni dei servizi rispetto alle regole delle policy. Queste regole possono essere personalizzate in base ai requisiti del team o dell'organizzazione. Per le funzioni Lambda, è possibile utilizzare le regole Guard per controllare la creazione di risorse e gli aggiornamenti della configurazione definendo le impostazioni di proprietà richieste necessarie durante la creazione o l'aggiornamento di una funzione Lambda.

Gli amministratori addetti alla conformità definiscono l'elenco dei controlli e delle policy di governance necessari per l'implementazione e l'aggiornamento delle funzioni Lambda. Gli amministratori della piattaforma implementano i controlli nelle pipeline CI/CD, come webhook di convalida pre-commit con repository di codice, e forniscono agli sviluppatori strumenti a riga di comando per la convalida di modelli e codice nelle postazioni di lavoro locali. Gli sviluppatori creano codice, convalidano i modelli con strumenti a riga di comando e quindi eseguono il commit del codice nei repository, che vengono poi convalidati in automatico attraverso le pipeline CI/CD prima dell'implementazione in un ambiente AWS.

Guard ti consente di scrivere le regole e di implementare i controlli con un linguaggio specifico per il dominio come riportato di seguito.



Ad esempio, supponi di volerti assicurare che gli sviluppatori scelgano solo i runtime più recenti. Potresti specificare due policy diverse, una per identificare i runtime già ritirati e l'altra per identificare

i runtime che verranno ritirati a breve. A tale scopo, potresti scrivere il file `etc/rules.guard` seguente:

```
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.>>
            }
        }
    }
}
```

Supponiamo ora di scrivere il seguente modello CloudFormation `iac/lambda.yaml`, che definisce una funzione 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
```

Dopo aver eseguito l'installazione della funzionalità Guard, convalida il modello:

```
cfn-guard validate --rules etc/rules.guard --data iac/lambda.yaml
```

L'output sarà il seguente:

```
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

      }
    }
```

```
  }
}
```

Guard consente agli sviluppatori di vedere dalle loro postazioni di lavoro locali che devono aggiornare il modello per utilizzare un runtime consentito dall'organizzazione. Ciò avviene prima di effettuare il commit in un repository di codice e quindi di non superare i controlli all'interno di una pipeline CI/CD. Di conseguenza, gli sviluppatori ricevono questo feedback su come sviluppare modelli conformi e dedicare più tempo alla scrittura di codice che offra valore aziendale. Questo controllo può essere applicato alla postazione di lavoro locale degli sviluppatori, a un webhook di convalida pre-commit e/o alla pipeline CI/CD prima dell'implementazione.

Avvertenze

Se utilizzi modelli AWS Serverless Application Model (AWS SAM) per definire le funzioni Lambda, tieni presente che devi aggiornare la regola Guard per cercare il tipo di risorsa `AWS::Serverless::Function` come riportato di seguito.

```
let lambda_functions = Resources.*[
    Type == "AWS::Serverless::Function"
]
```

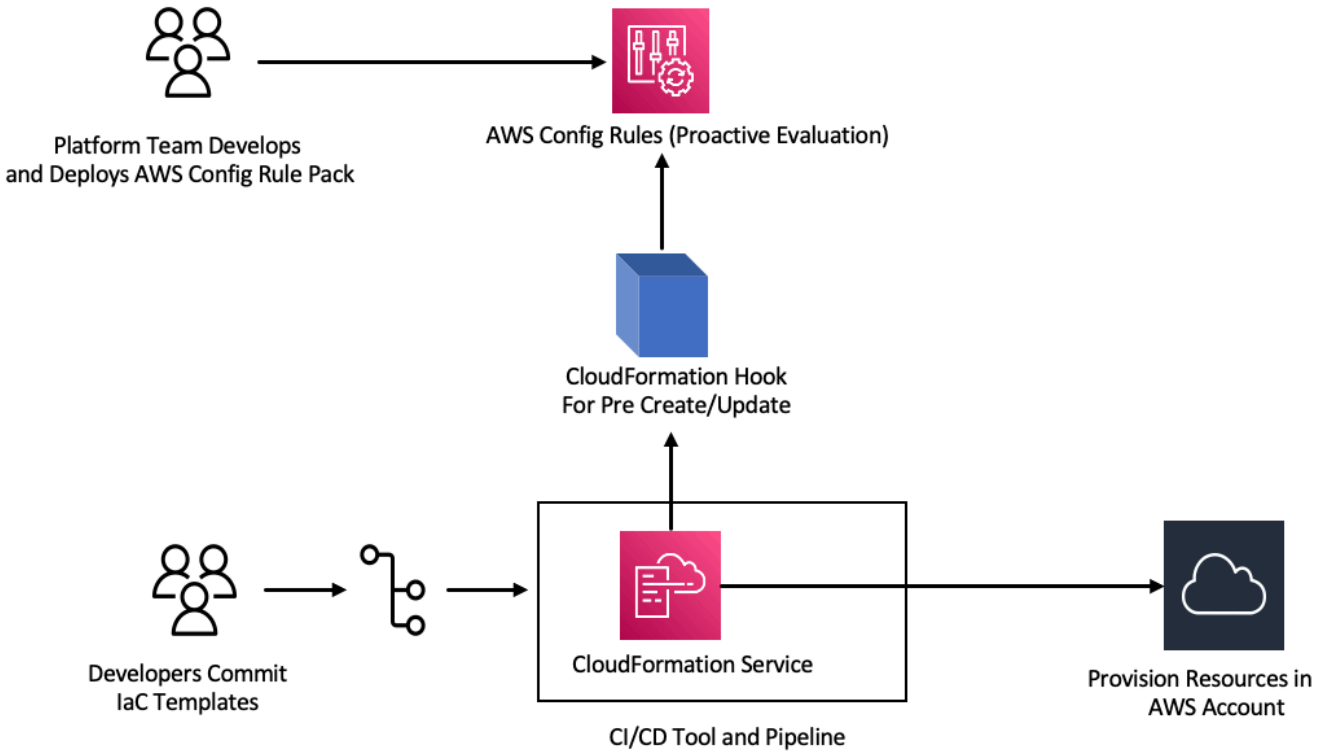
Guard si aspetta inoltre che le proprietà vengano incluse nella definizione della risorsa. Nel frattempo, i modelli AWS SAM consentono di specificare le proprietà in una sezione Globali separata. Le proprietà definite nella sezione Globali non vengono convalidate con le regole Guard.

Come indicato nella documentazione per la risoluzione dei problemi di Guard, tieni presente che Guard non supporta funzioni intrinseche in formato breve come `!GetAtt` o `!Sub` e richiede invece l'utilizzo dei formati espansi: `Fn::GetAtt` e `Fn::Sub`. (L'esempio precedente non valuta la proprietà Ruolo, quindi per semplicità è stata utilizzata la funzione intrinseca in formato breve.)

Implementa controlli preventivi per Lambda con AWS Config

È essenziale garantire la conformità delle applicazioni serverless il più presto possibile nel processo di sviluppo. In questo argomento, spieghiamo come implementare controlli preventivi utilizzando AWS Config. Ciò consente di implementare i controlli di conformità nelle fasi iniziali del processo di sviluppo e di implementare gli stessi controlli nelle pipeline. CI/CD Ciò standardizza anche i controlli in un archivio di regole gestito centralmente in modo da poter applicare i controlli in modo uniforme su tutti gli account. AWS

Ad esempio, supponiamo che gli amministratori della conformità abbiano definito un requisito per garantire che tutte le funzioni Lambda includano il tracciamento. AWS X-Ray Con AWS Config la modalità proattiva, puoi eseguire controlli di conformità sulle risorse delle funzioni Lambda prima dell'implementazione, riducendo il rischio di implementare funzioni Lambda configurate in modo errato e facendo risparmiare tempo agli sviluppatori fornendo loro un feedback più rapido sull'infrastruttura sotto forma di modelli di codice. Di seguito è riportata una visualizzazione del flusso per i controlli preventivi con: AWS Config



Immagina che vi sia un requisito per cui il tracciamento deve essere abilitato in tutte le funzioni Lambda. In risposta, il team della piattaforma identifica la necessità di una AWS Config regola specifica da applicare in modo proattivo su tutti gli account. Questa regola contrassegna come risorsa non conforme ogni funzione Lambda priva di una configurazione di tracciamento X-Ray configurata. Il team sviluppa una regola, la racchiude in un pacchetto di conformità e distribuisce il pacchetto di conformità su tutti gli AWS account per garantire che tutti gli account dell'organizzazione applichino questi controlli in modo uniforme. Puoi scrivere la regola nella sintassi 2.x.x di AWS CloudFormation Guard , che assume la forma seguente:

```
rule name when condition { assertion }
```

Di seguito è riportato un esempio di regola Guard che verifica l'abilitazione del tracciamento nelle funzioni Lambda:

```
rule lambda_tracing_check {
  when configuration.tracingConfig exists {
      configuration.tracingConfig.mode == "Active"
  }
}
```

Il team della piattaforma intraprende ulteriori azioni imponendo che ogni implementazione richiami un hook di pre-creazione/aggiornamento. AWS CloudFormation Il team si assume la piena responsabilità dello sviluppo di tale hook e di configurare la pipeline, rafforzando il controllo centralizzato delle regole di conformità e sostenendo l'applicazione coerente in tutte le implementazioni. Per sviluppare, impacchettare e registrare un hook, consulta Developing AWS CloudFormation Hooks nella documentazione dell'interfaccia a riga di CloudFormation comando (CFN-CLI). Puoi usare la CloudFormation CLI per creare il progetto hook:

```
cfn init
```

Questo comando richiede alcune informazioni di base sul progetto hook e crea un progetto contenente i file seguenti:

```
README.md
<hook-name>.json
rpdk.log
src/handler.py
template.yml
```

```
hook-role.yaml
```

Come sviluppatore di hook, devi aggiungere il tipo di risorsa di destinazione desiderato nel file di configurazione `<hook-name>.json`. Nella configurazione seguente, un hook è configurato per l'esecuzione prima di creare qualsiasi funzione Lambda utilizzando. CloudFormation Puoi anche aggiungere gestori simili per le azioni `preUpdate` e `preDelete`.

```
    "handlers": {
        "preCreate": {
            "targetNames": [
                "AWS::Lambda::Function"
            ],
            "permissions": []
        }
    }
```

È inoltre necessario assicurarsi che l' CloudFormation hook disponga delle autorizzazioni appropriate per chiamare il. AWS Config APIs Puoi farlo aggiornando il file di definizione del ruolo denominato `hook-role.yaml`. Per impostazione predefinita, il file di definizione del ruolo ha la seguente politica di fiducia, che consente di CloudFormation assumere il ruolo.

```
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - hooks.cloudformation.amazonaws.com
                - resources.cloudformation.amazonaws.com
```

Per consentire a questo hook di chiamare config APIs, è necessario aggiungere le seguenti autorizzazioni all'istruzione Policy. Quindi invii il progetto hook utilizzando il `cfn submit` comando, dove CloudFormation crea un ruolo per te con le autorizzazioni richieste.

```
      Policies:
        - PolicyName: HookTypePolicy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action:
```

```
                  - "config:Describe*"
                  - "config:Get*"
                  - "config:List*"
                  - "config:SelectResourceConfig"
                Resource: "*
```

Successivamente, devi scrivere una funzione Lambda in un file `src/handler.py`. All'interno di tale file, troverai i metodi denominati `preCreate`, `preUpdate` e `preDelete` già creati all'avvio del progetto. Il tuo obiettivo è scrivere una funzione comune e riutilizzabile che richiami l' AWS Config `StartResourceEvaluation`API in modalità proattiva utilizzando il. AWS SDK per Python (Boto3) Questa chiamata API accetta le proprietà della risorsa come input e valuta la risorsa rispetto alla definizione della regola.

```
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.")
```

Ora puoi effettuare la chiamata alla funzione comune dal gestore per l'hook di pre-creazione. Di seguito è riportato un esempio del gestore:

```
@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}")
```

Dopo questo passaggio è possibile registrare l'hook e configurarlo per ascoltare tutti gli eventi di creazione delle AWS Lambda funzioni.

Uno sviluppatore prepara il modello di infrastructure as code (IaC) per un microservizio serverless utilizzando Lambda. La preparazione prevede il rispetto degli standard interni, a cui seguono il test e il commit locale del modello nel repository. Ecco un esempio di modello 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
```

Come parte del CI/CD processo, quando il CloudFormation modello viene distribuito, il CloudFormation servizio richiama l'hook di pre-creazione/aggiornamento subito prima del provisioning del tipo di risorsa. `AWS::Lambda::Function` L'hook utilizza AWS Config regole eseguite in modalità proattiva per verificare che la configurazione della funzione Lambda includa la configurazione di tracciamento obbligatoria. La risposta dell'hook determina la fase successiva. Se conforme, l'hook segnala l'esito positivo e procede alla fornitura delle risorse. CloudFormation In caso contrario, l'implementazione CloudFormation dello stack fallisce, la pipeline si arresta immediatamente e il sistema registra i dettagli per la successiva revisione. Le notifiche di conformità vengono inviate ai soggetti interessati.

Puoi trovare le success/fail informazioni sull'hook nella console: CloudFormation

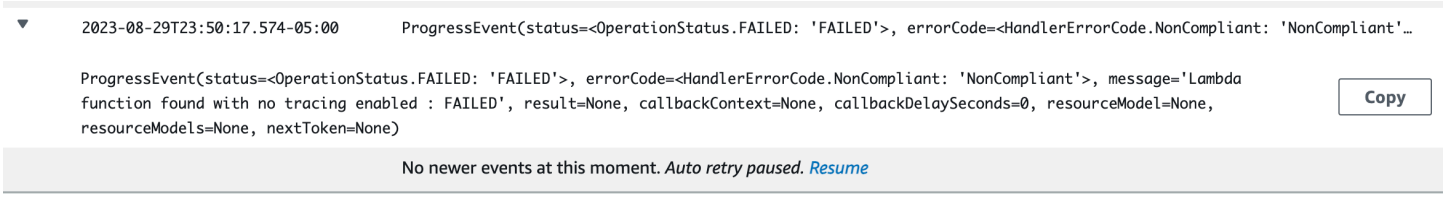
Stack info | Events | Resources | Outputs | Parameters | Template | Change sets

Events (19)

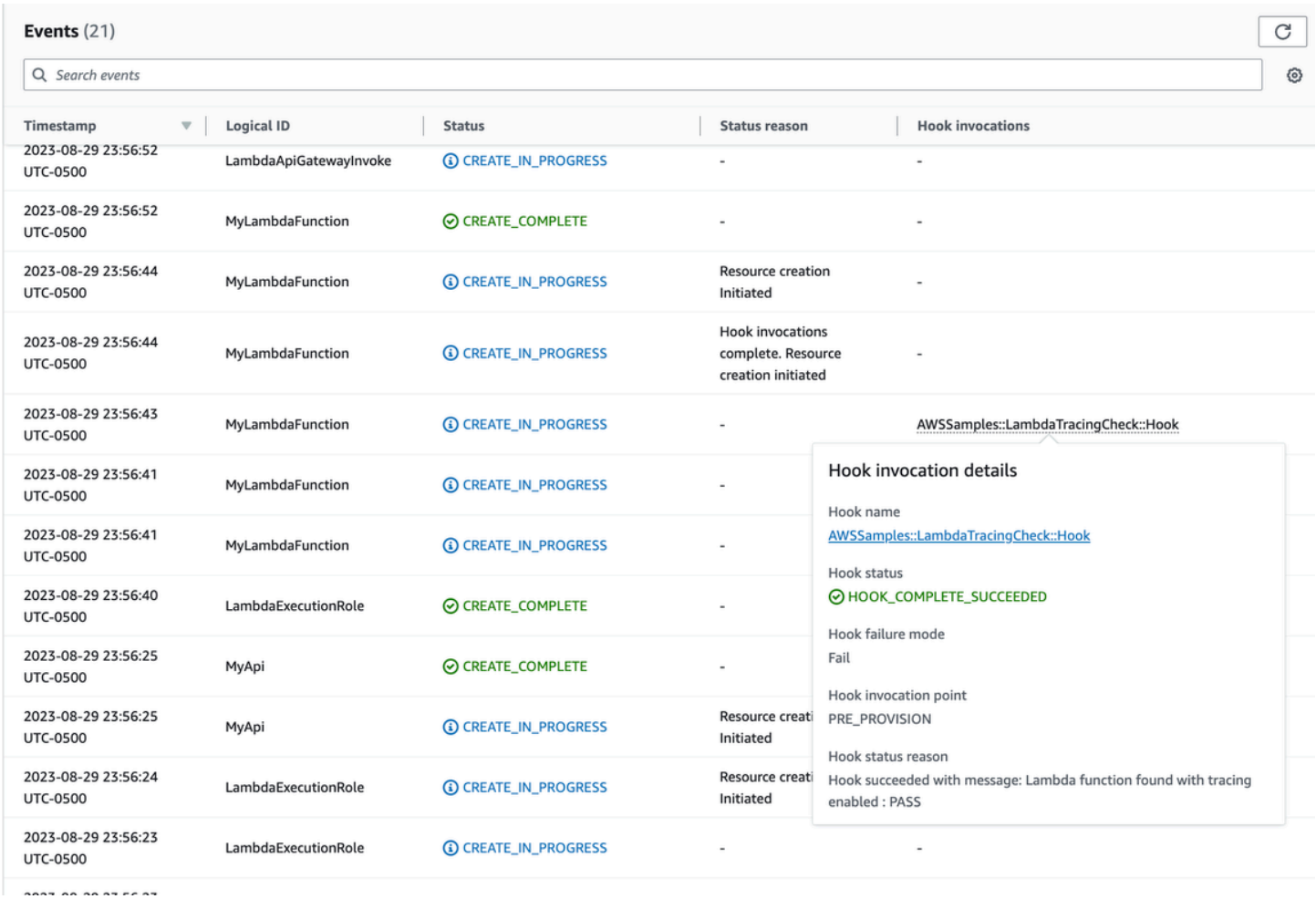
Search events

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	-

Se hai abilitato i log per il tuo CloudFormation hook, puoi acquisire il risultato della valutazione dell'hook. Di seguito è riportato un log di esempio relativo a un hook con stato non riuscito, che indica il fatto che nella funzione Lambda non è abilitato X-Ray:



Se lo sviluppatore sceglie di modificare l'IaC per aggiornare il valore `TracingConfig Mode` in `Active` ed eseguire nuovamente l'implementazione, l'hook viene eseguito correttamente e lo stack procede alla creazione della risorsa Lambda.



In questo modo, puoi implementare controlli preventivi AWS Config in modalità proattiva durante lo sviluppo e la distribuzione di risorse serverless nei tuoi account. AWS Integrando AWS Config le regole nella CI/CD pipeline, puoi identificare e, facoltativamente, bloccare le distribuzioni di risorse

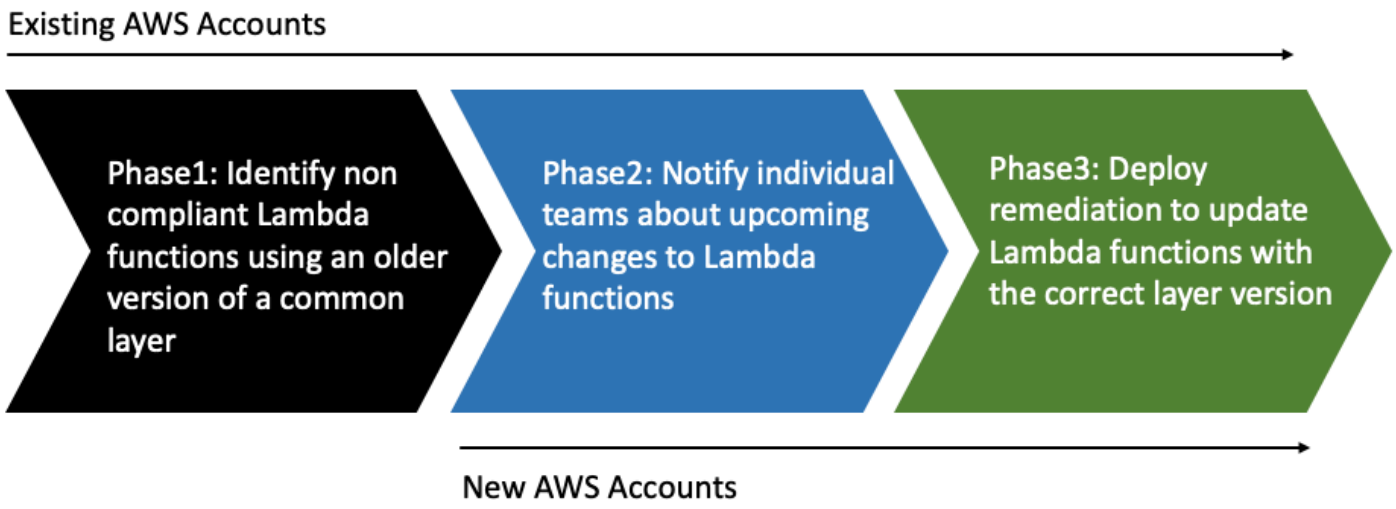
non conformi, come le funzioni Lambda prive di una configurazione di tracciamento attiva. Ciò garantisce che negli ambienti vengano implementate solo le risorse conformi alle più recenti politiche di governance. AWS

Rilevare implementazioni e configurazioni Lambda non conformi con AWS Config

Oltre alla valutazione proattiva, AWS Config può anche rilevare in modo reattivo le implementazioni e le configurazioni di risorse non conformi alle policy di governance. Si tratta di una funzione importante, perché le policy di governance si evolvono man mano che l'organizzazione apprende e implementa nuove best practice.

Immagina uno scenario in cui imposti una nuova policy durante l'implementazione o l'aggiornamento delle funzioni Lambda: tutte le funzioni Lambda devono sempre utilizzare una versione di livello Lambda specifica e approvata. Puoi configurare AWS Config per monitorare le funzioni nuove o aggiornate per le configurazioni di livello. Se rileva una funzione che non utilizza una versione di livello approvata, AWS Config la contrassegna come risorsa non conforme. Come opzione, puoi configurare AWS Config affinché ripari in automatico la risorsa specificando un'azione di riparazione attraverso un documento di automazione AWS Systems Manager. Ad esempio, è possibile scrivere un documento di automazione in Python utilizzando AWS SDK per Python (Boto3), che aggiorna la funzione non conforme in modo che punti alla versione di livello approvata. Pertanto, AWS Config funge da controllo sia di rilevamento che di correzione e in tal modo automatizza la gestione della conformità.

Suddividiamo questo processo in tre importanti fasi di implementazione:



Fase 1: identificazione delle risorse di accesso

Comincia attivando AWS Config su tutti i tuoi account e configurando il registro affinché registri le funzioni AWS Lambda. Ciò consente a AWS Config di osservare quando le funzioni Lambda vengono

create o aggiornate. Puoi quindi configurare regole di policy personalizzate per verificare la presenza di eventuali violazioni di policy specifiche, che utilizzano la sintassi AWS CloudFormation Guard. Le regole di Guard assumono la seguente forma generale:

```
rule name when condition { assertion }
```

Di seguito è riportato un esempio di regola che verifica che un livello non sia impostato su una versione di livello vecchia:

```
rule desiredlayer when configuration.layers !empty {
    some configuration.layers[*].arn != CONFIG_RULE_PARAMETERS.OldLayerArn
}
```

Analizziamo la sintassi e la struttura della regola:

- Nome della regola: il nome della regola nell'esempio presentato è `desiredlayer`.
- Condizione: questa clausola specifica la condizione in base alla quale la regola deve essere verificata. Nell'esempio fornito, la condizione è `configuration.layers !empty`. Ciò significa che la risorsa deve essere valutata solo quando la proprietà `layers` nella configurazione non è vuota.
- Asserzione: dopo la clausola `when`, un'asserzione determina che cosa verifica la regola. L'asserzione `some configuration.layers[*].arn != CONFIG_RULE_PARAMETERS.OldLayerArn` verifica se uno qualsiasi degli ARN di livello Lambda non corrisponde al valore `OldLayerArn`. Se non corrispondono, l'asserzione è vera e la regola viene rispettata; in caso contrario, ha esito negativo.

`CONFIG_RULE_PARAMETERS` è un insieme di parametri speciali configurato con la regola AWS Config. In questo caso, `OldLayerArn` è un parametro all'interno di `CONFIG_RULE_PARAMETERS`. Ciò consente agli utenti di fornire un valore di ARN specifico che ritengono vecchio o ritirato, quindi la regola verifica se alcune funzioni Lambda utilizzano l'ARN vecchio in questione.

Fase 2: visualizzazione e progettazione

AWS Config raccoglie i dati di configurazione e li archivia in bucket Amazon Simple Storage Service (Amazon S3). Puoi utilizzare Amazon Athena per inviare query a tali dati direttamente dai bucket S3. Con Athena, puoi aggregare questi dati a livello dell'organizzazione, generando una visione onnicomprensiva delle configurazioni delle risorse in tutti gli account. Per configurare l'aggregazione

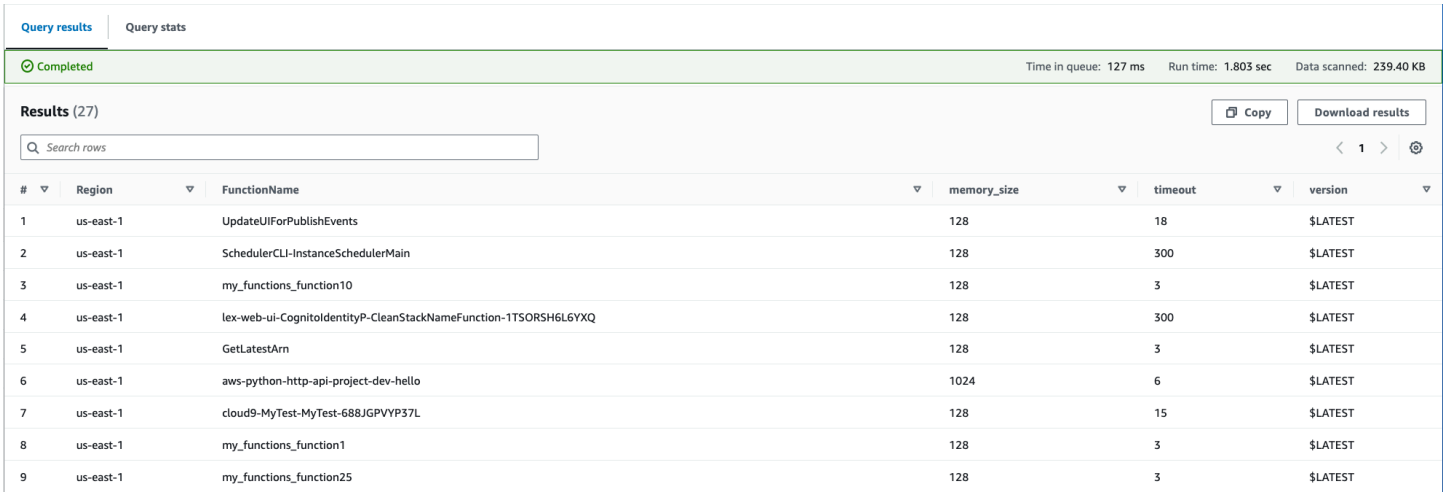
dei dati di configurazione delle risorse, consulta Visualizing AWS Config data using Athena and Amazon QuickSight nel blog AWS Cloud Operations and Management.

Di seguito è riportato un esempio di query Athena per identificare tutte le funzioni Lambda che utilizzano un particolare ARN di livello:

```
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%'
```

Ecco i risultati della query:



Query results | Query stats

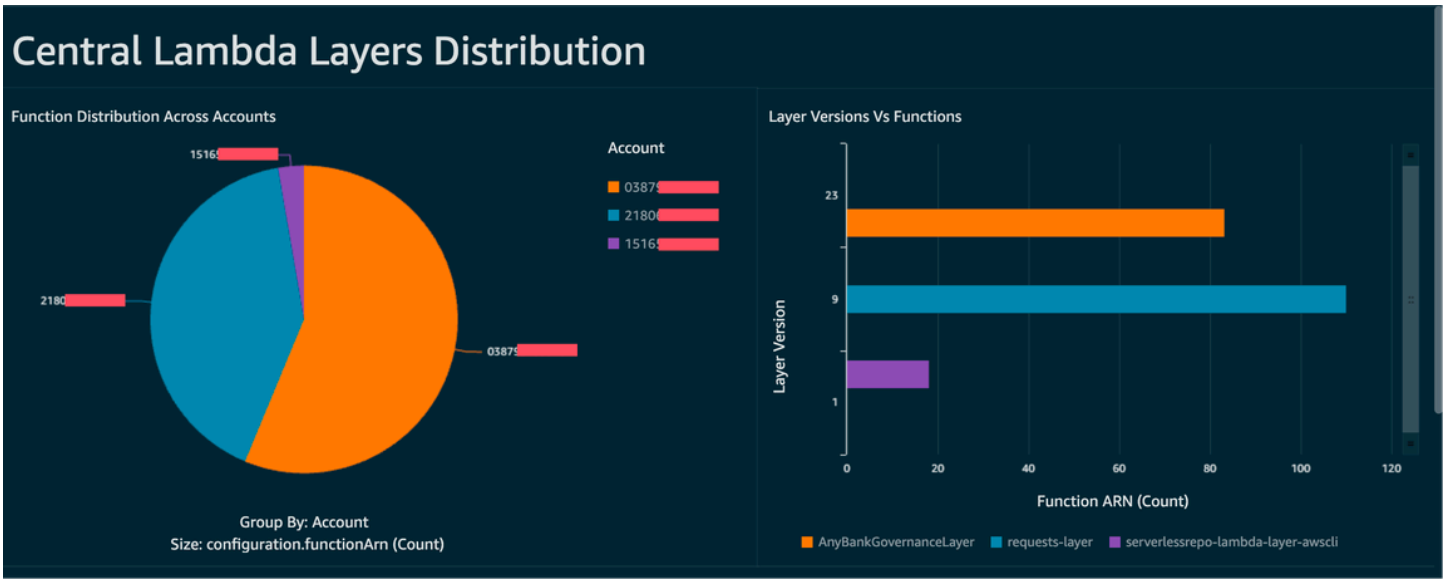
Completed — Time in queue: 127 ms — Run time: 1.803 sec — Data scanned: 239.40 KB

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

Con i dati di AWS Config aggregati in tutta l'organizzazione, puoi quindi creare una dashboard utilizzando Amazon QuickSight. Importando i risultati di Athena in Amazon QuickSight, puoi visualizzare in che misura le funzioni Lambda aderiscono alla regola della versione di livello. Questa dashboard può evidenziare le risorse conformi e non conformi. Ciò ti aiuta a determinare la policy di applicazione, come indicato nella sezione successiva. L'immagine seguente è un esempio di dashboard che riporta la distribuzione delle versioni di livello applicate alle funzioni all'interno dell'organizzazione.



Fase 3: implementazione e applicazione

Ora, come opzione, puoi abbinare la regola della versione di livello che hai creato nella fase 1 con un'azione di correzione attraverso un documento di automazione Systems Manager, creato come script Python scritto con AWS SDK per Python (Boto3). Lo script effettua una chiamata all'operazione

API UpdateFunctionConfiguration per ogni funzione Lambda e aggiorna la configurazione della funzione con il nuovo ARN di livello. In alternativa, potresti fare in modo che lo script invii una richiesta pull al repository del codice per aggiornare l'ARN del livello. In questo modo anche le future implementazioni di codice vengono aggiornate con l'ARN di livello corretto.

Firma del codice Lambda con AWS Signer

AWS Signer è un servizio di firma del codice completamente gestito che consente di convalidare il codice rispetto a una firma digitale per confermare che il codice sia inalterato e provenga da un editore affidabile. AWS Signer può essere utilizzato insieme a AWS Lambda per verificare che le funzioni e i livelli siano inalterati prima dell'implementazione negli ambienti AWS. Ciò protegge l'organizzazione da malintenzionati che potrebbero aver ottenuto credenziali per creare nuove funzioni o aggiornare quelle esistenti.

Per impostare la firma del codice per le funzioni Lambda, comincia creando un bucket S3 con il controllo delle versioni abilitato. Dopodiché, crea un profilo di firma con AWS Signer, specifica Lambda come piattaforma e quindi specifica un periodo di giorni in cui il profilo di firma sarà valido. Esempio:

```
Signer:
  Type: AWS::Signer::SigningProfile
  Properties:
    PlatformId: AWSLambda-SHA384-ECDSA
    SignatureValidityPeriod:
      Type: DAYS
      Value: !Ref pValidDays
```

Quindi utilizza il profilo di firma e crea una configurazione di firma con Lambda. Devi specificare che cosa fare quando la configurazione di firma rileva un artefatto che non corrisponde alla firma digitale prevista: avvisare (ma consentire l'implementazione) o imporre (e bloccare l'implementazione). L'esempio seguente è configurato per imporre e bloccare le implementazioni.

```
SigningConfig:
  Type: AWS::Lambda::CodeSigningConfig
  Properties:
    AllowedPublishers:
      SigningProfileVersionArns:
        - !GetAtt Signer.ProfileVersionArn
    CodeSigningPolicies:
      UntrustedArtifactOnDeployment: Enforce
```

Ora ha configurato AWS Signer con Lambda per bloccare le implementazioni non attendibili. Supponiamo che tu abbia finito di codificare una richiesta di funzionalità e che ora stia aspettando di implementare la funzione. La prima fase consiste nel comprimere in formato zip il codice con le

dipendenze appropriate e quindi nel firmare l'artefatto utilizzando il profilo di firma che hai creato. Puoi farlo caricando l'artefatto zip nel bucket S3 e quindi avviando un processo di firma.

```
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
```

Otterrai un output come il seguente, dove `jobId` è l'oggetto creato nel bucket e nel prefisso di destinazione e `jobOwner` è l'ID dell'Account AWS a 12 cifre in cui il processo è stato eseguito.

```
{
    "jobId": "87a3522b-5c0b-4d7d-b4e0-4255a8e05388",
    "jobOwner": "111122223333"
  }
```

Ora puoi implementare la funzione utilizzando l'oggetto S3 firmato e la configurazione di firma del codice che hai creato.

```
  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
```

In alternativa, puoi testare l'implementazione di una funzione con l'artefatto zip di origine non firmato autentico. L'implementazione dovrebbe avere esito negativo con il seguente messaggio di errore:

```
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.
```

Se stai creando e implementando le funzioni con AWS Serverless Application Model (AWS SAM), il comando del pacchetto gestisce il caricamento dell'artefatto zip in S3, avvia anche il processo di firma e ottiene l'artefatto firmato. Puoi eseguire questa operazione con il comando e i parametri riportati di seguito:

```
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 ti aiuta a verificare che gli artefatti zip implementati negli account siano affidabili per l'implementazione. Puoi includere il processo di cui sopra nelle pipeline CI/CD e richiedere che tutte le funzioni abbiano una configurazione di firma del codice associata utilizzando le tecniche descritte negli argomenti precedenti. Utilizzando la firma del codice con le implementazioni delle funzioni Lambda, impedisci a malintenzionati che potrebbero aver ottenuto le credenziali per creare o aggiornare le funzioni di iniettare codice dannoso nelle funzioni.

Automatizzare le valutazioni di sicurezza per Lambda con Amazon Inspector

Amazon Inspector è un servizio di gestione delle vulnerabilità che scansiona continuamente i carichi di lavoro alla ricerca di vulnerabilità del software ed esposizione alla rete non intenzionale. Amazon Inspector crea un esito che descrive la vulnerabilità, identifica la risorsa interessata, valuta la gravità della vulnerabilità e fornisce indicazioni per la correzione.

Il supporto di Amazon Inspector fornisce valutazioni continue e automatizzate delle vulnerabilità di sicurezza per le funzioni e i livelli Lambda. Amazon Inspector offre due tipi di scansione per Lambda:

- Scansione standard Lambda (impostazione predefinita): esegue la scansione delle dipendenze dell'applicazione all'interno di una funzione Lambda e dei relativi livelli alla ricerca di vulnerabilità dei pacchetti.
- Scansione del codice Lambda: esegue la scansione del codice dell'applicazione personalizzato nelle funzioni e nei livelli alla ricerca di vulnerabilità del codice. Puoi attivare la scansione standard Lambda da sola o insieme alla scansione del codice Lambda.

Per abilitare Amazon Inspector, accedi alla console Amazon Inspector, espandi la sezione Impostazioni e scegli Gestione dell'account. Nella scheda Account, scegli Attiva, quindi seleziona una delle opzioni di scansione.

Puoi abilitare Amazon Inspector per più account e delegare le autorizzazioni a gestire Amazon Inspector per l'organizzazione ad account specifici durante la configurazione di Amazon Inspector. Durante l'abilitazione, devi concedere le autorizzazioni ad Amazon Inspector creando il ruolo: `AWSServiceRoleForAmazonInspector2`. La console Amazon Inspector ti consente di creare questo ruolo utilizzando un'opzione con un solo clic.

Per la scansione standard Lambda, Amazon Inspector avvia scansioni di vulnerabilità delle funzioni Lambda nelle situazioni seguenti:

- Non appena Amazon Inspector rileva una funzione Lambda esistente.
- Quando si implementa una nuova funzione Lambda.
- Quando si implementa un aggiornamento al codice dell'applicazione o alle dipendenze di una funzione Lambda esistente o dei relativi livelli.

- Ogni volta che Amazon Inspector aggiunge un nuovo elemento di vulnerabilità ed esposizioni comuni (common vulnerabilities and exposures, CVE) al suo database e tale CVE è pertinente alla funzione.

Per la scansione del codice Lambda, Amazon Inspector valuta il codice dell'applicazione della funzione Lambda utilizzando ragionamento automatico e machine learning che analizzano il codice dell'applicazione per quanto riguarda la conformità di sicurezza generale. Se rileva una vulnerabilità nel codice dell'applicazione della funzione Lambda, Amazon Inspector produce un esito di Vulnerabilità del codice dettagliato. Per un elenco di possibili rilevamenti, consulta Amazon CodeGuru Detector Library.

Per visualizzare gli esiti, accedi alla console Amazon Inspector. Nel menu Esiti, seleziona Per funzione Lambda per visualizzare i risultati della scansione di sicurezza eseguita sulle funzioni Lambda.

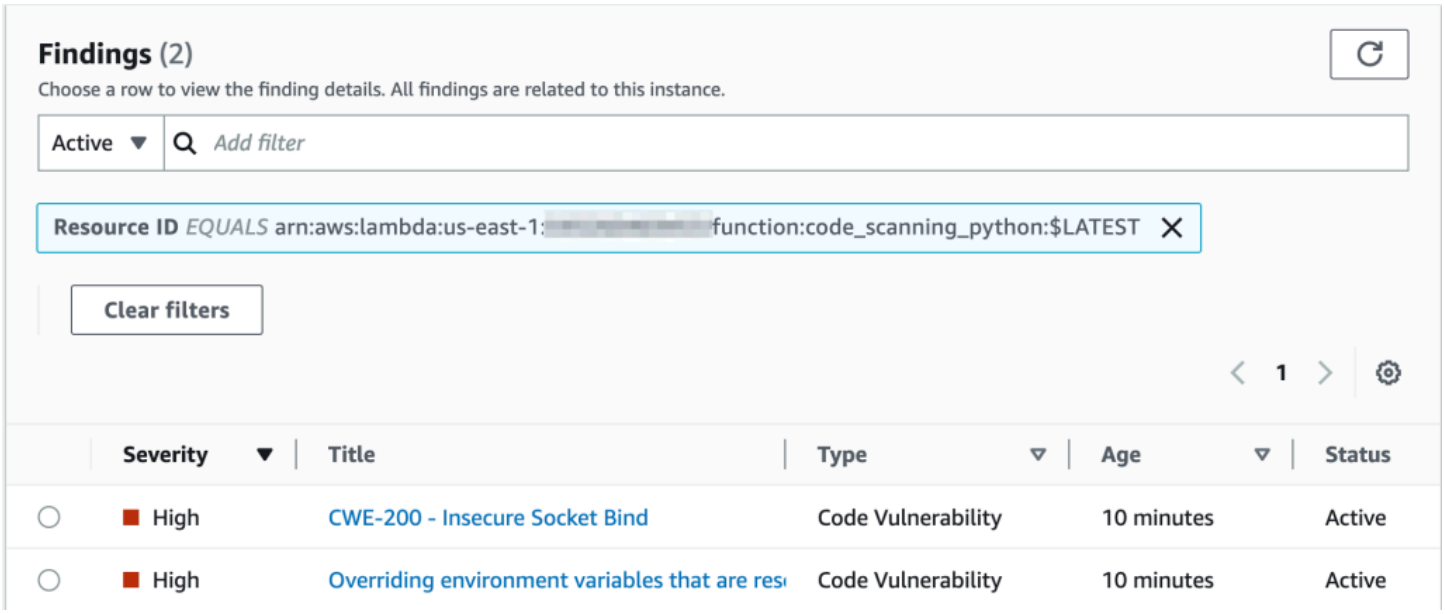
Per escludere una funzione Lambda dalla scansione standard, etichetta la funzione con la seguente coppia chiave-valore:

- `Key:InspectorExclusion`
- `Value:LambdaStandardScanning`

Per escludere una funzione Lambda dalle scansioni del codice, etichetta la funzione con la seguente coppia chiave-valore:

- `Key:InspectorCodeExclusion`
- `Value:LambdaCodeScanning`

Ad esempio, come mostrato nell'immagine seguente, Amazon Inspector rileva in automatico le vulnerabilità e classifica gli esiti di tipo Vulnerabilità del codice, il che indica che la vulnerabilità si trova nel codice della funzione e non in una delle librerie dipendenti dal codice. Puoi controllare questi dettagli per una funzione specifica o per più funzioni contemporaneamente.



Puoi approfondire ciascuno degli esiti e scoprire come risolvere il problema.

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.

Mentre lavori con le funzioni Lambda, assicurati di rispettare le convenzioni di denominazione delle funzioni Lambda. Per ulteriori informazioni, consulta Utilizzo delle variabili di ambiente Lambda.

La responsabilità sui suggerimenti di riparazione che accetti è tua. Esamina sempre i suggerimenti di riparazione prima di accettarli. Per garantire che il codice funzioni come previsto, potrebbe essere necessario apportare modifiche ai suggerimenti di riparazione.

Implementazione dell'osservabilità per la sicurezza e la conformità Lambda

AWS Config è uno strumento utile per trovare e correggere risorse Serverless non conformi AWS . Ogni modifica apportata alle risorse serverless viene registrata in. AWS Config Inoltre, AWS Config consente di archiviare i dati delle istantanee di configurazione su S3. Puoi usare Amazon Athena e Amazon Quick Suite per creare dashboard e visualizzare i dati. AWS Config In Rilevare implementazioni e configurazioni Lambda non conformi con AWS Config, abbiamo discusso come visualizzare una determinata configurazione quali i livelli Lambda. Il presente argomento approfondisce tali concetti.

Visibilità sulle configurazioni Lambda

Puoi utilizzare le query per richiamare configurazioni importanti come Account AWS ID, regione, configurazione di AWS X-Ray tracciamento, configurazione VPC, dimensione della memoria, runtime e tag. Di seguito è riportato un esempio di query che puoi utilizzare per ottenere queste informazioni da 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
```

Puoi utilizzare la query per creare una dashboard di Quick Suite e visualizzare i dati. Per aggregare i dati di configurazione AWS delle risorse, creare tabelle in Athena e creare dashboard di Quick Suite sui dati di Athena, consulta Visualizing AWS Config data using Athena and Amazon Quick Suite on the Cloud Operations and Management blog. AWS In particolare, questa query recupera anche le informazioni dei tag per le funzioni. Ciò consente di ottenere maggiori approfondimenti sui carichi di lavoro e sugli ambienti, soprattutto se impieghi tag personalizzati.



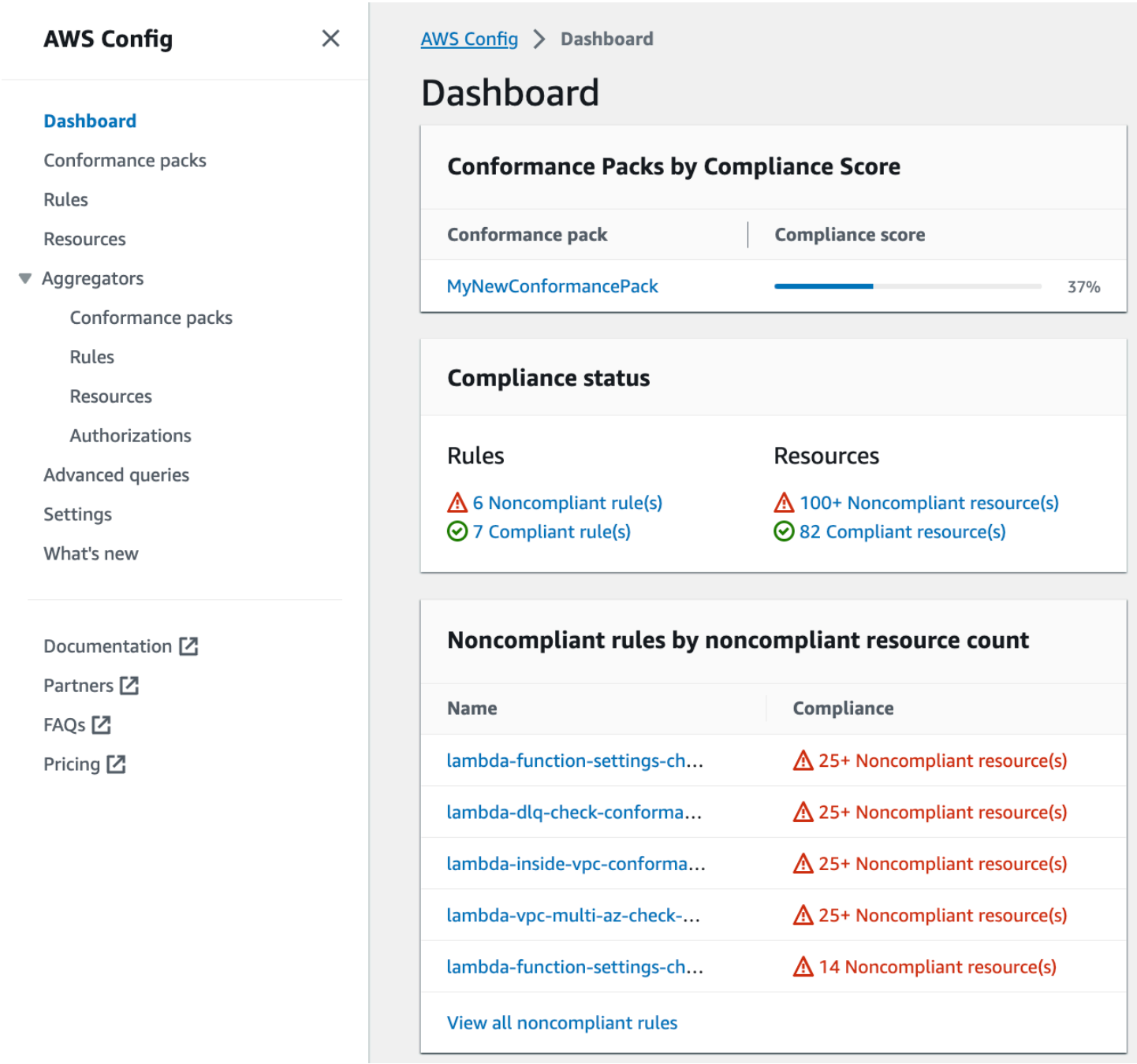
Per ulteriori informazioni sulle azioni che puoi intraprendere, consulta la sezione Risoluzione degli esiti di osservabilità più avanti in questo argomento.

Visibilità sulla conformità Lambda

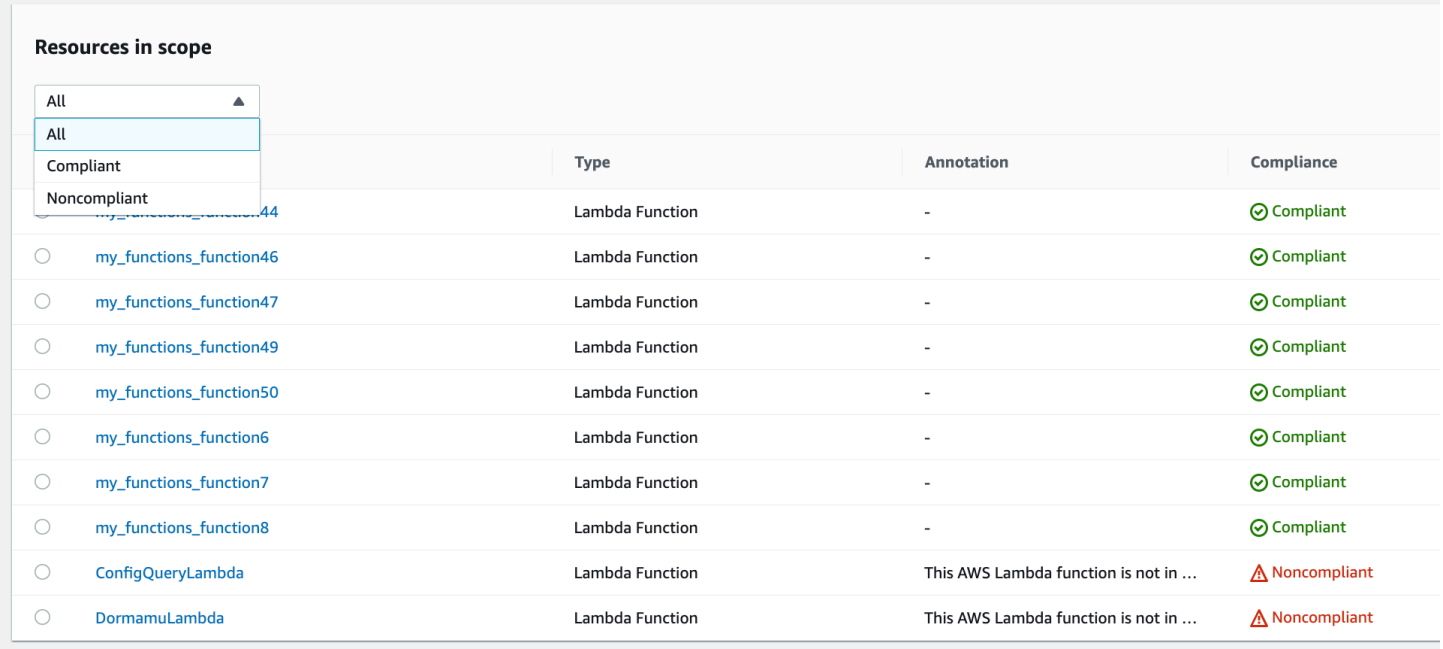
Con i dati generati da AWS Config, puoi creare dashboard a livello di organizzazione per monitorare la conformità. Ciò consente il tracciamento e il monitoraggio coerenti di:

- Pacchetti di conformità per punteggio di conformità

- Regole per risorse non conformi
- Compliance status (Stato di conformità)

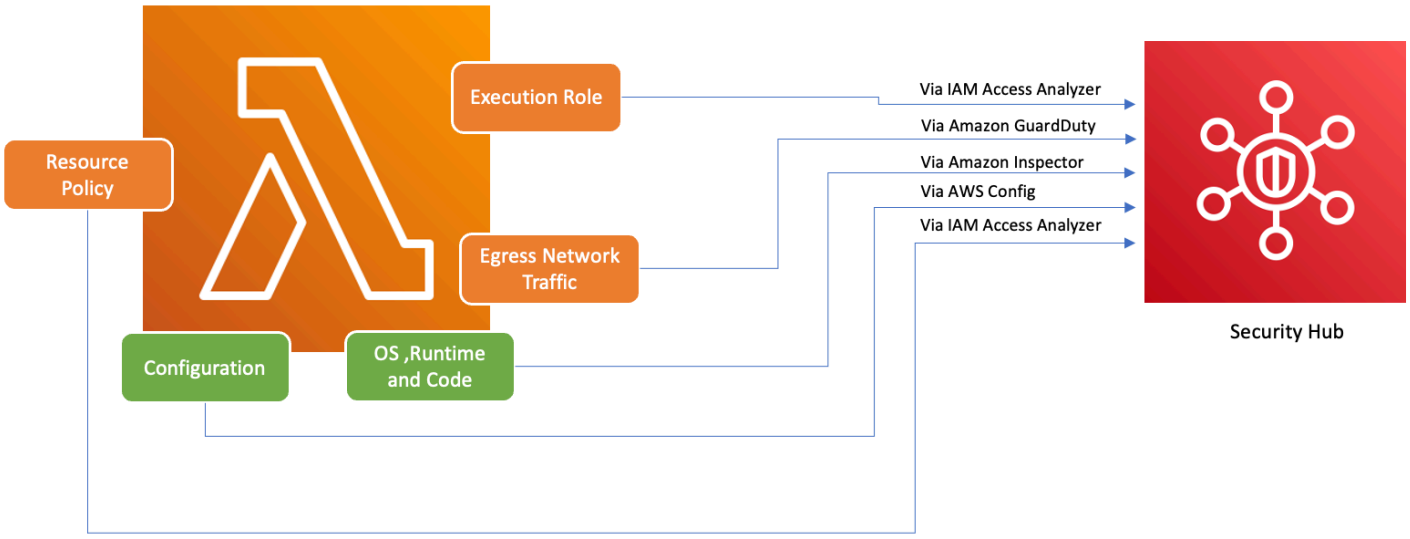


Controlla ogni regola per identificare le risorse non conformi alla stessa. Ad esempio, se l'organizzazione impone che tutte le funzioni Lambda siano associate a un VPC e se è stata implementata AWS Config una regola per identificare la conformità, è possibile selezionare `lambda-inside-vpc` la regola nell'elenco precedente.



Per ulteriori informazioni sulle azioni che puoi intraprendere, consulta la sezione [Risoluzione degli esiti di osservabilità](#) di seguito.

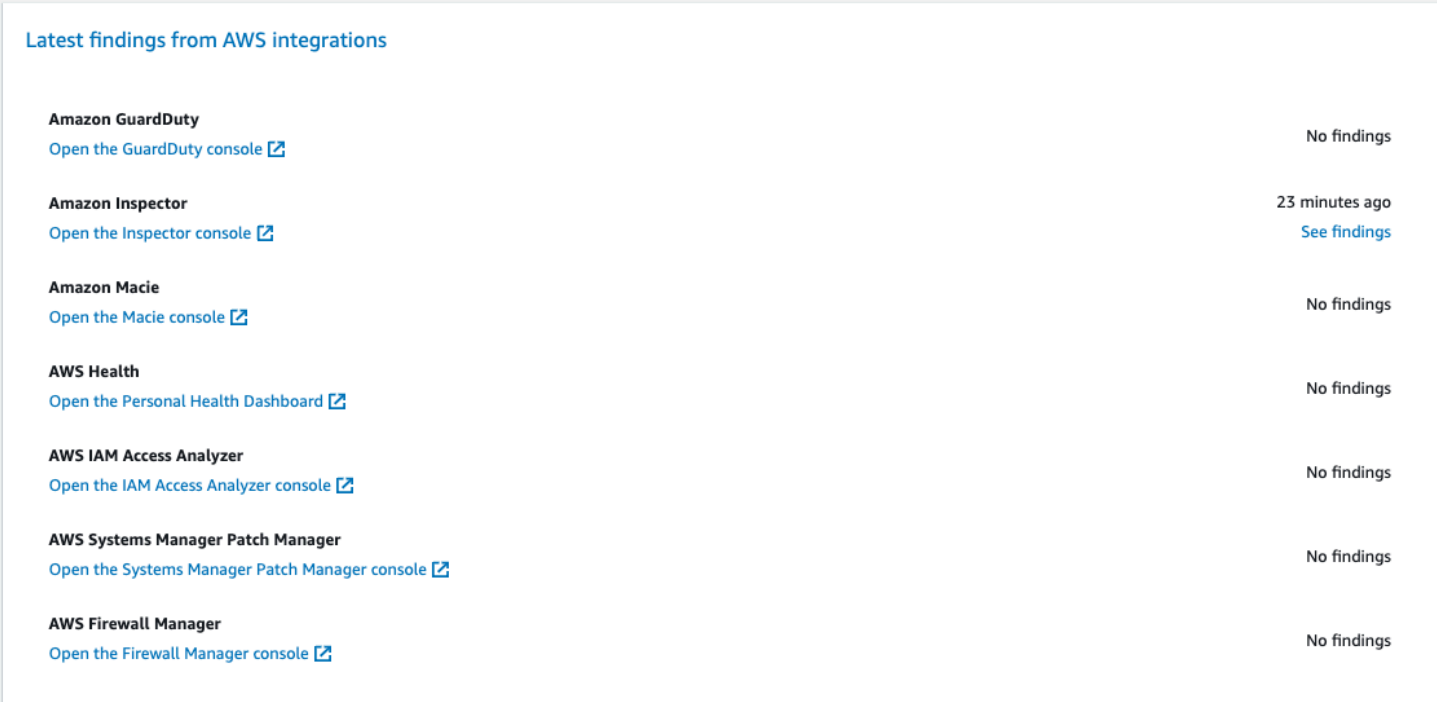
Visibilità nei limiti delle funzioni Lambda utilizzando Security Hub CSPM



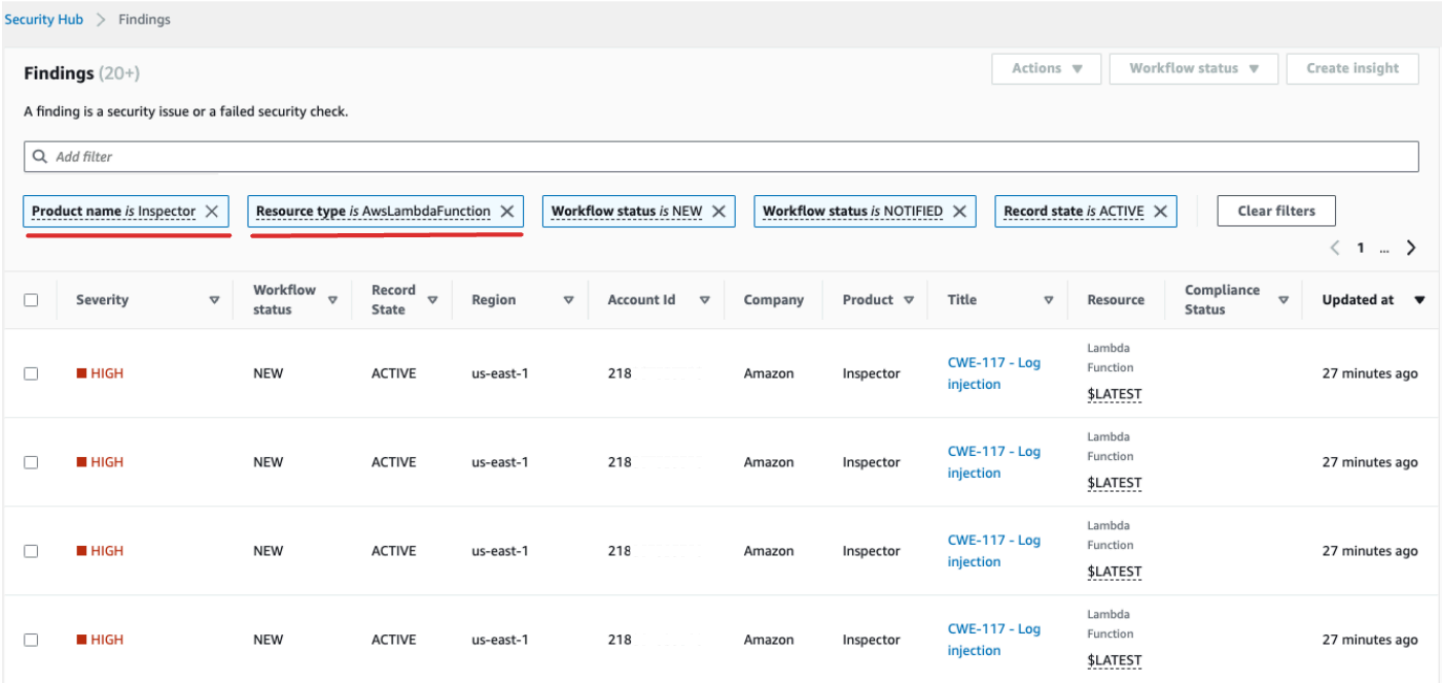
Per garantire che AWS i servizi, tra cui Lambda, vengano utilizzati in modo sicuro, sono state AWS introdotte le Foundational Security Best Practices v1.0.0. Questo insieme di best practice fornisce linee guida chiare per proteggere risorse e dati nell' AWS ambiente, sottolineando l'importanza di mantenere un solido livello di sicurezza. A ciò AWS Security Hub CSPM si aggiunge l'offerta di un centro unificato per la sicurezza e la conformità. Aggrega, organizza e dà priorità ai risultati di

sicurezza provenienti da più servizi come AWS Amazon Inspector e Amazon. AWS Identity and Access Management Access Analyzer GuardDuty

Se hai Security Hub CSPM, Amazon Inspector, IAM Access Analyzer GuardDuty e sei abilitato all'interno della tua organizzazione AWS , Security Hub CSPM aggrega automaticamente i risultati di questi servizi. Prendiamo ad esempio Amazon Inspector. Utilizzando Security Hub CSPM, puoi identificare in modo efficiente le vulnerabilità del codice e dei pacchetti nelle funzioni Lambda. Nella console Security Hub CSPM, vai alla sezione inferiore denominata Ultimi risultati delle integrazioni. AWS Qui puoi visualizzare e analizzare i risultati provenienti da vari servizi integrati. AWS



Per visualizzare i dettagli, scegli il link Vedi risultati nella seconda colonna. Viene visualizzato un elenco di risultati filtrati per prodotto, ad esempio Amazon Inspector. Per limitare la ricerca alle funzioni Lambda, imposta `ResourceType` su `AwsLambdaFunction`. Vengono visualizzati gli esiti di Amazon Inspector relativi alle funzioni Lambda.



Infatti GuardDuty, è possibile identificare modelli di traffico di rete sospetti. Tali anomalie potrebbero suggerire l'esistenza di codice potenzialmente dannoso all'interno della funzione Lambda.

Con Sistema di analisi degli accessi IAM, puoi controllare le policy, in particolare quelle con istruzioni condizionali che concedono l'accesso alle funzioni a entità esterne. Inoltre, IAM Access Analyzer valuta le autorizzazioni impostate quando si utilizza l'AddPermissionoperazione nell'API Lambda insieme a un. `EventSourceToken`

Risoluzione degli esiti di osservabilità

Data l'ampia gamma di configurazioni possibili per le funzioni Lambda e i relativi requisiti distinti, una soluzione di automazione standardizzata per la riparazione potrebbe non essere adatta a ogni situazione. Inoltre, le modifiche vengono implementate in modo diverso nei vari ambienti. Se riscontri una configurazione che sembra non conforme, tieni presente le seguenti linee guida:

1. Strategia di assegnazione tag

   Consigliamo di implementare una strategia di assegnazione tag completa. A ogni funzione Lambda si dovrebbero assegnare tag con informazioni chiave come le seguenti:

   - Proprietario: la persona o il team responsabile della funzione.
   - Ambiente: produzione, staging, sviluppo o ambiente di sperimentazione (sandbox).
   - Applicazione: il contesto più ampio a cui appartiene la funzione, se applicabile.

2. Contatto del proprietario

   Anziché automatizzare le modifiche sostanziali (come la regolazione della configurazione del VPC), contatta in modo proattivo i proprietari delle funzioni non conformi (identificati dal tag del proprietario) fornendo loro il tempo sufficiente per adottare una delle misure seguenti:

   - Regolare le configurazioni non conformi sulle funzioni Lambda.
   - Fornire una spiegazione e richiedere un'eccezione oppure perfezionare gli standard di conformità.

3. Mantenimento di un database di gestione della configurazione (CMDB)

   Sebbene i tag possano fornire un contesto immediato, mantenere un CMDB centralizzato può fornire maggiori approfondimenti. Il database può contenere informazioni più granulari in merito a ogni funzione Lambda, alle relative dipendenze e altri metadati critici. Un CMDB è una risorsa di valore inestimabile per l'audit, i controlli di conformità e l'identificazione dei proprietari delle funzioni.

Poiché il panorama dell'infrastruttura serverless è in continua evoluzione, è essenziale adottare un atteggiamento proattivo nei confronti del monitoraggio. Con strumenti come AWS Config Security Hub CSPM e Amazon Inspector, è possibile identificare rapidamente potenziali anomalie o configurazioni non conformi. Tuttavia, gli strumenti da soli non possono garantire la conformità totale o configurazioni ottimali. È fondamentale abbinare tali strumenti a processi e best practice ben documentati.

- Ciclo di feedback: una volta intraprese le misure correttive, assicurati che vi sia un ciclo di feedback. Ciò significa riesaminare su base periodica le risorse non conformi per confermare se sono state aggiornate o se presentano ancora gli stessi problemi.
- Documentazione: documenta sempre le osservazioni, le azioni intraprese e le eventuali eccezioni concesse. Una documentazione adeguata non solo aiuta durante gli audit, ma contribuisce anche a migliorare il processo per ottenere una maggiore conformità e sicurezza in futuro.
- Formazione e consapevolezza: assicurati che tutte le parti interessate, in particolare i responsabili delle funzioni Lambda, ricevano una formazione periodica e siano informate su best practice, policy dell'organizzazione e obblighi di conformità. Workshop, webinar o sessioni di formazione regolari possono dare un contributo notevole per garantire che tutti siano sulla stessa lunghezza d'onda in materia di sicurezza e conformità.

In conclusione, mentre gli strumenti e le tecnologie forniscono funzionalità solide per rilevare e segnalare potenziali problemi, l'elemento umano (comprensione, comunicazione, formazione e documentazione) rimane di cruciale importanza. Insieme, questi aspetti formano una combinazione molto efficace per garantire che le funzioni Lambda e l'infrastruttura in senso lato rimangano conformi, sicure e ottimizzate per le esigenze aziendali.

Convalida della conformità per AWS Lambda

Revisori di terze parti valutano la sicurezza e la conformità di AWS Lambda all'interno di molteplici programmi di conformità di AWS. Questi includono SOC, PCI, FedRAMP, HIPAA e altri.

Per un elenco di servizi AWS che rientrano nell'ambito di programmi di conformità specifici, consultare Servizi AWS coperti dal programma di compliance. Per informazioni generali, consultare Programmi per la conformità di AWS.

Puoi scaricare i report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta Download dei report in AWS Artifact.

La tua responsabilità di conformità durante l'utilizzo di Lambda è determinata dalla riservatezza dei dati, dagli obiettivi dell'azienda e dalle leggi e normative vigenti in materia. Puoi implementare controlli di governance per garantire che le funzioni Lambda della tua azienda soddisfino i requisiti di conformità. Per ulteriori informazioni, consulta Creare una strategia di governance per le funzioni e i livelli Lambda.

Resilienza nell' AWS Lambda

L'infrastruttura globale di AWSè basata su Regioni e zone di disponibilità AWS. AWS forniscono più zone di disponibilità fisicamente separate e isolate che sono connesse tramite reti altamente ridondanti, a bassa latenza e con throughput elevato. Con le zone di disponibilità, è possibile progettare e gestire applicazioni e database che eseguono il failover automatico tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture tradizionali a data center singolo o multiplo.

Per ulteriori informazioni sulle Regioni e le Zone di disponibilità AWS, consulta Infrastruttura globale di AWS.

Oltre all'infrastruttura globale di AWS, Lambda offre numerose funzionalità per supportare la resilienza dei dati e le esigenze di backup.

- Gestione delle versioni – È possibile usare la gestione delle versioni in Lambda per salvare il codice e la configurazione in fase di sviluppo. Insieme all'utilizzo delgi alias, è possibile utilizzare la gestione delle versioni per eseguire distribuzioni blue/green e progressive. Per informazioni dettagliate, vedi Gestire le versioni delle funzioni Lambda.
- Dimensionamento – Quando la funzione riceve una richiesta durante l'elaborazione di una richiesta precedente, Lambda avvia un'altra istanza della funzione per gestire l'aumento del carico. Lambda

dimensiona automaticamente per gestire 1.000 esecuzioni simultanee per regione, un quota che può essere aumentata se necessario. Per informazioni dettagliate, vedi Informazioni sulla scalabilità della funzione Lambda.

- Elevata disponibilità – Lambda esegue la funzione in più zone di disponibilità per assicurare che sia disponibile per elaborare eventi in caso di interruzione del servizio in una specifica zona. Se si configura la tua funzione affinché si connetta a un cloud privato virtuale (VPC) nel proprio account, specificare le sottoreti in più zone di disponibilità per garantire un'elevata disponibilità. Per informazioni dettagliate, vedi Consentire alle funzioni Lambda l'accesso alle risorse in un Amazon VPC.
- Simultaneità riservata – Per garantire che la funzione sia sempre in grado di ridimensionare le risorse per gestire le richieste aggiuntive, è possibile riservare la simultaneità. Impostare la simultaneità riservata per una funzione garantisce che questa possa ridimensionarsi fino a un dato numero di chiamate simultanee, senza tuttavia superarlo. In questo modo non andranno perse delle richieste a causa di altre funzioni onerose in termini di disponibilità simultanea. Per informazioni dettagliate, vedi Configurazione della simultaneità riservata per una funzione.
- Tentativi ripetuti – Per le invocazioni asincrone e un sottoinsieme di invocazioni attivate da altri servizi, Lambda esegue automaticamente dei nuovi tentativi in caso di errore con specifici ritardi tra i tentativi successivi. Altri client e Servizi AWS che invocano le funzioni in modo sincrono sono responsabili dell'esecuzione di tentativi ripetuti. Per informazioni dettagliate, vedi Informazioni sul comportamento relativo ai nuovi tentativi in Lambda.
- Coda DLQ – In caso di invocazioni asincrone, è possibile configurare Lambda affinché invii le richieste a una dead-letter queue se tutti i tentativi ripetuti non vanno a buon fine. Una coda DLQ è un argomento Amazon SNS o una coda Amazon SQS che riceve gli eventi a scopo di risoluzione dei problemi o rielaborazione. Per informazioni dettagliate, vedi Aggiunta di una coda DLQ.

Sicurezza dell'infrastruttura in AWS Lambda

Come servizio gestito, AWS Lambda è protetto dalla sicurezza di rete globale AWS. Per informazioni sui servizi di sicurezza AWS e su come AWS protegge l'infrastruttura, consulta la pagina Sicurezza del cloud AWS. Per progettare l'ambiente AWS utilizzando le best practice per la sicurezza dell'infrastruttura, consulta la pagina Protezione dell'infrastruttura nel Pilastro della sicurezza di AWS Well-Architected Framework.

Utilizza le chiamate API pubblicate da AWS per accedere a Lambda tramite la rete. I client devono supportare quanto segue:

- Transport Layer Security (TLS). È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Suite di cifratura con Perfect Forward Secrecy (PFS), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità.

Protezione dei carichi di lavoro con endpoint pubblici

Per i carichi di lavoro accessibili pubblicamente, AWS offre una serie di funzionalità e servizi che possono aiutare a mitigare determinati rischi. In questa sezione viene trattata l'autenticazione e l'autorizzazione degli utenti delle applicazioni e la protezione degli endpoint delle API.

Autenticazione e autorizzazione

L'autenticazione si riferisce all'identità mentre l'autorizzazione si riferisce alle operazioni. Usa l'autenticazione per controllare chi può richiamare una funzione Lambda, quindi usa l'autorizzazione per controllare cosa può fare. Per molte applicazioni, IAM è sufficiente per gestire entrambi i meccanismi di controllo.

Per le applicazioni con utenti esterni, come le applicazioni Web o per dispositivi mobili, è comune utilizzare JSON Web Tokens (JWT) per gestire l'autenticazione e l'autorizzazione. A differenza della tradizionale gestione delle password basata su server, i JWT vengono trasmessi dal client a ogni richiesta. Sono un modo sicuro dal punto di vista della crittografia per verificare l'identità e le affermazioni utilizzando i dati trasmessi dal client. Per le applicazioni basate su Lambda, ciò consente di proteggere ogni chiamata a ciascun endpoint API senza fare affidamento su un server centrale per l'autenticazione.

Puoi implementare JWTS con Amazon Cognito, un servizio di directory utente in grado di gestire la registrazione, l'autenticazione, il ripristino dell'account e altre operazioni comuni di gestione degli account. Framework Amplify fornisce librerie per semplificare l'integrazione di questo servizio nell'applicazione frontend. Puoi anche utilizzare servizi di partner di terze parti come Auth0.

Dato il ruolo fondamentale di sicurezza di un servizio di provider di identità, è importante utilizzare strumenti professionali per salvaguardare l'applicazione. Non è consigliabile creare servizi personalizzati per gestire l'autenticazione o l'autorizzazione. Qualsiasi vulnerabilità nelle librerie personalizzate può avere implicazioni significative per la sicurezza del carico di lavoro e dei relativi dati.

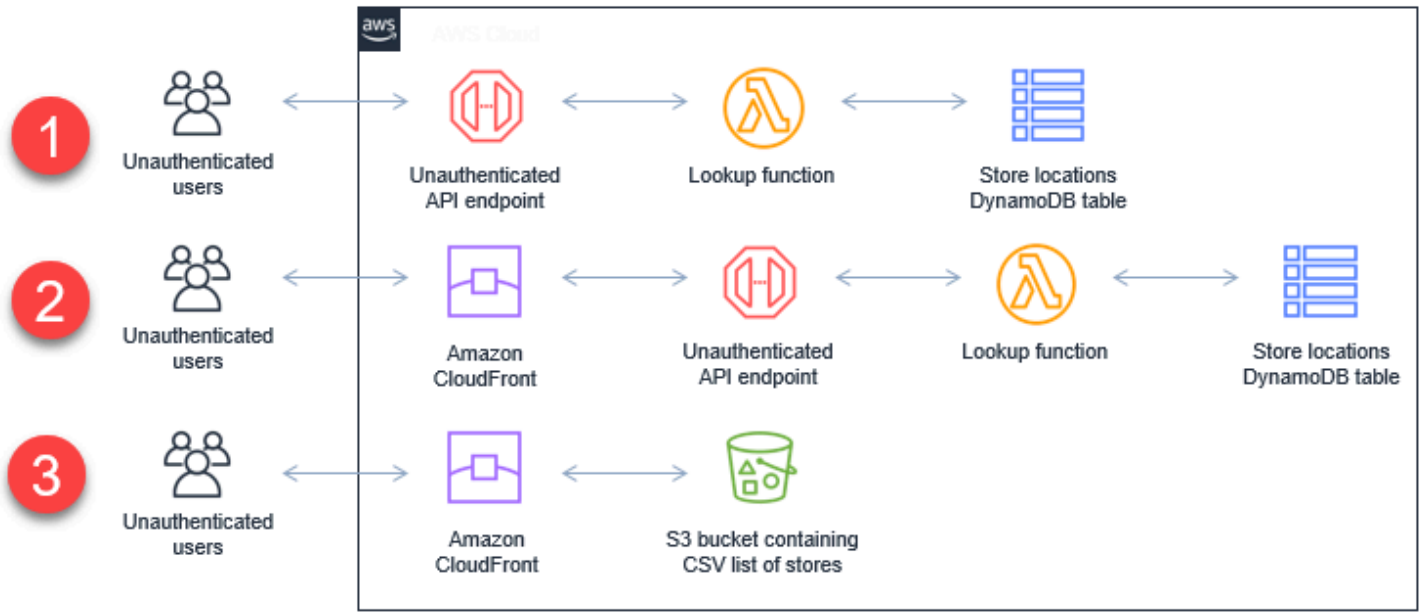
Protezione degli endpoint API

Per le applicazioni serverless, il modo preferito per servire pubblicamente un'applicazione di backend consiste nell'utilizzare Gateway Amazon API. Ciò può aiutarti a proteggere un'API da utenti malintenzionati o da picchi di traffico.

API Gateway offre due tipi di endpoint per gli sviluppatori serverless: REST API e API HTTP. Entrambi supportano l'autorizzazione tramite Lambda, IAM o Amazon Cognito. Quando si utilizza IAM o Amazon Cognito, le richieste in entrata vengono valutate e se mancano di un token richiesto o contengono un'autenticazione non valida, la richiesta viene rifiutata. Queste richieste non ti vengono addebitate e non vengono conteggiate ai fini delle quote di limitazione.

Le route API non autenticate sono accessibili da chiunque sulla rete Internet pubblica, quindi si consiglia di limitare l'uso di API non autenticate. Se è necessario utilizzare API non autenticate, è importante proteggerle da rischi comuni, come gli attacchi denial-of-service (DoS). L'applicazione di WAF a queste API può aiutare a proteggere l'applicazione da attacchi di iniezione SQL e cross-site scripting (XSS). API Gateway implementa anche la limitazione a livello di account AWS e per client quando vengono utilizzate le chiavi API.

In molti casi, la funzionalità fornita da un'API non autenticata può essere ottenuta con un approccio alternativo. Ad esempio, un'applicazione Web può fornire un elenco di negozi al dettaglio dei clienti da una tabella DynamoDB agli utenti che non hanno effettuato l'accesso. Questa richiesta può provenire da un'applicazione Web frontend o da qualsiasi altra fonte che richiama l'endpoint URL. Questo diagramma mette a confronto tre soluzioni:



1. Questa API non autenticata può essere chiamata da chiunque su Internet. In un attacco Denial of Service, è possibile esaurire i limiti di limitazione delle API, la simultaneità di Lambda o la capacità di lettura fornita da DynamoDB su una tabella sottostante.
2. Una distribuzione CloudFront davanti all'endpoint dell'API con una configurazione time-to-live (TTL) appropriata assorbirebbe la maggior parte del traffico in un attacco DoS, senza modificare la soluzione sottostante per il recupero dei dati.
3. In alternativa, per i dati statici che cambiano raramente, la distribuzione CloudFront potrebbe servire i dati da un bucket Amazon S3.

Utilizzo della firma del codice per verificare l'integrità del codice con Lambda

La firma del codice consente di garantire che solo il codice attendibile venga eseguito nelle funzioni Lambda. Utilizzando AWS Signer, puoi creare pacchetti di codice con firma digitale per le tue funzioni. Quando aggiungi una configurazione di firma del codice a una funzione, Lambda verifica che tutte le nuove implementazioni di codice siano firmate da una fonte attendibile. Poiché i controlli di convalida della firma del codice vengono eseguiti al momento dell'implementazione, non vi è alcun impatto sull'esecuzione delle funzioni.

> **Important**
>
> Le configurazioni di firma del codice impediscono solo nuove implementazioni di codice non firmato. Se aggiungi una configurazione di firma del codice a una funzione esistente con codice non firmato, tale codice continua a funzionare finché non distribuisci un nuovo pacchetto di codice.

Quando attivi la firma del codice per una funzione, tutti i livelli aggiunti alla funzione devono essere firmati anche da uno dei profili di firma consentiti.

Non sono previsti costi aggiuntivi per l'utilizzo AWS Signer o la firma del codice. AWS Lambda

Convalida della firma

Lambda esegue i seguenti controlli di convalida quando si distribuisce un pacchetto di codice firmato alla funzione:

1. Integrità: convalida che il pacchetto di codice non sia stato modificato dopo la firma. Lambda confronta l'hash del pacchetto con l'hash della firma.
2. Scadenza: convalida che la firma del pacchetto di codice non sia scaduta.
3. Mancata corrispondenza: convalida che il pacchetto di codice sia firmato con un profilo di firma consentiti.
4. Revoca: convalida che la firma del pacchetto di codice non sia stata revocata.

Quando crei una configurazione di firma del codice, puoi utilizzare il UntrustedArtifactOnDeploymentparametro per specificare come Lambda deve rispondere se i controlli di scadenza, mancata corrispondenza o revoca falliscono. Puoi scegliere una di queste azioni:

- `Warn`: è l'impostazione predefinita. Lambda consente l'implementazione del pacchetto di codice, ma emette un avviso. Lambda emette un nuovo CloudWatch parametro Amazon (`SignatureValidationErrors`) e memorizza anche l'avviso nel registro. CloudTrail
- `Enforce`: Lambda emette un avviso (lo stesso dell'azione `Warn`) e blocca l'implementazione del pacchetto di codice.

Argomenti

- Creazione di configurazioni di firma del codice per Lambda
- Configurazione delle policy IAM per le configurazioni di firma del codice Lambda
- Utilizzo di tag nelle configurazioni di firma del codice

Creazione di configurazioni di firma del codice per Lambda

Per abilitare la firma del codice per una funzione, creare una configurazione di firma del codice e collegarla alla funzione. Una configurazione di firma del codice definisce un elenco di profili di firma consentiti e le operazioni delle policy da eseguire in caso di esito negativo dei controlli di convalida.

> **Note**
>
> Le funzioni definite come immagini di container non supportano la firma del codice.

Sezioni

- Prerequisiti di configurazione
- Creazione delle configurazioni di firma del codice
- Abilitazione della firma del codice per una funzione

Prerequisiti di configurazione

Prima di configurare la firma del codice per una funzione Lambda, utilizzare AWS Signer per effettuare le seguenti operazioni:

- Creare uno o più profili di firma.
- Utilizzare un profilo di firma per creare un pacchetto di codice firmato per la funzione.

Creazione delle configurazioni di firma del codice

Una configurazione di firma del codice definisce un elenco di profili di firma consentiti e la policy di convalida della firma.

Per creare una configurazione di firma del codice (console)

1. Aprire la pagina Configurazioni di firma del codice della console Lambda.
2. Scegli Crea configurazione.
3. In Description (Descrizione), immettere un nome descrittivo per la configurazione.
4. In Signing profiles (Profili di firma) aggiungere fino a 20 profili di firma alla configurazione.
    a. Per l'ARN della versione del profilo di firma, scegliere l'ARN (Amazon Resource Name) di una versione del profilo oppure inserire l'ARN.
    b. Per aggiungere un profilo di firma aggiuntivo, scegliere Add signing profiles (Aggiungi profili di firma).
5. In Signature validation policy (Policy di convalida della firma), scegliere Warn (Avvisa) o Enforce (Applica).
6. Scegli Crea configurazione.

Abilitazione della firma del codice per una funzione

Per abilitare la firma del codice per una funzione, aggiungi una configurazione di firma del codice alla funzione.

> **Important**
>
> Le configurazioni di firma del codice impediscono solo nuove implementazioni di codice non firmato. Se aggiungi una configurazione di firma del codice a una funzione esistente con codice non firmato, tale codice continua a funzionare finché non distribuisci un nuovo pacchetto di codice.

Per associare una configurazione di firma del codice a una funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere la funzione per la quale si desidera abilitare la firma del codice.
3. Apri la scheda Configurazione.
4. Scorri verso il basso e scegli Firma del codice.
5. Scegli Modifica.
6. In Edit code signing (Modifica della firma del codice), scegliere una configurazione di firma del codice per questa funzione.
7. Selezionare Salva.

Configurazione delle policy IAM per le configurazioni di firma del codice Lambda

Per concedere l'autorizzazione a un utente di accedere alle operazioni dell'API di firma del codice Lambda, allegare una o più istruzioni delle policy alla policy dell'utente. Per ulteriori informazioni sulle policy utente, consulta Policy IAM basate sull'identità per Lambda.

La seguente istruzione di policy di esempio concede l'autorizzazione per creare, aggiornare e recuperare le configurazioni di firma del codice.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": [
          "lambda:CreateCodeSigningConfig",
          "lambda:UpdateCodeSigningConfig",
          "lambda:GetCodeSigningConfig"
        ],
      "Resource": "*"
    }
  ]
}
```

Gli amministratori possono utilizzare la chiave di condizione `CodeSigningConfigArn` per specificare le configurazioni di firma del codice che gli sviluppatori devono utilizzare per creare o aggiornare le funzioni.

La seguente istruzione di policy di esempio concede l'autorizzazione per creare una funzione. La dichiarazione di policy include una condizione `lambda:CodeSigningConfigArn` per specificare la configurazione di firma del codice consentita. Lambda blocca le richieste `CreateFunction` API se il CodeSigningConfigArnparametro manca o non corrisponde al valore nella condizione.

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"
        }
      }
    }
  ]
}
```

Utilizzo di tag nelle configurazioni di firma del codice

Puoi contrassegnare le configurazioni di firma del codice per organizzare e gestire le risorse. I tag sono coppie chiave-valore a forma libera associate alle risorse supportate su Servizi AWS. Per ulteriori informazioni sui casi d'uso dei tag, consulta Strategie di tagging comuni nella Guida AWS alle risorse di etichettatura e all'editor di tag.

Puoi utilizzare l' AWS Lambda API per visualizzare e aggiornare i tag. Puoi anche visualizzare e aggiornare i tag mentre gestisci una configurazione specifica di firma del codice nella console Lambda.

Sections

- Autorizzazioni necessarie per lavorare con i tag
- Utilizzo di tag con la console Lambda
- Usare i tag con AWS CLI

Autorizzazioni necessarie per lavorare con i tag

Per consentire a un'identità AWS Identity and Access Management (IAM) (utente, gruppo o ruolo) di leggere o impostare tag su una risorsa, concedile le autorizzazioni corrispondenti:

- lambda: ListTags —Quando una risorsa ha dei tag, concedi questa autorizzazione a chiunque debba richiamarla`ListTags`. Per le funzioni con tag, questa autorizzazione è necessaria anche per `GetFunction`.
- lambda: TagResource —Concedi questa autorizzazione a chiunque abbia bisogno di chiamare `TagResource` o eseguire un tag durante la creazione.

Facoltativamente, valuta la possibilità di concedere anche l'UntagResourceautorizzazione lambda: per consentire `UntagResource` le chiamate alla risorsa.

Per ulteriori informazioni, consulta Policy IAM basate sull'identità per Lambda.

Utilizzo di tag con la console Lambda

Puoi utilizzare la console Lambda per creare configurazioni di firma del codice che hanno tag, aggiungere tag a configurazioni di firma del codice esistenti e filtrare le configurazioni di firma del codice per tag.

Per aggiungere un tag durante la creazione di una configurazione di firma del codice

1. Apri la pagina Configurazioni di firma del codice della console Lambda.
2. Dall'intestazione del riquadro dei contenuti, scegli Crea configurazione.
3. Nella sezione nella parte superiore del riquadro dei contenuti, configura la configurazione di firma del codice. Per ulteriori informazioni sull'impostazione delle configurazione di firma del codice, consulta the section called “Firma del codice”.
4. Nella sezione Tag, seleziona Aggiungi nuovo tag.
5. Nel campo Chiave, inserisci la chiave del tag. Per informazioni sulle restrizioni relative ai tag, consulta Limiti e requisiti di denominazione dei tag nella Guida alle risorse di etichettatura e all'editor di tag AWS .
6. Scegli Crea configurazione.

Per aggiungere un tag a una configurazione di firma del codice esistente

1. Apri la pagina Configurazioni di firma del codice della console Lambda.
2. Scegli il nome della configurazione di firma del codice.
3. Nelle schede sotto il riquadro Dettagli, scegli Tag.
4. Scegliere Gestisci tag.
5. Scegliere Aggiungi nuovo tag.
6. Nel campo Chiave, inserisci la chiave del tag. Per informazioni sulle restrizioni relative ai tag, consulta Limiti e requisiti di denominazione dei tag nella Guida alle risorse di etichettatura e all'editor di tag AWS .
7. Scegli Save (Salva).

Per filtrare le configurazioni di firma del codice per tag

1. Apri la pagina Configurazioni di firma del codice della console Lambda.
2. Scegli la barra di ricerca.
3. Dall'elenco a discesa, seleziona il tag sotto il sottotitolo Tag.
4. Seleziona Usa: "nome-tag" per vedere tutte le configurazioni di firma del codice etichettate con questa chiave, oppure scegli un operatore per filtrare ulteriormente in base al valore.
5. Seleziona il valore del tag da filtrare in base a una combinazione di chiave e valore del tag.

La barra di ricerca supporta anche la ricerca di chiavi di tag. Immetti il nome di una chiave per trovarla nell'elenco.

Usare i tag con AWS CLI

Puoi aggiungere e rimuovere tag sulle risorse Lambda esistenti, incluse le configurazioni di firma del codice, con l'API Lambda. Puoi aggiungere i tag anche quando crei una configurazione di firma del codice, che ti consente di mantenere etichettata una risorsa per tutto il suo ciclo di vita.

Aggiornamento dei tag con il tag Lambda APIs

Puoi aggiungere e rimuovere tag per le risorse Lambda supportate tramite le operazioni TagResourcee UntagResourceAPI.

Puoi chiamare queste operazioni tramite la AWS CLI. Per aggiungere i tag a una risorsa esistente, utilizza il comando `tag-resource`. Questo esempio aggiunge due tag, uno con la chiave *`Department`* e uno con la chiave*`CostCenter`*.

```
aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:resource-type:my-resource \
--tags Department=Marketing,CostCenter=1234ABCD
```

Pr rimuovere i tag, utilizza il comando `untag-resource`. Questo esempio rimuove il tag con la chiave*`Department`*.

```
aws lambda untag-resource --resource arn:aws:lambda:us-east-1:123456789012:resource-
type:resource-identifier \
--tag-keys Department
```

Aggiunta di tag durante la creazione di una configurazione di firma del codice

Per creare una nuova configurazione di firma del codice Lambda con tag, usa l'operazione CreateCodeSigningConfigAPI. Specifica il parametro `Tags`. È possibile richiamare questa operazione con il `create-code-signing-config` AWS CLI comando e l'`--tags`opzione. Per ulteriori informazioni sul comando CLI, vedere create-code-signing-confignel Command Reference AWS CLI .

Prima di utilizzare il parametro `Tags` con `CreateCodeSigningConfig`, assicurati che il tuo ruolo disponga dell'autorizzazione per etichettare le risorse oltre alle normali autorizzazioni necessarie per questa operazione. Per ulteriori informazioni sulle autorizzazioni per il tagging, consulta the section called “Autorizzazioni necessarie per lavorare con i tag”.

Visualizzazione dei tag con il tag Lambda APIs

Per visualizzare i tag applicati a una risorsa Lambda specifica, utilizza l'operazione API `ListTags`. Per ulteriori informazioni, consulta ListTags.

Puoi richiamare questa operazione con il `list-tags` AWS CLI comando fornendo un ARN (Amazon Resource Name).

```
aws lambda list-tags --resource arn:aws:lambda:us-east-1:123456789012:resource-type:resource-identifier
```

Filtro delle risorse per tag

Puoi utilizzare l'operazione AWS Resource Groups Tagging API GetResourcesAPI per filtrare le tue risorse in base ai tag. L'operazione `GetResources` riceve fino a 10 filtri, ognuno dei quali contenente una chiave di tag e un massimo di 10 valori di tag. Fornisci `GetResources` con un `ResourceType` per filtrare in base a tipi di risorse specifiche.

È possibile richiamare questa operazione utilizzando il `get-resources` AWS CLI comando. Per esempi di utilizzo di `get-resources`, consulta get-resources nella Riferimento ai comandi CLI di AWS .

Monitoraggio, debug e risoluzione dei problemi delle funzioni Lambda

AWS Lambda si integra con altri Servizi AWS per aiutarti a monitorare e risolvere i problemi relativi alle tue funzioni Lambda. Lambda monitora automaticamente le funzioni Lambda per tuo conto e segnala i parametri tramite Amazon CloudWatch. Per monitorare il codice durante la sua esecuzione, Lambda tiene traccia automaticamente del numero di richieste, della durata della chiamata di ogni richiesta e del numero di richieste che restituiscono un errore.

Puoi utilizzare altri Servizi AWS per risolvere i problemi relativi alle funzioni Lambda. In questa sezione viene descritto come utilizzare questi Servizi AWS per monitorare, tracciare, eseguire il debug e risolvere i problemi relativi alle funzioni e alle applicazioni Lambda. Per dettagli sulla registrazione delle funzioni e sugli errori in ogni runtime, consulta le singole sezioni del runtime.

Sezioni

- Prezzi
- Utilizzo dei parametri di CloudWatch Logs con Lambda
- Utilizzo dei registri delle funzioni Lambda
- Registrazione delle chiamate API di Esploratore di risorse AWS tramite AWS CloudTrail
- Visualizza le chiamate alla funzione Lambda utilizzando AWS X-Ray
- Monitorare le prestazioni delle funzioni con Lambda Insights di Amazon CloudWatch
- Monitoraggio delle applicazioni Lambda
- Monitoraggio delle prestazioni delle applicazioni con Amazon CloudWatch Application Signals
- Esecuzione da remoto del debug delle funzioni Lambda con Visual Studio Code

Prezzi

CloudWatch dispone di un piano gratuito senza termini di tempo. Oltre la soglia del piano gratuito, CloudWatch prevede dei costi per parametri, pannelli, allarmi, registri e informazioni dettagliate. Per ulteriori informazioni, consulta Prezzi di Amazon CloudWatch.

Utilizzo dei parametri di CloudWatch Logs con Lambda

Quando la funzione AWS Lambda termina l'elaborazione di un evento, Lambda invia ad Amazon CloudWatch i parametri relativi all'invocazione. Non è necessario concedere autorizzazioni aggiuntive al ruolo di esecuzione per ricevere i parametri delle funzioni e non sono previsti costi aggiuntivi per questi parametri.

Esistono molti tipi di parametri associati alle funzioni Lambda. Tra questi vi sono i parametri di invocazione, parametri delle prestazioni, parametri di simultaneità, parametri di invocazione asincrona e parametri dello strumento di mappatura dell'origine degli eventi. Per ulteriori informazioni, consulta the section called “Tipi di parametri”.

Sulla console CloudWatch, è possibile visualizzare questi parametri e creare grafici e pannelli di controllo con essi. È possibile impostare allarmi per rispondere alle modifiche apportate alle percentuali di utilizzo, prestazioni o errore. Lambda invia i dati dei parametri a CloudWatch in intervalli di 1 minuto. Se desideri una visione più immediata della funzione Lambda, è possibile creare parametri personalizzati ad alta risoluzione. Per i parametri personalizzati e gli allarmi CloudWatch sono previsti dei costi. Per ulteriori informazioni, consulta la pagina Prezzi di Amazon CloudWatch.

Visualizzazione di parametri per le funzioni Lambda

Usa la console CloudWatch per visualizzare i parametri per le tue funzioni Lambda. Nella console, puoi filtrare e ordinare i parametri della funzione in base al nome, all'alias o alla versione o all'UUID del mapping dell'origine di eventi.

Per visualizzare i parametri nella console CloudWatch

1. Aprire la pagina Metrics (Parametri) (namespace di `AWS/Lambda`) nella console CloudWatch.
2. Nella scheda Sfoglia, in Parametri, scegli una delle seguenti dimensioni:

   - Per nome funzione (`FunctionName`) – Visualizza i parametri aggregati per tutte le versioni e gli alias di una funzione.
   - Per risorsa (`Resource`) – Visualizza i parametri per una versione o un alias di una funzione.
   - Per version eseguita (`ExecutedVersion`) – Visualizza i parametri per una combinazione di alias e versione. Utilizzare la dimensione `ExecutedVersion` per confrontare le percentuali di errore per due versioni di una funzione che sono entrambe destinazioni di un alias ponderato.
   - Per UUID di mapping di origine di eventi (`EventSourceMappingUUID`): visualizza i parametri per uno strumento di mappatura dell'origine degli eventi.

- In tutte le funzioni (nessuna): visualizza i parametri aggregati per tutte le funzioni nella Regione AWS corrente.

3. Scegli un parametro. Il parametro dovrebbe apparire automaticamente nel grafico visivo e nella scheda Parametri nel grafico.

Per impostazione predefinita, i grafici utilizzano la statistica `Sum` per tutti i parametri. Per scegliere un parametro diverso e personalizzare il grafico, utilizzare le opzioni nella scheda Graphed metrics (Parametri grafico).

> **Note**
>
> Il timestamp su un parametro riflette quando la funzione è stata invocata. A seconda della durata della chiamata, possono essere necessari diversi minuti prima dell'emissione del parametro. Se, ad esempio, la funzione ha un timeout di 10 minuti, per trovare i parametri accurati bisogna cercare a ritroso oltre i 10 minuti.

Per ulteriori informazioni su CloudWatch, consulta la Guida per l'utente di Amazon CloudWatch.

Tipi di parametri per le funzioni Lambda

Nella sezione seguente sono descritti i tipi di parametri Lambda disponibili nella console CloudWatch.

Argomenti

- Parametri di invocazione
- Metodo di implementazione
- Parametri prestazionali
- Parametri di concorrenza
- Parametri di chiamata asincrona
- Parametri dello strumento di mappatura dell'origine degli eventi

Parametri di invocazione

I parametri di invocazione sono indicatori binari del risultato di una chiamata alla funzione Lambda. Visualizza questi parametri con la statistica `Sum`. Ad esempio, se la funzione restituisce un errore, Lambda invia il parametro `Errors` con un valore pari a 1. Per ottenere un conteggio del numero di

errori di funzione che si sono verificati ogni minuto, visualizzare la somma `Sum` del parametro `Errors` con un periodo di un minuto.

- `Invocations`: il numero di volte in cui viene chiamato il codice di funzione, incluse le chiamate riuscite e le chiamate che determinano un errore di funzione. Le chiamate non vengono registrate se la richiesta di chiamata è limitata o altrimenti viene generato un errore di chiamata. Il valore di `Invocations` equivale al numero di richieste fatturate.
- `Errors`: il numero di chiamate che provocano un errore di funzione. Gli errori di funzione includono eccezioni generate dal codice e eccezioni generate dal runtime Lambda. Il runtime restituisce errori per problemi quali timeout ed errori di configurazione. Per calcolare la percentuale di errore, dividere il valore di `Errors` per il valore di `Invocations`. Tieni presente che il timestamp di un parametro di errore riflette quando è stata richiamata la funzione, non quando si è verificato l'errore.
- `DeadLetterErrors`: per la chiamata asincrona, il numero di tentativi di invio non riusciti da parte di Lambda di un evento a una coda DLQ. Gli errori DLQ possono verificarsi a causa di risorse configurate erroneamente o limiti di dimensione.
- `DestinationDeliveryFailures`: per la chiamata asincrona e per lo strumento di mappatura dell'origine degli eventi supportato, indica il numero di tentativi di invio non riusciti da parte di Lambda di un evento a una destinazione. Per gli strumenti di mappatura dell'origine degli eventi, Lambda supporta destinazioni per le origini di flusso (DynamoDB e Kinesis). Gli errori di recapito possono verificarsi a causa di errori di autorizzazioni, risorse configurate erroneamente o limiti di dimensione. Gli errori possono verificarsi anche se la destinazione che hai configurato è di tipo non supportato, ad esempio una coda FIFO di Amazon SQS o un argomento FIFO di Amazon SNS.
- `Throttles`: il numero di richieste di chiamata con throttling. Quando tutte le istanze di funzione elaborano le richieste e non è disponibile alcuna simultaneità per l'aumento, Lambda rifiuta le richieste aggiuntive con un errore `TooManyRequestsException`. Le richieste con limitazione e altri errori di chiamata non contano come `Invocations` o `Errors`.
- `OversizedRecordCount`: per le origini di eventi di Amazon DocumentDB, il numero di eventi che la funzione riceve dal flusso di modifiche è superiore a 6 MB. Lambda elimina il messaggio ed emette questo parametro.
- `ProvisionedConcurrencyInvocations`: il numero di volte in cui il codice di funzione viene richiamato tramite la simultaneità fornita.
- `ProvisionedConcurrencySpilloverInvocations`: il numero di volte in cui il codice di funzione viene chiamato tramite la simultaneità standard quando è in uso tutta la simultaneità fornita.

- `RecursiveInvocationsDropped`: il numero di volte in cui Lambda ha interrotto l'invocazione della funzione perché ha rilevato che la funzione fa parte di un ciclo ricorsivo infinito. Il rilevamento di un ciclo ricorsivo monitora quante volte una funzione viene richiamata come parte di una catena di richieste tracciando i metadati aggiunti dagli SDK AWS supportati. Per impostazione predefinita, se la funzione viene richiamata come parte di una catena di richieste circa 16 volte, Lambda interrompe l'invocazione successiva. Se disabiliti il rilevamento del ciclo ricorsivo, questo parametro non viene emesso. Per ulteriori informazioni sull'utilizzo di questa funzionalità, consulta Usa il rilevamento di un ciclo ricorsivo Lambda per prevenire loop infiniti.

Metodo di implementazione

Le metriche di distribuzione forniscono informazioni sugli eventi di distribuzione della funzione Lambda e sui relativi processi di convalida.

- `SignatureValidationErrors`— Il numero di volte in cui si è verificata la distribuzione di un pacchetto di codice con errori di convalida della firma quando la politica di configurazione della firma del codice è impostata su. `Warn` Questa metrica viene emessa quando i controlli di scadenza, mancata corrispondenza o revoca hanno esito negativo, ma la distribuzione è comunque consentita a causa dell'impostazione dei criteri. `Warn` Per ulteriori informazioni sulla firma, consulta Utilizzo della firma del codice per verificare l'integrità del codice con Lambda.

Parametri prestazionali

I parametri delle prestazioni forniscono dettagli delle prestazioni relativi a una singola chiamata della funzione. Ad esempio, il parametro `Duration` indica il tempo in millisecondi che la funzione impiega per l'elaborazione di un evento. Per avere un'idea della velocità con cui la funzione elabora gli eventi, visualizzare questi parametri con la statistica `Average` o `Max`.

- `Duration` – La quantità di tempo che il codice della funzione impiega durante l'elaborazione di un evento. La durata fatturata per una invocazione è il valore di `Duration` arrotondato per eccesso al millisecondo più vicino. `Duration` non include il tempo di avvio a freddo.
- `PostRuntimeExtensionsDuration` – La quantità cumulativa di tempo che il runtime trascorre eseguendo il codice per le estensioni dopo il completamento del codice funzione.
- `IteratorAge`: per le origini eventi DynamoDB, Kinesis e Amazon DocumentDB, l'età (in millisecondi) dell'ultimo record dell'evento. Questo parametro misura il tempo che passa tra il momento in cui il flusso riceve il record e il momento in cui lo strumento di mappatura dell'origine degli eventi invia l'evento alla funzione.

- `OffsetLag`: per le origini eventi Apache Kafka autogestite e Streaming gestito da Amazon per Apache Kafka (Amazon MSK), la differenza di offset tra l'ultimo record scritto su un argomento e l'ultimo record elaborato dal gruppo di consumer della funzione. Sebbene un argomento di Kafka possa avere più partizioni, questo parametro misura il ritardo di offset a livello di argomento.

`Duration` supporta anche le statistiche percentili (p). Utilizzare i percentili per escludere valori estremi che incideranno sulle statistiche `Average` e `Maximum`. Ad esempio, la statistica `p95` mostra la durata massima del 95% delle chiamate, escludendo il 5% più lento. Per ulteriori informazioni, consulta Percentili nella Guida per l'utente di Amazon CloudWatch.

Parametri di concorrenza

Lambda segnala i parametri di simultaneità come conteggio aggregato del numero di istanze che elaborano eventi in una funzione, una versione, un alias o una Regione AWS. Per vedere quanto sei vicino al superamento dei limiti di simultaneità, visualizza questi parametri con la statistica `Max`.

- `ConcurrentExecutions` – Il numero di istanze di funzione che stanno elaborando gli eventi. Se questo numero raggiunge la quota di esecuzioni simultanee per la regione o il limite di simultaneità riservato configurato per la funzione, Lambda limita le richieste di chiamata aggiuntive.
- `ProvisionedConcurrentExecutions`: il numero di istanze di funzione che stanno elaborando eventi tramite la simultaneità fornita. Per ogni chiamata di un alias o versione con la simultaneità fornita, Lambda emette il conteggio corrente. Se la tua funzione è inattiva o non riceve richieste, Lambda non emette questa metrica.
- `ProvisionedConcurrencyUtilization`: per una versione o un alias, il valore di `ProvisionedConcurrentExecutions` diviso per la quantità totale di simultaneità fornita configurata. Ad esempio, se configuri una simultaneità fornita pari a 10 per la funzione e `ProvisionedConcurrentExecutions` è 7, allora `ProvisionedConcurrencyUtilization` è 0,7.

  Se la tua funzione è inattiva o non riceve richieste, Lambda non emette questa metrica perché è basata su. `ProvisionedConcurrentExecutions` Tienilo a mente se lo utilizzi `ProvisionedConcurrencyUtilization` come base per gli allarmi CloudWatch.
- `UnreservedConcurrentExecutions`: per una regione, il numero di eventi che vengono elaborati da funzioni che non dispongono di simultaneità riservata.
- `ClaimedAccountConcurrency`: per una Regione, la quantità di simultaneità non disponibile per le invocazioni on demand. `ClaimedAccountConcurrency` corrisponde a `UnreservedConcurrentExecutions` più la quantità di simultaneità allocata (ovvero la

simultaneità totale riservata più la simultaneità totale fornita). Per ulteriori informazioni, consulta Lavorare con il parametro `ClaimedAccountConcurrency`.

Parametri di chiamata asincrona

I parametri di chiamata asincrona forniscono dettagli sulle chiamate asincrone da origini di eventi e sulle chiamate dirette. Puoi impostare le soglie e gli allarmi per la notifica di alcuni cambiamenti. Ad esempio, quando si verifica un aumento indesiderato del numero di eventi in coda per l'elaborazione (`AsyncEventsReceived`). Oppure, quando un evento aspetta da molto tempo di essere elaborato (`AsyncEventAge`).

- `AsyncEventsReceived`: il numero di eventi che Lambda mette correttamente in coda per l'elaborazione. Questo parametro fornisce informazioni sul numero di eventi ricevuti da una funzione Lambda. Monitora questo parametro e imposta gli allarmi relativi alle soglie per verificare eventuali problemi. Ad esempio, per rilevare un numero indesiderato di eventi inviati a Lambda e diagnosticare rapidamente i problemi derivanti da configurazioni errate di trigger o funzioni. Le discrepanze tra `AsyncEventsReceived` e `Invocations` possono indicare una disparità nell'elaborazione, l'eliminazione degli eventi o un potenziale arretrato della coda.
- `AsyncEventAge`: il tempo che intercorre tra il momento in cui Lambda mette in coda correttamente l'evento e il momento in cui la funzione viene richiamata. Il valore di questo parametro aumenta quando gli eventi vengono ritentati a causa di errori di chiamata o limitazioni. Monitora questo parametro e imposta allarmi per rilevare le soglie su diverse statistiche relative a quando si verifica un accumulo di code. Per risolvere un aumento di questo parametro, consulta il parametro `Errors` per identificare gli errori della funzione e il parametro `Throttles` per identificare i problemi di simultaneità.
- `AsyncEventsDropped`: il numero di eventi eliminati senza eseguire correttamente la funzione. Se configuri una coda DLQ o una destinazione `OnFailure`, gli eventi vengono inviati lì prima di essere eliminati. Gli eventi vengono eliminati per diversi motivi. Ad esempio, possono superare la durata massima o esaurire il numero massimo di tentativi oppure la simultaneità riservata potrebbe essere impostata su 0. Per risolvere il problema relativo all'eliminazione degli eventi, consulta il parametro `Errors` per identificare gli errori della funzione e il parametro `Throttles` per identificare i problemi di simultaneità.

Parametri dello strumento di mappatura dell'origine degli eventi

I parametri dello strumento di mappatura dell'origine degli eventi forniscono informazioni sul comportamento di elaborazione dello strumento di mappatura dell'origine degli eventi. Questi parametri consentono di monitorare il flusso e lo stato degli eventi, compresi gli eventi che lo strumento di mappatura dell'origine degli eventi ha elaborato, filtrato o eliminato con successo.

Devi attivare la ricezione di parametri relative ai conteggi (`PolledEventCount`, `FilteredOutEventCount`, `InvokedEventCount`, `FailedInvokeEventCount`, `DroppedEventCount`, `OnFailureDestinationDeliveredEventCount` e `DeletedEventCount`). Per attivare l'accesso, puoi utilizzare la console o l'API Lambda.

Per abilitare i parametri o uno strumento di mappatura dell'origine degli eventi (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la funzione per la quale desideri abilitare i parametri.
3. Scegli la scheda Configurazione, quindi scegli Trigger.
4. Scegli lo strumento di mappatura dell'origine degli eventi per cui desideri abilitare i parametri, quindi scegli Modifica.
5. In Configurazione dello strumento di mappatura dell'origine degli eventi, scegli Abilita parametri.
6. Selezionare Salva.

In alternativa, puoi abilitare i parametri per lo strumento di mappatura dell'origine degli eventi a livello di codice utilizzando l'oggetto EventSourceMappingMetricsConfig in EventSourceMappingConfiguration. Ad esempio, il seguente comando UpdateEventSourceMapping della CLI abilita i parametri per uno strumento di mappatura dell'origine degli eventi:

```
aws lambda update-event-source-mapping \
    --uuid a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
    --metrics-config Metrics=EventCount
```

Visualizza i parametri relativi al conteggio degli eventi con la statistica `Sum`.

> **Warning**
>
> Gli strumenti di mappatura dell'origine degli eventi elaborano ogni evento almeno una volta e può verificarsi un'elaborazione duplicata dei record. Per questo motivo, gli eventi possono essere contati più volte nei parametri che coinvolgono il conteggio degli eventi.

- `PolledEventCount`: il numero di eventi che Lambda legge correttamente dall'origine eventi. Se Lambda esegue un polling di eventi ma riceve un polling vuoto (nessun nuovo record), Lambda emette un valore 0 per questo parametro. Usa questo parametro per rilevare se lo strumento di mappatura dell'origine degli eventi esegue correttamente il polling dei nuovi eventi.
- `FilteredOutEventCount`: per lo strumento di mappatura dell'origine degli eventi con un criterio di filtro, il numero di eventi filtrati in base a tali criteri di filtro. Utilizza questo parametro per rilevare se lo strumento di mappatura dell'origine degli eventi filtra correttamente gli eventi. Per gli eventi che soddisfano i criteri di filtro, Lambda emette un parametro 0.
- `InvokedEventCount`: il numero di eventi che hanno richiamato la funzione Lambda. Usa questo parametro per verificare che gli eventi stiano richiamando correttamente la tua funzione. Se un evento causa un errore nella funzione o una limitazione, `InvokedEventCount` può contare più volte per lo stesso evento sottoposto a polling a causa dei vari tentativi automatici.
- `FailedInvokeEventCount`: il numero di eventi con cui Lambda ha provato a richiamare la funzione ma senza riuscirci. Le invocazioni possono avere esito negativo per motivi quali problemi di configurazione della rete, autorizzazioni non corrette o una funzione, una versione o un alias Lambda eliminati. Se lo strumento di mappatura dell'origine degli eventi ha abilitato le risposte in batch parziali, `FailedInvokeEventCount` include qualsiasi evento con un valore non vuoto `BatchItemFailures` nella risposta.

  > **Note**
  >
  > Il timestamp per il parametro `FailedInvokeEventCount` rappresenta la fine dell'invocazione della funzione. Questo comportamento è diverso dagli altri parametri di errore di invocazione Lambda, che hanno un timestamp all'inizio dell'invocazione della funzione.

- `DroppedEventCount`: il numero di eventi che Lambda ha interrotto a causa della scadenza o dell'esaurimento dei nuovi tentativi. In particolare, si tratta del numero di record che superano i valori configurati per `MaximumRecordAgeInSeconds` o `MaximumRetryAttempts`. È importante

sottolineare che questo non include il numero di record che scadono a causa del superamento delle impostazioni di conservazione dell'origine eventi. Gli eventi eliminati escludono anche gli eventi inviati a una destinazione in errore. Utilizza questo parametro per rilevare un aumento del backlog di eventi.

- `OnFailureDestinationDeliveredEventCount`: per lo strumento di mappatura dell'origine degli eventi con una destinazione in errore configurata, il numero di eventi inviati a tale destinazione. Utilizza questo parametro per monitorare gli errori di funzione relativi alle chiamate da questa origine eventi. Se la consegna alla destinazione non riesce, Lambda gestisce i parametri come segue:
  - Lambda non emette il parametro `OnFailureDestinationDeliveredEventCount`.
  - Per il parametro `DestinationDeliveryFailures`, Lambda emette un 1.
  - Per il parametro `DroppedEventCount`, Lambda emette un numero pari al numero di eventi che hanno avuto esito negativo nella consegna.
- `DeletedEventCount`: il numero di eventi che Lambda elimina correttamente in seguito all'elaborazione. Se Lambda prova a eliminare un evento ma fallisce, Lambda emette un parametro 0. Utilizza questo parametro per assicurarti che gli eventi elaborati correttamente vengano eliminati dall'origine eventi.

Se lo strumento di mappatura dell'origine degli eventi è disabilitato, non riceverai i parametri dello strumento. Potresti anche vedere parametri mancanti se CloudWatch o Lambda presentano una disponibilità ridotta.

Non tutti i parametri dello strumento di mappatura dell'origine degli eventi sono disponibili per ogni origine eventi. Al momento, i parametri dello strumento di mappatura dell'origine degli eventi sono disponibili per le origini eventi dei flussi Amazon SQS, Kinesis e DynamoDB. La seguente matrice di disponibilità riassume i parametri supportati per ogni tipo di origine eventi.

Parametro dello strumento di mappatura dell'origine degli eventi	Supporto per Amazon SQS	Supporto per flussi Kinesis e DynamoDB
`PolledEventCount`	Sì	Sì
`FilteredOutEventCount`	Sì	Sì

Parametro dello strumento di mappatura dell'origine degli eventi	Supporto per Amazon SQS	Supporto per flussi Kinesis e DynamoDB
`InvokedEventCount`	Sì	Sì
`FailedInvokeEventCount`	Sì	Sì
`DroppedEventCount`	No	Sì
`OnFailureDestinationDeliveredEventCount`	No	Sì
`DeletedEventCount`	Sì	No

Inoltre, se lo strumento di mappatura dell'origine degli eventi delle origini eventi è in modalità provisioning, Lambda fornisce il seguente parametro:

- `ProvisionedPollers`: per lo strumento di mappatura dell'origine degli eventi in modalità provisioning, il numero di poller di eventi che sono attivamente in esecuzione. Visualizza questo parametro utilizzando il parametro `MAX`.

Utilizzo dei registri delle funzioni Lambda

Per aiutarti a risolvere i problemi, AWS Lambda monitora automaticamente le funzioni Lambda per tuo conto. Puoi visualizzare i log delle funzioni Lambda utilizzando la console Lambda, la console, CloudWatch la AWS CLI(), AWS Command Line Interface l'API. CloudWatch Puoi anche configurare Lambda per inviare log ad Amazon S3 e Firehose.

Se il ruolo di esecuzione della funzione dispone delle autorizzazioni necessarie, Lambda acquisisce i log per tutte le richieste gestite dalla funzione e li invia ad CloudWatch Amazon Logs, che è la destinazione predefinita. Puoi anche usare la console Lambda per configurare Amazon S3 o Firehose come destinazioni di registrazione.

- CloudWatch Logs è la destinazione di registrazione predefinita per le funzioni Lambda. CloudWatch Logs offre funzionalità di visualizzazione e analisi dei log in tempo reale, con supporto per la creazione di metriche e allarmi basati sui dati di registro.
- Amazon S3 è conveniente per lo storage a lungo termine e servizi come Athena possono essere utilizzati per analizzare i log. La latenza è in genere superiore.
- Firehose offre lo streaming gestito dei log verso varie destinazioni. Se è necessario inviare log ad altri AWS servizi (ad esempio, OpenSearch Service o Redshift Data API) o piattaforme di terze parti (come Datadog, New Relic o Splunk), Firehose semplifica tale processo fornendo integrazioni predefinite. Puoi anche eseguire lo streaming su endpoint HTTP personalizzati senza configurare un'infrastruttura aggiuntiva.

Scelta della destinazione del servizio a cui inviare i log

Nella scelta di un servizio come destinazione per i registri delle funzioni, tenete conto dei seguenti fattori chiave:

- La gestione dei costi varia in base al servizio. Amazon S3 offre in genere l'opzione più economica per lo storage a lungo termine, mentre CloudWatch Logs consente di visualizzare log, elaborare log e configurare avvisi in tempo reale. I costi di Firehose includono sia il servizio di streaming che i costi associati alla destinazione a cui lo si configura per lo streaming.
- Le funzionalità di analisi differiscono tra i servizi. CloudWatch Logs eccelle nel monitoraggio in tempo reale e si integra nativamente con altre CloudWatch funzionalità, come Logs Insights e Live Tail. Amazon S3 funziona bene con strumenti di analisi come Athena e può integrarsi con vari servizi, sebbene possa richiedere una configurazione aggiuntiva. Firehose semplifica lo streaming diretto verso AWS servizi specifici (come Service OpenSearch e Redshift Data API) e piattaforme

di terze parti supportate (come Datadog e Splunk) fornendo integrazioni predefinite, riducendo potenzialmente il lavoro di configurazione.

- La configurazione e la facilità d'uso variano in base al servizio. CloudWatch Logs è la destinazione di log predefinita: funziona immediatamente senza configurazioni aggiuntive e fornisce una visualizzazione e un'analisi dei log semplici tramite la console. CloudWatch Se hai bisogno di inviare i log ad Amazon S3, dovrai eseguire una configurazione iniziale nella console Lambda e configurare le autorizzazioni del bucket. Se avete bisogno di log inviati direttamente a servizi come OpenSearch Service o piattaforme di analisi di terze parti, Firehose può semplificare tale processo.

Configurazione delle destinazioni dei log

AWS Lambda supporta più destinazioni per i registri delle funzioni. Questa guida spiega le destinazioni di registrazione disponibili e ti aiuta a scegliere l'opzione giusta per le tue esigenze. Indipendentemente dalla destinazione scelta, Lambda offre opzioni per controllare il formato, il filtraggio e la consegna dei log.

Lambda supporta sia i formati JSON che quelli di testo semplice per i log delle funzioni. I log strutturati JSON offrono una migliore ricercabilità e consentono l'analisi automatizzata, mentre i log di testo semplice offrono semplicità e costi di archiviazione potenzialmente ridotti. Puoi controllare quali log Lambda invia alla destinazione prescelta configurando i livelli di registro per i log di sistema e delle applicazioni. Il filtraggio consente di gestire i costi di archiviazione e semplifica la ricerca delle voci di registro pertinenti durante il debug.

Per istruzioni di configurazione dettagliate per ciascuna destinazione, consulta le seguenti sezioni:

- Lambda invia automaticamente i log delle funzioni a CloudWatch Logs.
- Invio dei log delle funzioni Lambda a Firehose
- Invio di log delle funzioni Lambda ad Amazon S3

Configurazione dei controlli di registrazione avanzati per le funzioni Lambda

Per darti un maggiore controllo sul modo in cui i log delle funzioni vengono acquisiti, elaborati e consumati, Lambda offre le seguenti opzioni di configurazione della registrazione:

- Formato di registro: scegli tra testo semplice e formato JSON strutturato per i log della tua funzione.

- Livello di registro: per i log strutturati JSON, scegli il livello di dettaglio dei log a cui Lambda invia, CloudWatch ad esempio`FATAL`,`ERROR`,`WARN`, `INFO` e. `DEBUG` `TRACE`
- Gruppo di log: scegli il gruppo di CloudWatch log a cui la funzione invia i log.

Per ulteriori informazioni sulla configurazione dei controlli di registrazione avanzati, consulta le seguenti sezioni:

- Configurazione dei formati di log JSON e testo normale
- Filtraggio a livello di log
- Configurazione dei gruppi di CloudWatch log

Configurazione dei formati di log JSON e testo normale

L'acquisizione degli output log come coppie chiave-valore JSON semplifica la ricerca e il filtraggio durante il debug delle funzioni. Con i log in formato JSON, puoi aggiungere ai log anche tag e informazioni contestuali. Questo può aiutarti a eseguire analisi automatizzate di grandi volumi di dati di log. A meno che il flusso di lavoro di sviluppo non si basi su strumenti esistenti che utilizzano i log Lambda in testo normale, ti consigliamo di selezionare JSON come formato di log.

> **Istanze gestite Lambda**
>
> Le istanze gestite Lambda supportano solo il formato di registro JSON. Quando crei una funzione Managed Instances, Lambda configura automaticamente il formato di registro in JSON e non è possibile modificarlo in testo semplice. Per ulteriori informazioni sulle istanze gestite, consulta. Istanze gestite Lambda

Per tutti i runtime gestiti da Lambda, puoi scegliere se inviare i log di sistema della funzione a CloudWatch Logs in formato testo semplice non strutturato o JSON. I log di sistema sono i log generati da Lambda e a volte vengono chiamati log eventi della piattaforma.

Per i runtime supportati, quando si utilizza uno dei metodi di registrazione integrati supportati, Lambda può anche generare i log delle applicazioni della funzione (i log generati dal codice della funzione) in formato JSON strutturato. Quando configuri il formato di log della funzione per questi runtime, la configurazione scelta si applica sia ai log di sistema sia a quelli delle applicazioni.

Per i runtime supportati, se la funzione utilizza una libreria o un metodo di registrazione supportato, non è necessario apportare modifiche al codice esistente per Lambda per acquisire i log in formato JSON strutturato.

> **Note**
>
> L'utilizzo della formattazione dei log JSON aggiunge metadati aggiuntivi e codifica i messaggi di log come oggetti JSON contenenti una serie di coppie chiave-valore. Per questo motivo, la dimensione dei messaggi di log della funzione può aumentare.

Runtime e metodi di registrazione supportati

Lambda attualmente supporta l'opzione di generare i log delle applicazioni in formato JSON strutturato per i seguenti runtime.

Lingua	Versioni supportate
Java	Tutti i runtime Java eccetto Java 8 su Amazon Linux 1
.NET	.NET 8
Node.js	Node.js 16 e versioni successive
Python	Python 3.8 e versioni successive
Rust	N/A

Affinché Lambda invii i log delle applicazioni della funzione CloudWatch in formato JSON strutturato, la funzione deve utilizzare i seguenti strumenti di registrazione integrati per generare i log:

- Java: il logger o Log4j2. `LambdaLogger` Per ulteriori informazioni, consulta the section called “Registrazione dei log”.
- .NET: l'`ILambdaLogger`istanza sull'oggetto contestuale. Per ulteriori informazioni, consulta Registrare e monitorare le funzioni Lambda con C#.

- Node.js: i metodi della console `console.trace` `console.debugconsole.log`,`console.info`,`console.error`, `econsole.warn`. Per ulteriori informazioni, consulta the section called “Registrazione dei log”.
- Python: la libreria Python standard. `logging` Per ulteriori informazioni, consulta the section called “Registrazione dei log”.
- Rust: La cassa. `tracing` Per ulteriori informazioni, consulta the section called “Registrazione dei log”.

Per altri runtime Lambda gestiti, Lambda attualmente supporta nativamente solo l'acquisizione dei log di sistema in formato JSON strutturato. Tuttavia, è ancora possibile acquisire i log delle applicazioni in formato JSON strutturato in qualsiasi runtime utilizzando strumenti di registrazione come Powertools per AWS Lambda generare output di log in formato JSON.

Formati di log predefiniti

Attualmente, il formato di log predefinito per tutti i runtime Lambda è il testo normale. Per le istanze gestite Lambda, il formato di registro è sempre JSON e non può essere modificato.

Se utilizzi già librerie di registrazione come Powertools AWS Lambda per generare i log delle funzioni in formato strutturato JSON, non devi modificare il codice se selezioni la formattazione dei log JSON. Lambda non codifica due volte i log che sono già codificati in JSON, quindi i log delle applicazioni della funzione continueranno a essere acquisiti come prima.

Formato JSON per i log di sistema

Quando configuri il formato di log della funzione come JSON, ogni elemento del log di sistema (evento della piattaforma) viene acquisito come un oggetto JSON contenente coppie chiave-valore con le seguenti chiavi:

- `"time"`: l'ora in cui è stato generato il messaggio di log
- `"type"`: il tipo di evento che viene registrato
- `"record"`: il contenuto dell'output log

Il formato del valore `"record"` varia in base al tipo di evento registrato. Per ulteriori informazioni, consulta the section called “Tipi di oggetti `Event` dell'API di telemetria”. Per ulteriori informazioni sui livelli di log assegnati ai log eventi di sistema, consulta la pagina the section called “Strumento di mappatura degli eventi a livello di log di sistema”.

A titolo di confronto, i due esempi seguenti mostrano lo stesso output log in formato di testo normale e in formato JSON strutturato. Tieni presente che, nella maggior parte dei casi, i log eventi di sistema contengono più informazioni quando vengono emessi in formato JSON rispetto a quando vengono emessi in testo normale.

Example testo normale:

```
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 strutturato:

```
{
  "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**
>
> L'the section called “API di telemetria” emette sempre eventi della piattaforma come `START` e `REPORT` in formato JSON. La configurazione del formato dei log di sistema a cui Lambda invia CloudWatch non influisce sul comportamento dell'API Lambda Telemetry.

Formato JSON per i log delle applicazioni

Quando configuri il formato di log della funzione come JSON, gli output di log delle applicazioni scritti utilizzando le librerie e i metodi di registrazione di log supportati vengono acquisiti come oggetti JSON che contengono coppie chiave-valore con le chiavi riportate di seguito.

- `"timestamp"`: l'ora in cui è stato generato il messaggio di log

- `"level"`: il livello di log assegnato al messaggio
- `"message"`: il contenuto del messaggio di log
- `"requestId"` (Python, .NET e Node.js) o `"AWSrequestId"` (Java): l'ID di richiesta univoco per l'invocazione della funzione

A seconda del runtime e del metodo di registrazione di log utilizzato dalla funzione, questo oggetto JSON può anche contenere coppie di chiavi aggiuntive. Ad esempio, in Node.js, se la funzione utilizza metodi della `console` per registrare i log degli oggetti di errore con più argomenti, l'oggetto JSON conterrà coppie chiave-valore aggiuntive con le chiavi `errorMessage`, `errorType` e `stackTrace`. Per ulteriori informazioni sui log in formato JSON in diversi runtime Lambda, consulta the section called “Registrazione dei log”, the section called “Registrazione dei log” e the section called “Registrazione dei log”.

> **Note**
>
> La chiave utilizzata da Lambda per il valore del timestamp è diversa per i log di sistema e i log delle applicazioni. Per i log di sistema, Lambda utilizza la chiave `"time"` per mantenere la coerenza con l'API di telemetria. Per i log delle applicazioni, Lambda segue le convenzioni dei runtime supportati e utilizza `"timestamp"`.

A titolo di confronto, i due esempi seguenti mostrano lo stesso output log in formato di testo normale e in formato JSON strutturato.

Example testo normale:

```
2024-10-27T19:17:45.586Z 79b4f56e-95b1-4643-9700-2807f4e68189 INFO some log message
```

Example JSON strutturato:

```
{
    "timestamp":"2024-10-27T19:17:45.586Z",
    "level":"INFO",
    "message":"some log message",
    "requestId":"79b4f56e-95b1-4643-9700-2807f4e68189"
}
```

Impostazione del formato di log della funzione

Per configurare il formato di registro per la tua funzione, puoi usare la console Lambda o il AWS Command Line Interface ()AWS CLI. Puoi anche configurare il formato di registro di una funzione utilizzando i comandi CreateFunctione l'API UpdateFunctionConfigurationLambda, la risorsa AWS Serverless Application Model (AWS SAM) e la AWS::Serverless::Function CloudFormationAWS::Lambda::Functionrisorsa.

La modifica del formato di registro della funzione non influisce sui log esistenti archiviati in CloudWatch Logs. Solo i nuovi log utilizzeranno il formato aggiornato.

Se modifichi il formato di log della funzione in JSON e non imposti il livello di log, Lambda imposta in automatico il livello di log dell'applicazione e il livello di log del sistema della funzione su INFO. Ciò significa che Lambda invia solo output di log di livello INFO e inferiore a Logs. CloudWatch Per ulteriori informazioni sul filtraggio a livello di log di applicazioni e sistemi, consulta the section called “Filtraggio a livello di log”

> **Note**
>
> Per i runtime Python, quando il formato di log della funzione è impostato su testo semplice, l'impostazione predefinita a livello di log è WARN. Ciò significa che Lambda invia solo output di log di livello WARN e inferiore a Logs. CloudWatch La modifica del formato di log della funzione in JSON modifica questo comportamento predefinito. Per ulteriori informazioni sulla registrazione di log in Python, consulta the section called “Registrazione dei log”.

Per le funzioni Node.js che emettono log in formato EMF (Embedded Metric Format), la modifica del formato di registro della funzione in JSON potrebbe comportare l'impossibilità di riconoscere le metriche. CloudWatch

> **Important**
>
> Se la vostra funzione utilizza Powertools for AWS Lambda (TypeScript) o le librerie client EMF open source per emettere i log EMF, aggiornate le librerie Powertools ed EMF alle versioni più recenti per assicurarvi che possa continuare ad analizzare i log correttamente. CloudWatch Se passi al formato di log JSON, ti consigliamo anche di eseguire dei test per garantire la compatibilità con i parametri incorporati della tua funzione. Per ulteriori consigli

sulle funzioni node.js che emettono log EMF, consulta la pagina the section called “Utilizzo di librerie client formato del parametro incorporato (EMF) con log JSON strutturati”.

Configurazione del formato di log di una funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione
3. Nella pagina di configurazione della funzione, scegli Strumenti di monitoraggio e gestione.
4. Nel riquadro Configurazione della registrazione, scegli Modifica.
5. In Contenuto del log, per Formato del log seleziona Testo o JSON.
6. Scegli Save (Salva).

Modifica del formato di log di una funzione esistente (AWS CLI)

- Per modificare il formato di log di una funzione esistente, utilizza il comando update-function-configuration. Imposta l'opzione `LogFormat` in `LoggingConfig` su `JSON` o `Text`.

```
aws lambda update-function-configuration \
  --function-name myFunction \
  --logging-config LogFormat=JSON
```

Configurazione del formato di log durante la creazione di una funzione (AWS CLI)

- Per configurare il formato di log quando crei una nuova funzione, utilizza l'opzione `--logging-config` nel comando create-function. Impostare `LogFormat` su `JSON` o su `Text`. Il comando di esempio seguente crea una funzione Node.js che genera i log in formato JSON strutturato.

  Se non specifichi un formato di log quando crei una funzione, Lambda utilizzerà il formato di log predefinito per la versione di runtime selezionata. Per informazioni sui formati di registrazione predefiniti, consulta la pagina the section called “Formati di log predefiniti”.

```
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
```

Filtraggio a livello di log

Lambda può filtrare i log della funzione in modo che solo i log con un certo livello di dettaglio o inferiore vengano inviati a Logs. CloudWatch Puoi configurare il filtraggio a livello di log separatamente per i log di sistema della funzione (i log generati da Lambda) e i log delle applicazioni (i log generati dal codice della funzione).

Per the section called “Runtime e metodi di registrazione supportati”, non è necessario apportare modifiche al codice della funzione per Lambda per filtrare i log delle applicazioni della funzione in Lambda.

Per tutti gli altri runtime e metodi di registrazione, il codice della funzione deve generare log eventi in `stdout` o `stderr` come oggetti in formato JSON contenenti una coppia chiave-valore con la chiave `"level"`. Ad esempio, Lambda interpreta il seguente output su `stdout` come un log di livello DEBUG.

```
print('{"level": "debug", "msg": "my debug log", "timestamp":
 "2024-11-02T16:51:31.587199Z"}')
```

Se il campo del valore `"level"` non è valido o è assente, Lambda assegnerà all'output log il livello INFO. Affinché Lambda utilizzi il campo timestamp, è necessario specificare l'ora in un formato timestamp RFC 3339 valido. Se non fornisci un timestamp valido, Lambda assegnerà al log il livello INFO e aggiungerà un timestamp per tuo conto.

Quando assegni un nome alla chiave timestamp, segui le convenzioni del runtime che stai utilizzando. Lambda supporta le convenzioni di denominazione più comuni utilizzate dai runtime gestiti.

> **Note**
>
> Per utilizzare il filtraggio a livello di log, la funzione deve essere configurata per utilizzare il formato di log JSON. Attualmente, il formato di log predefinito per tutti i runtime gestiti da Lambda è il testo normale. Per informazioni su come impostare il formato di log della funzione su JSON, consulta la pagina the section called “Impostazione del formato di log della funzione”.

Per i log delle applicazioni (i log generati dal codice della funzione), puoi scegliere tra i seguenti livelli di log.

Livello di log	Utilizzo standard
TRACE (dettaglio massimo)	Le informazioni più dettagliate utilizzate per tracciare il percorso di esecuzione del codice
DEBUG	Informazioni dettagliate per il debug del sistema
INFO	Messaggi che registrano il normale funzionam ento della funzione
WARN	Messaggi relativi a potenziali errori che possono portare a comportamenti imprevisti se non risolti
ERRORE	Messaggi relativi a problemi che impediscono al codice di funzionare come previsto
FATAL (dettaglio minimo)	Messaggi relativi a errori gravi che causano l'interruzione del funzionamento dell'appl icazione

Quando si seleziona un livello di registro, Lambda invia i log a quel livello e successivamente a Logs. CloudWatch Ad esempio, se imposti il livello di log dell'applicazione di una funzione su WARN, Lambda non invia output log ai livelli INFO e DEBUG. Il livello di log dell'applicazione predefinito per il filtraggio dei log è INFO.

Quando Lambda filtra i log delle applicazioni della funzione, ai messaggi di log senza livello verrà assegnato il livello di log INFO.

Per i log di sistema (i log generati dal servizio Lambda), puoi scegliere tra i seguenti livelli di log.

Livello di log	Utilizzo
DEBUG (dettaglio massimo)	Informazioni dettagliate per il debug del sistema

Livello di log	Utilizzo
INFO	Messaggi che registrano il normale funzionamento della funzione
WARN (dettaglio minimo)	Messaggi relativi a potenziali errori che possono portare a comportamenti imprevisti se non risolti

Quando si seleziona un livello di log, Lambda invia i log di quel livello e di livello inferiore. Ad esempio, se imposti il livello di log di sistema di una funzione su INFO, Lambda non invia output log a livello DEBUG.

Per impostazione predefinita, Lambda imposta il livello di log del sistema su INFO. Con questa impostazione, Lambda invia `"start"` e `"report"` registra automaticamente i messaggi a. CloudWatch Per ricevere log di sistema più o meno dettagliati, modifica il livello di log in DEBUG o WARN. Per visualizzare un elenco dei livelli di log a cui Lambda mappa i diversi log eventi di sistema, consulta la pagina the section called “Strumento di mappatura degli eventi a livello di log di sistema”.

Configurazione del filtraggio a livello di log

Per configurare il filtro a livello di registro dell'applicazione e del sistema per la tua funzione, puoi utilizzare la console Lambda o il (). AWS Command Line Interface AWS CLI Puoi anche configurare il livello di registro di una funzione utilizzando i comandi CreateFunctione l'API UpdateFunctionConfigurationLambda, la risorsa AWS Serverless Application Model (AWS SAM) e la AWS::Serverless::Function CloudFormationAWS::Lambda::Functionrisorsa.

Tieni presente che se imposti il livello di log della funzione nel codice, questa impostazione ha la precedenza su qualsiasi altra impostazione del livello di log che configuri. Ad esempio, se utilizzi il metodo `logging` `setLevel()` di Python per impostare il livello di registrazione di log della funzione su INFO, questa impostazione ha la precedenza su un'impostazione di WARN configurata utilizzando la console Lambda.

Configurazione del livello di log dell'applicazione o di sistema di una funzione esistente (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Nella pagina di configurazione della funzione, scegli Strumenti di monitoraggio e gestione.

4. Nel riquadro Configurazione della registrazione, scegli Modifica.
5. In Contenuto del log, per Formato del log assicurati che sia selezionato JSON.
6. Utilizzando i pulsanti di opzione, seleziona i valori di Livello di log dell'applicazione e Livello di log del sistema desiderati per la funzione.
7. Scegli Save (Salva).

Configurazione del livello di log dell'applicazione o di sistema di una funzione esistente (AWS CLI)

- Per modificare il livello di log dell'applicazione o del sistema di una funzione esistente, utilizza il comando update-function-configuration. Utilizza `--logging-config` per impostare `SystemLogLevel` su `DEBUG`, `INFO` o `WARN`. Imposta `ApplicationLogLevel` su `DEBUG`, `INFO`, `WARN`, `ERROR` o `FATAL`.

```
aws lambda update-function-configuration \
  --function-name myFunction \
  --logging-config LogFormat=JSON,ApplicationLogLevel=ERROR,SystemLogLevel=WARN
```

Configurazione del filtraggio a livello di log durante la creazione di una funzione

- Per configurare il filtraggio a livello di log quando crei una nuova funzione, utilizza `--logging-config` per impostare le chiavi `SystemLogLevel` e `ApplicationLogLevel` nel comando create-function. Imposta `SystemLogLevel` su `DEBUG`, `INFO` o `WARN`. Imposta `ApplicationLogLevel` su `DEBUG`, `INFO`, `WARN`, `ERROR` o `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
```

Strumento di mappatura degli eventi a livello di log di sistema

Per gli eventi di log a livello di sistema generati da Lambda, la tabella seguente definisce il livello di log assegnato a ciascun evento. Per ulteriori informazioni sugli eventi elencati nella tabella, consulta la pagina the section called “Riferimento allo schema `Event`”

Nome evento	Condizione	Livello di log assegnato
initStart	runtimeVersion è impostato	INFO
initStart	runtimeVersion non è impostato	DEBUG
initRuntimeDone	status=success	DEBUG
initRuntimeDone	status=success	WARN
initReport	initializationType!=on-demand	INFO
initReport	initializationType=on-demand	DEBUG
initReport	status=success	WARN
restoreStart	runtimeVersion è impostato	INFO
restoreStart	runtimeVersion non è impostato	DEBUG
restoreRuntimeDone	status=success	DEBUG
restoreRuntimeDone	status=success	WARN
restoreReport	status=success	INFO
restoreReport	status=success	WARN
rapida	-	INFO
runtimeDone	status=success	DEBUG
runtimeDone	status=success	WARN

Nome evento	Condizione	Livello di log assegnato
report	status=success	INFO
report	status=success	WARN
Estensione	state=success	INFO
Estensione	state!=success	WARN
logSubscription	-	INFO
telemetrySubscription	-	INFO
logsDropped	-	WARN

> **Note**
>
> L'the section called “API di telemetria” emette sempre il set completo di eventi della piattaforma. La configurazione del livello dei log di sistema a cui Lambda invia CloudWatch non influisce sul comportamento dell'API Lambda Telemetry.

Filtraggio delle applicazioni a livello di log con runtime personalizzati

Quando configuri il filtraggio a livello di log dell'applicazione per la tua funzione, dietro le quinte Lambda imposta il livello di log dell'applicazione nel runtime utilizzando la variabile di ambiente `AWS_LAMBDA_LOG_LEVEL`. Lambda imposta anche il formato di log della funzione utilizzando la variabile di ambiente `AWS_LAMBDA_LOG_FORMAT`. Puoi utilizzare queste variabili per integrare i controlli di registrazione avanzati di Lambda in un runtime personalizzato.

Per poter configurare le impostazioni di registrazione per una funzione utilizzando un runtime personalizzato con la console Lambda e APIs Lambda AWS CLI, configura il runtime personalizzato per controllare il valore di queste variabili di ambiente. È quindi possibile configurare i logger del runtime in base al formato di log e ai livelli di log selezionati.

Lambda invia automaticamente i log delle funzioni a CloudWatch Logs.

Per impostazione predefinita, Lambda acquisisce automaticamente i log per tutte le chiamate di funzione e li invia a CloudWatch Logs, a condizione che il ruolo di esecuzione della funzione disponga delle autorizzazioni necessarie. <function-name>Per impostazione predefinita, questi log sono archiviati in un gruppo di log denominato /aws/lambda/. Per migliorare il debug, puoi inserire istruzioni di registrazione personalizzate nel codice, che Lambda integrerà perfettamente con CloudWatch Logs. Se desideri che la tua funzione invii i log a un altro gruppo, puoi configurarlo utilizzando la console Lambda, AWS CLI o l'API Lambda. Per ulteriori informazioni, consulta the section called “Configurare i registri CloudWatch delle funzioni”.

Puoi visualizzare i log per le funzioni Lambda tramite la console Lambda, la console CloudWatch, la console AWS Command Line Interface (AWS CLI) o l'API CloudWatch. Per ulteriori informazioni, consulta la pagina Visualizzazione dei CloudWatch log per le funzioni Lambda.

> **Note**
>
> Potrebbero essere necessari da 5 a 10 minuti prima che i log vengano visualizzati dopo una chiamata di funzione.

Autorizzazioni IAM richieste

Il ruolo di esecuzione richiede la seguente autorizzazione per caricare i log in CloudWatch Logs:

- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:PutLogEvents`

Per ulteriori informazioni, consulta Utilizzo delle policy basate su identità (policy IAM) per CloudWatch Logs nella Guida per l'utente di Amazon CloudWatch.

È possibile aggiungere le autorizzazioni CloudWatch Logs utilizzando la policy gestita `AWSLambdaBasicExecutionRole` AWS fornita da Lambda. Per aggiungere questa policy al ruolo, esegui il seguente comando:

```
aws iam attach-role-policy --role-name your-role --policy-arn arn:aws:iam::aws:policy/
service-role/AWSLambdaBasicExecutionRole
```

Per ulteriori informazioni, consulta the section called “AWS politiche gestite”.

Prezzi

Non sono previsti costi aggiuntivi per l'utilizzo dei log di Lambda, ma vengono applicati i costi CloudWatch Logs standard. Per ulteriori informazioni, consultare la pagina dei prezzi di CloudWatch

Configurazione dei gruppi di CloudWatch log

Per impostazione predefinita, crea CloudWatch automaticamente un gruppo di log denominato in base alla funzione quando viene richiamata `/aws/lambda/<function name>` per la prima volta. Per configurare la tua funzione per l'invio dei log a un gruppo di log esistente o per creare un nuovo gruppo di log per la funzione, puoi utilizzare la console Lambda o la AWS CLI. Puoi anche configurare gruppi di log personalizzati utilizzando i comandi API CreateFunctione UpdateFunctionConfigurationLambda e la risorsa AWS Serverless Application Model (AWS SAM) AWS: :Serverless: :Function.

È possibile configurare più funzioni Lambda per inviare i log allo stesso CloudWatch gruppo di log. Ad esempio, è possibile utilizzare un singolo gruppo di log per archiviare i log per tutte le funzioni Lambda che costituiscono una particolare applicazione. Quando si utilizza un gruppo di log personalizzato per una funzione Lambda, i flussi di log creati da Lambda includono il nome e la versione della funzione. Ciò garantisce che la mappatura tra i messaggi di log e le funzioni venga preservata, anche se si utilizza lo stesso gruppo di log per più funzioni.

Il formato di denominazione per i flussi di log per i gruppi di log personalizzati segue questa convenzione:

```
YYYY/MM/DD/<function_name>[<function_version>][<execution_environment_GUID>]
```

Tieni presente che quando configuri un gruppo di log personalizzato, il nome selezionato per il gruppo di log deve seguire le regole di denominazione dei CloudWatch log. Inoltre, i nomi dei gruppi di log personalizzati non devono cominciare con la stringa `aws/`. Se crei un gruppo di log personalizzato che comincia con `aws/`, Lambda non sarà in grado di crearlo. Di conseguenza, i log della funzione non verranno inviati a. CloudWatch

Modifica del gruppo di log di una funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.

3. Nella pagina di configurazione della funzione, scegli Strumenti di monitoraggio e gestione.
4. Nel riquadro Configurazione della registrazione, scegli Modifica.
5. Nel riquadro Logging group, per il gruppo di CloudWatch log, scegli Personalizzato.
6. In Gruppo di log personalizzato, inserisci il nome del gruppo di CloudWatch log a cui desideri che la funzione invii i log. Se immetti il nome di un gruppo di log esistente, la funzione utilizzerà quel gruppo. Se non esiste alcun gruppo di log con il nome immesso, Lambda creerà un nuovo gruppo di log per la funzione con tale nome.

Modifica del gruppo di log di una funzione (AWS CLI)

- Per modificare il gruppo di log di una funzione esistente, utilizza il comando update-function-configuration.

```
aws lambda update-function-configuration \
  --function-name myFunction \
  --logging-config LogGroup=myLogGroup
```

Definizione di un gruppo di log personalizzato durante la creazione di una funzione (AWS CLI)

- Per specificare un gruppo di log personalizzato quando si crea una nuova funzione Lambda utilizzando AWS CLI, utilizzare l'`--logging-config`opzione. Il comando di esempio seguente crea una funzione Lambda Node.js che invia i log a un gruppo di log denominato `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
```

Autorizzazioni del ruolo di esecuzione

Per inviare i log a CloudWatch Logs, la funzione deve disporre dell'autorizzazione logs:. PutLogEvents Quando configuri il gruppo di log della tua funzione utilizzando la console Lambda, Lambda aggiungerà questa autorizzazione al ruolo nelle seguenti condizioni:

- La destinazione del servizio è impostata su Logs CloudWatch
- Il ruolo di esecuzione della funzione non dispone delle autorizzazioni per caricare i log in CloudWatch Logs (la destinazione predefinita)

> **Note**
>
> Lambda non aggiunge alcuna autorizzazione Put per le destinazioni di log di Amazon S3 o Firehose.

Quando Lambda aggiunge questo permesso, concede alla funzione il permesso di inviare i log a qualsiasi gruppo di log CloudWatch Logs.

Per evitare che Lambda aggiorni automaticamente il ruolo di esecuzione della funzione e modificarlo invece manualmente, espandi Autorizzazioni e deseleziona Aggiungi autorizzazioni richieste.

Quando configuri il gruppo di log della tua funzione utilizzando AWS CLI, Lambda non aggiungerà automaticamente l'`logs:PutLogEvents`autorizzazione. Aggiungi l'autorizzazione al ruolo di esecuzione della tua funzione, se non ne dispone già. Questa autorizzazione è inclusa nella politica AWSLambdaBasicExecutionRolegestita.

CloudWatch registrazione per istanze gestite Lambda

Quando si utilizzano le istanze gestite da Lambda, ci sono ulteriori considerazioni per l'invio dei log ai log: CloudWatch

Requisiti di rete VPC

Le istanze gestite Lambda vengono eseguite su istanze di proprietà del cliente all'interno EC2 del tuo VPC. Per inviare i log a CloudWatch Logs and trace to X-Ray, devi assicurarti che AWS APIs siano instradabili dal tuo VPC. A questo scopo, sono disponibili numerose opzioni:

- AWS PrivateLink (consigliato): AWS PrivateLinkDa utilizzare per creare endpoint VPC per i servizi CloudWatch Logs e X-Ray. Ciò consente alle istanze di accedere a questi servizi in modo privato senza richiedere un gateway Internet o un gateway NAT. Per ulteriori informazioni, consulta Utilizzo dei CloudWatch log con gli endpoint VPC dell'interfaccia.
- Gateway NAT: configura un gateway NAT per consentire l'accesso a Internet in uscita dalle sottoreti private.

- Gateway Internet: per le sottoreti pubbliche, assicurati che il tuo VPC abbia un gateway Internet configurato.

Se CloudWatch Logs o APIs X-Ray non sono instradabili dal tuo VPC, i log e le tracce delle funzioni non verranno consegnati.

Richiamazioni simultanee e attribuzione dei log

Gli ambienti di esecuzione di Lambda Managed Instances possono elaborare più chiamate contemporaneamente. Quando vengono eseguite più chiamate contemporaneamente, le relative voci di registro vengono interlacciate nello stesso flusso di log. Per filtrare e analizzare efficacemente i log delle chiamate simultanee, è necessario assicurarsi che ogni voce di registro includa l'ID della richiesta. AWS

Consigliamo uno dei seguenti approcci:

- Usa logger di runtime Lambda predefiniti (consigliato): le librerie di registrazione predefinite fornite dai runtime gestiti da Lambda includono automaticamente l'ID della richiesta in ogni voce di registro.
- Implementa la registrazione JSON strutturata: se stai creando un runtime personalizzato o hai bisogno di una registrazione personalizzata, implementa log in formato JSON che includono l'ID della richiesta in ogni voce. Le istanze gestite Lambda supportano solo il formato di log JSON. Includi il `requestId` campo nei log JSON per abilitare il filtraggio per chiamata:

```
{
  "timestamp": "2025-01-15T10:30:00.000Z",
  "level": "INFO",
  "requestId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "message": "Processing request"
}
```

Con l'attribuzione dell'ID della richiesta, puoi filtrare le voci di registro di CloudWatch Logs per una chiamata specifica utilizzando le query di Logs Insights. CloudWatch Esempio:

```
fields @timestamp, @message
| filter requestId = "a1b2c3d4-e5f6-7890-abcd-ef1234567890"
| sort @timestamp asc
```

Per ulteriori informazioni sui requisiti di registrazione delle istanze gestite di Lambda, vedere. Comprendere l'ambiente di esecuzione di Lambda Managed Instances

Visualizzazione dei CloudWatch log per le funzioni Lambda

Puoi visualizzare CloudWatch i log di Amazon per la tua funzione Lambda utilizzando la console Lambda, CloudWatch la console o il (). AWS Command Line Interface AWS CLI Segui le istruzioni riportate nelle sezioni seguenti per accedere ai log della funzione.

Trasmetti in streaming i log delle funzioni con Logs Live Tail CloudWatch

Amazon CloudWatch Logs Live Tail ti aiuta a risolvere rapidamente i problemi delle tue funzioni visualizzando un elenco in streaming di nuovi eventi di registro direttamente nella console Lambda. Puoi visualizzare e filtrare i log importati dalle funzioni Lambda in tempo reale, in modo da poter rilevare e risolvere rapidamente i problemi.

> **Note**
>
> Le sessioni Live Tail comportano costi al minuto in base al tempo di utilizzo della sessione. Per ulteriori informazioni sui prezzi, consulta la pagina CloudWatch dei prezzi di Amazon.

Confronto tra Live Tail e --log-type Tail

Esistono diverse differenze tra CloudWatch Logs Live Tail e l'opzione LogType: Tail nell'API Lambda `--log-type Tail` (in AWS CLI):

- `--log-type Tail` restituisce solo i primi 4 KB dei log di invocazione. Live Tail non condivide questo limite e può ricevere fino a 500 eventi del log al secondo.
- `--log-type Tail` acquisisce e invia i log con la risposta, il che può influire sulla latenza di risposta della funzione. Live Tail non influisce sulla latenza di risposta della funzione.
- `--log-type Tail` supporta solo chiamate sincrone. Live Tail funziona sia per le invocazioni sincrone che asincrone.

> **Note**
>
> Lambda Managed Instances non supporta l'opzione. `--log-type Tail` Usa CloudWatch Logs Live Tail o interroga direttamente CloudWatch Logs per visualizzare i log delle funzioni delle istanze gestite.

Permissions

Le seguenti autorizzazioni sono necessarie per avviare e interrompere CloudWatch le sessioni di Logs Live Tail:

- `logs:DescribeLogGroups`
- `logs:StartLiveTail`
- `logs:StopLiveTail`

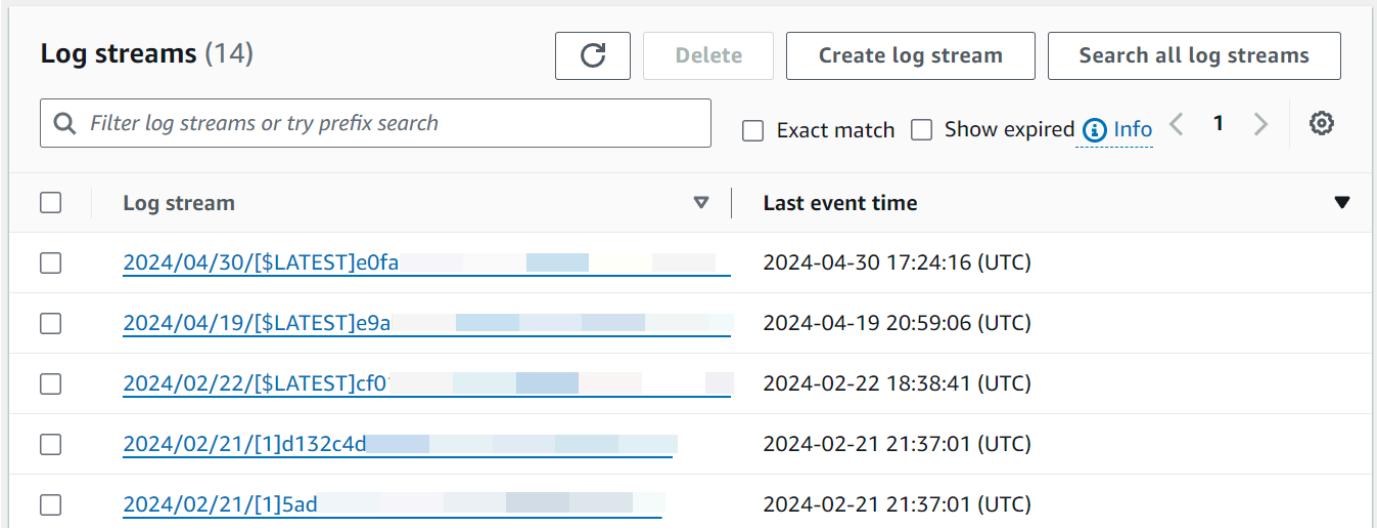
Avviare una sessione Live Tail nella console Lambda

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere il nome della funzione.
3. Seleziona la scheda Test.
4. Nel riquadro Test event, scegli CloudWatch Logs Live Tail.
5. Per Seleziona gruppi di log, il gruppo di log della funzione è selezionato per impostazione predefinita. È possibile selezionare fino a cinque gruppi di log alla volta.
6. (Facoltativo) Per visualizzare solo gli eventi del log che contengono determinate parole o altre stringhe, inserisci la parola o la stringa nella casella Aggiungi modello di filtro. I filtri fanno distinzione tra maiuscole e minuscole. Puoi includere più termini e operatori di modelli in questo campo, comprese le espressioni regolari (regex). Per ulteriori informazioni sulla sintassi dei pattern, consulta Filter pattern syntax. nella Amazon CloudWatch Logs User Guide.
7. Scegli Avvia. I log eventi corrispondenti vengono visualizzati nella finestra.
8. Per arrestare la sessione Live Tail, scegli Arresta.

> **Note**
>
> La sessione Live Tail si interrompe automaticamente dopo 15 minuti di inattività o quando la sessione della console Lambda scade.

Accedere ai log della funzione tramite la console

1. Aprire la pagina Funzioni della console Lambda.
2. Seleziona una funzione.
3. Selezionare la scheda Monitor (Monitora).
4. Scegli Visualizza CloudWatch i log per aprire la console. CloudWatch
5. Scorri verso il basso e scegli il flusso di log per le invocazioni delle funzioni che desideri esaminare.

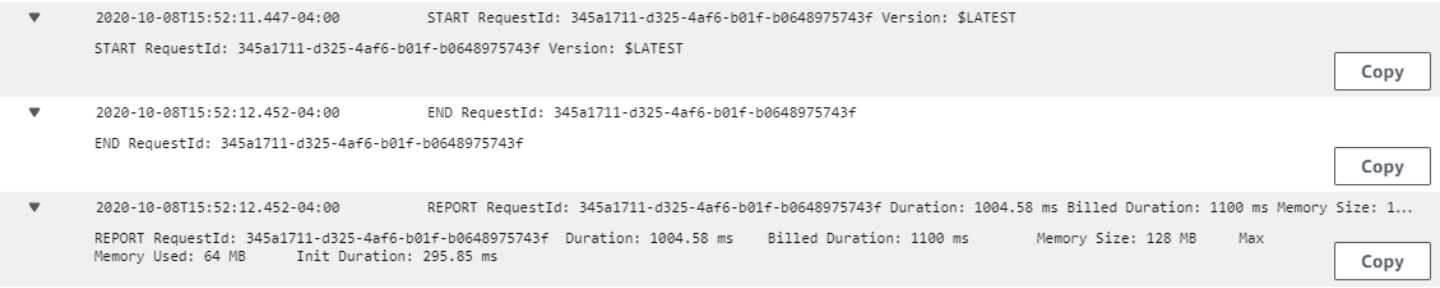


Ogni istanza di una funzione Lambda ha un flusso di log dedicato. Se una funzione aumenta, ogni istanza simultanea ha il suo flusso di log. Ogni volta che viene creato un nuovo ambiente di esecuzione in risposta a una chiamata, viene generato un nuovo flusso di log. La convenzione di denominazione per i flussi di log è:

```
YYYY/MM/DD[Function version][Execution environment GUID]
```

Un singolo ambiente di esecuzione scrive sullo stesso flusso di log durante il suo ciclo di vita. Il flusso di log contiene i messaggi provenienti da quell'ambiente di esecuzione e anche qualsiasi output

del codice della funzione Lambda. Ogni messaggio ha un timestamp, compresi i log personalizzati. Anche se la funzione non registra alcun output dal codice, vengono generate tre istruzioni di log minime per ogni invocazione (START, END e REPORT):



Questi log mostrano:

- RequestId— si tratta di un ID univoco generato per richiesta. Se la funzione Lambda ritenta una richiesta, questo ID non cambia e viene visualizzato nei log per ogni tentativo successivo.
- Start/End: contrassegnano una singola chiamata ai segnalibri, quindi ogni riga di registro compresa tra queste appartiene alla stessa invocazione.
- Durata: il tempo totale di invocazione per la funzione di gestione, codice escluso. `INIT`
- Durata fatturata: applica la logica di arrotondamento ai fini della fatturazione.
- Dimensione della memoria: la quantità di memoria allocata alla funzione.
- Memoria massima utilizzata: la quantità massima di memoria utilizzata durante la chiamata.
- Durata di inizializzazione: il tempo impiegato per eseguire la `INIT` sezione di codice, al di fuori del gestore principale.

Accedi ai log con AWS CLI

AWS CLI È uno strumento open source che consente di interagire con i AWS servizi utilizzando i comandi nella shell della riga di comando. Per completare le fasi riportate in questa sezione, è necessario disporre della AWS CLI versione 2.

È possibile utilizzare AWS CLI per recuperare i log per una chiamata utilizzando l'opzione di comando `--log-type`. La risposta include un campo `LogResult` che contiene fino a 4 KB di log con codifica base64 del richiamo.

Example recuperare un ID di log

Nell'esempio seguente viene illustrato come recuperare un ID di log dal `LogResult` campo per una funzione denominata `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Verrà visualizzato l'output seguente:

```
{
    "StatusCode": 200,
    "LogResult":
 "U1RBUlQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificare i log

Nello stesso prompt dei comandi, utilizzare l'`base64` utilità per decodificare i log. Nell'esempio seguente viene illustrato come recuperare i log codificati in base64 per `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
```

L'cli-binary-formatopzione è obbligatoria se si utilizza la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 2.

Verrà visualizzato l'output seguente:

```
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
```

L'utilità `base64` è disponibile su Linux, macOS e Ubuntu su Windows. Gli utenti macOS potrebbero dover utilizzare `base64 -D`.

Example Script get-logs.sh

Nello stesso prompt dei comandi, utilizzare lo script seguente per scaricare gli ultimi cinque eventi di log. Lo script utilizza `sed` per rimuovere le virgolette dal file di output e rimane in sospensione per 15 secondi in attesa che i log diventino disponibili. L'output include la risposta di Lambda e l'output del comando `get-log-events`.

Copiare il contenuto del seguente esempio di codice e salvare nella directory del progetto Lambda come `get-logs.sh`.

L'cli-binary-formatopzione è obbligatoria se utilizzi la AWS CLI versione 2. Per rendere questa impostazione come predefinita, esegui `aws configure set cli-binary-format raw-in-base64-out`. Per ulteriori informazioni, consulta la pagina AWS CLI supported global command line options nella Guida per l'utente di AWS Command Line Interface versione 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 (solo) macOS e Linux

Nello stesso prompt dei comandi, gli utenti macOS e Linux potrebbero dover eseguire il seguente comando per assicurarsi che lo script sia eseguibile.

```
chmod -R 755 get-logs.sh
```

Example recuperare gli ultimi cinque eventi di log

Nello stesso prompt dei comandi, eseguire lo script seguente per ottenere gli ultimi cinque eventi di log.

```
./get-logs.sh
```

Verrà visualizzato l'output seguente:

```
{
    "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"
}
```

Analisi dei log e registrazione strutturata

Con CloudWatch Logs Insights, puoi cercare e analizzare i dati di registro utilizzando una sintassi di query specializzata. Esegue interrogazioni su più gruppi di log e fornisce potenti filtri utilizzando il pattern matching tra espressioni glob e regolari.

Puoi sfruttare queste funzionalità implementando la registrazione strutturata nelle tue funzioni Lambda. La registrazione strutturata organizza i log in un formato predefinito, semplificando le interrogazioni. L'utilizzo dei livelli di registro è un primo passo importante per generare log compatibili con i filtri che separano i messaggi informativi dagli avvisi o dagli errori. Ad esempio, si consideri il seguente codice Node.js:

```
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")
}
```

Il file di CloudWatch registro risultante contiene un campo separato che specifica il livello di registro:

```
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
```

Una query di CloudWatch Logs Insights può quindi filtrare a livello di registro. Ad esempio, per ricercare solo gli errori, è possibile utilizzare la seguente query:

```
fields @timestamp, @message
| filter @message like /ERROR/
| sort @timestamp desc
```

Registrazione strutturata JSON

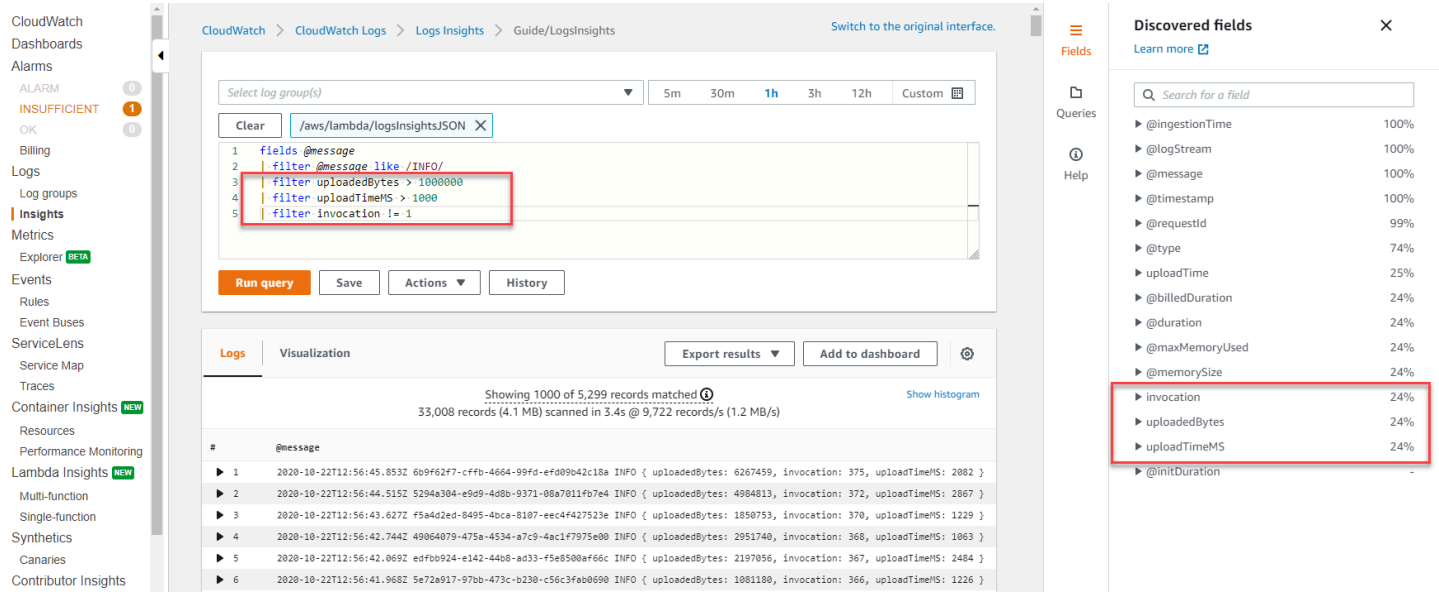
JSON è comunemente usato per fornire una struttura per i log delle applicazioni. Nell'esempio seguente, i log sono stati convertiti in JSON per generare tre valori distinti:

```
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
```

La funzionalità CloudWatch Logs Insights rileva automaticamente i valori nell'output JSON e analizza i messaggi come campi, senza la necessità di glob o espressioni regolari personalizzate. Utilizzando i log strutturati in JSON, la seguente query trova le invocazioni in cui il file caricato era più grande di 1 MB, il tempo di caricamento era superiore a 1 secondo e l'invocazione non era un avvio a freddo:

```
fields @message
| filter @message like /INFO/
| filter uploadedBytes > 1000000
| filter uploadTimeMS > 1000
| filter invocation != 1
```

Questa query potrebbe produrre il seguente risultato:



I campi rilevati in JSON vengono compilati automaticamente nel menu Campi rilevati sul lato destro. I campi standard emessi dal servizio Lambda hanno il prefisso '@' e puoi eseguire query su questi campi nello stesso modo. I log Lambda includono sempre i campi @timestamp, @logStream, @message, @requestId, @duration, @billedDuration, @type, @maxMemoryUsed, @memorySize. Se X-Ray è abilitato per una funzione, i log includono anche @ e @xrayTraceId . xraySegmentId

Quando un'origine di AWS eventi come Amazon S3, Amazon SQS o EventBridge Amazon richiama la tua funzione, l'intero evento viene fornito alla funzione come input di un oggetto JSON. Registrando questo evento nella prima riga della funzione, puoi quindi eseguire query su qualsiasi campo annidato utilizzando Logs Insights. CloudWatch

Query utili di Insights

La tabella seguente mostra esempi di query Insights che possono essere utili per il monitoraggio delle funzioni Lambda.

Description	Esempio di sintassi della query

Gli ultimi 100 errori	`fields Timestamp, LogLevel, Message`
	`| filter LogLevel == "ERR"`
	`| sort @timestamp desc`
	`| limit 100`

Le prime 100 invocazioni con il fatturato più alto	`filter @type = "REPORT"`
	`| fields @requestId, @billedDuration`
	`| sort by @billedDuration desc`
	`| limit 100`

Percentuale di avvii a freddo sul totale delle invocazioni	`filter @type = "REPORT"`
	`| stats sum(strcontains(@message,`
	`"Init Duration"))/count(*) * 100 as`
	`coldStartPct, avg(@duration)`
	`by bin(5m)`

Rapporto percentile della durata 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)`

Description	Esempio di sintassi della query
Rapporto percentile sull'utilizzo della memoria Lambda	`filter @type="REPORT" | stats avg(@maxMemoryUsed/1024/1024) as mean_MemoryUsed, min(@maxMemoryUsed/1024/1024) as min_MemoryUsed, max(@maxMemoryUsed/1024/1024) as max_MemoryUsed, percentile(@maxMemoryUsed/1024/1024, 95) as Percentile95`
Invocazioni che utilizzano il 100% della memoria assegnata	`filter @type = "REPORT" and @maxMemoryUsed=@memorySize | stats count_distinct(@requestId) by bin(30m)`
Memoria media utilizzata tra le invocazioni	`avgMemoryUsedPERC, avg(@billedDuration) as avgDurationMS by bin(5m)`
Visualizzazione delle statistiche sulla memoria	`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 avgDurationMS by bin(30m)`

Description	Esempio di sintassi della query
Invocazioni in cui Lambda è uscita	`filter @message like /Process exited/ | stats count() by bin(30m)`
Invocazioni scadute	`filter @message like /Task timed out/ | stats count() by bin(30m)`
Rapporto sulla latenza	`filter @type = "REPORT" | stats avg(@duration), max(@duration), min(@duration) by bin(5m)`
Memoria sovradimensionata	`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`

Visualizzazione dei log e pannelli di controllo

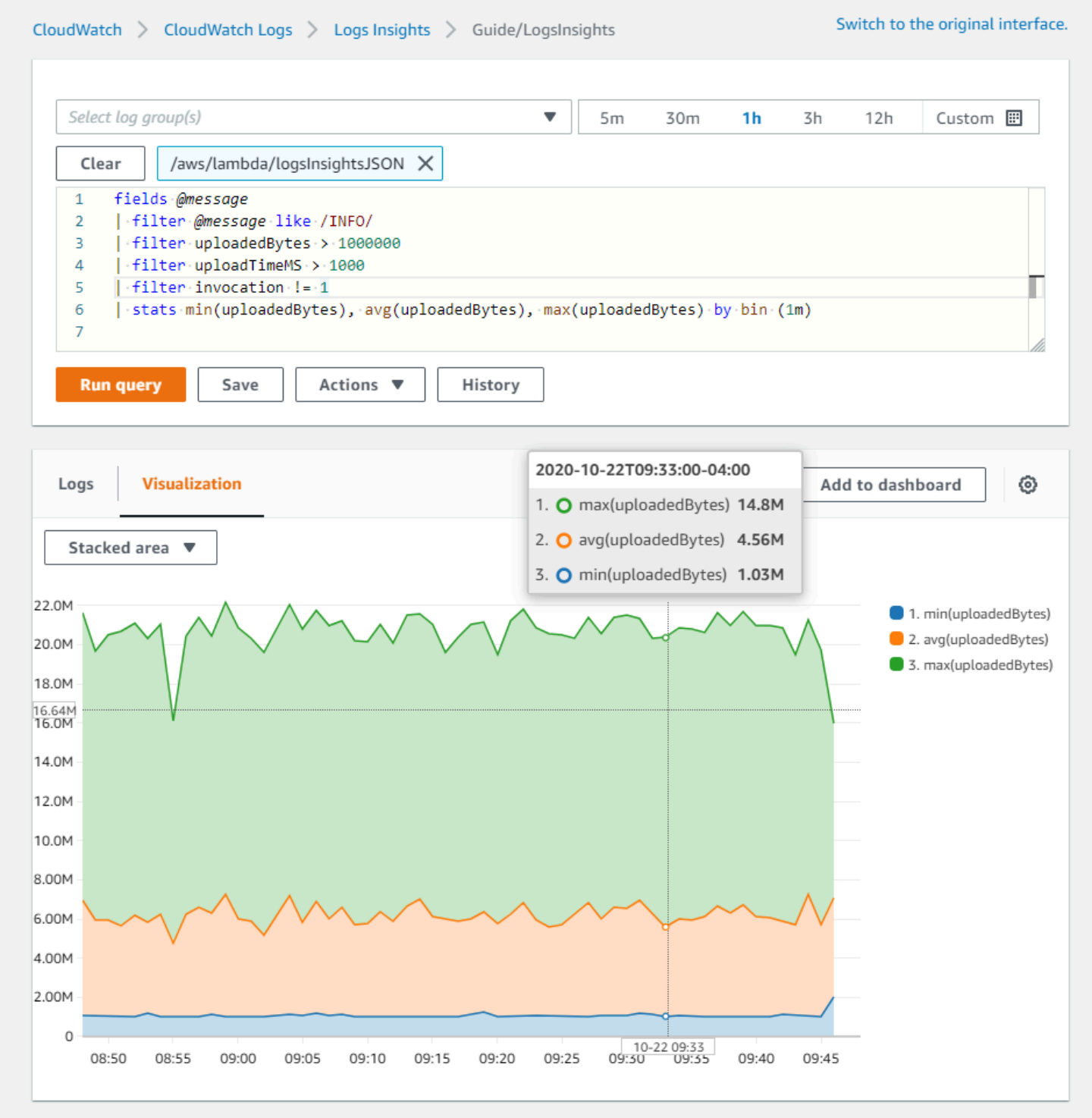
Per qualsiasi query di CloudWatch Logs Insights, puoi esportare i risultati in formato markdown o CSV. In alcuni casi, potrebbe essere più utile creare visualizzazioni a partire dalle query, purché esista almeno una funzione di aggregazione. La `stats` funzione consente di definire aggregazioni e raggruppamenti.

L'esempio precedente di logInsightsJSON filtrava in base alla dimensione e al tempo di caricamento ed escludeva le prime invocazioni. Ciò ha prodotto una tabella di dati. Per monitorare un sistema di produzione, può essere più utile visualizzare le dimensioni minime, massime e medie dei file per individuare i valori anomali. Per fare ciò, applica la funzione stats agli aggregati richiesti e raggruppa in base a un valore temporale come ogni minuto:

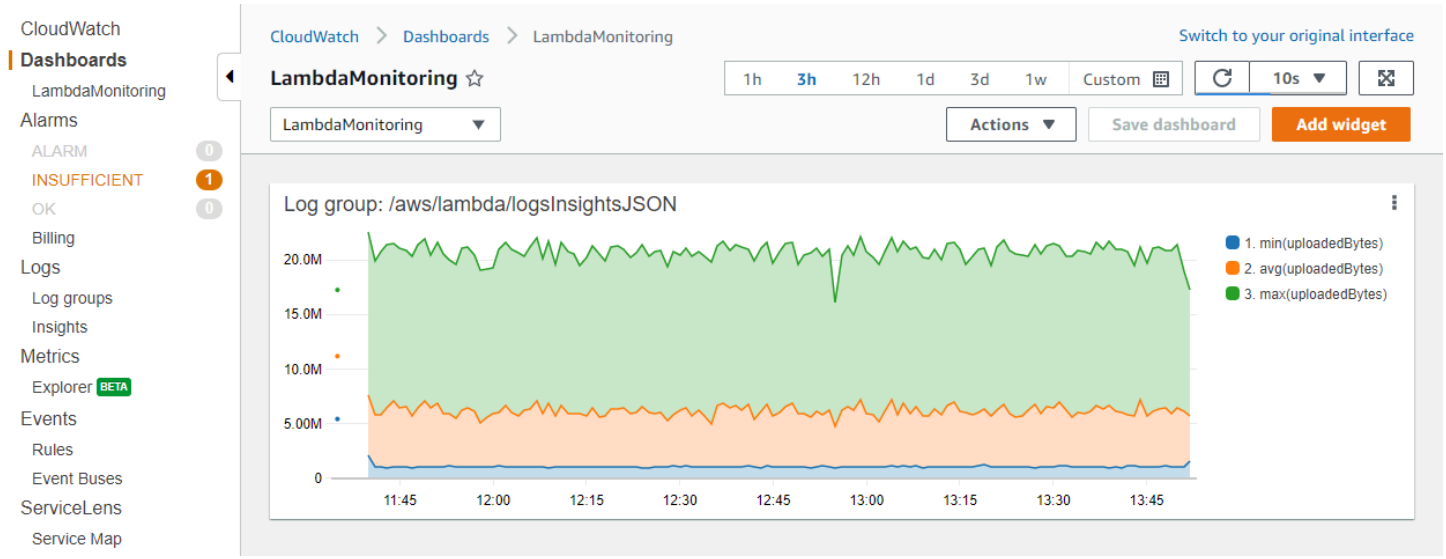
Ad esempio, si consideri la seguente interrogazione. Questa è la stessa query di esempio della the section called “Registrazione strutturata JSON” sezione, ma con funzioni di aggregazione aggiuntive:

```
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)
```

Abbiamo incluso questi aggregati perché potrebbe essere più utile visualizzare le dimensioni minime, massime e medie dei file per individuare i valori anomali. Puoi visualizzare i risultati nella scheda Visualizzazione:



Dopo aver completato la creazione della visualizzazione, puoi facoltativamente aggiungere il grafico a una dashboard. CloudWatch Per fare ciò, scegli Aggiungi al pannello di controllo sopra la visualizzazione. Ciò aggiunge la query come widget e consente di selezionare intervalli di aggiornamento automatici, semplificando il monitoraggio continuo dei risultati:



Invio dei log delle funzioni Lambda a Firehose

La console Lambda offre ora la possibilità di inviare registri delle funzioni a Firehose. Ciò consente lo streaming in tempo reale dei log verso varie destinazioni supportate da Firehose, inclusi strumenti di analisi di terze parti ed endpoint personalizzati.

> **Note**
>
> È possibile configurare i log delle funzioni Lambda da inviare a Firehose utilizzando la console AWS CLI Lambda e tutti gli SDK. AWS CloudFormation AWS

Prezzi

Per informazioni sui prezzi, consulta i prezzi di Amazon CloudWatch.

Autorizzazioni richieste per la destinazione del registro di Firehose

Quando si utilizza la console Lambda per configurare Firehose come destinazione di registro della funzione, è necessario:

1. Le autorizzazioni IAM richieste per utilizzare CloudWatch Logs with Lambda.
2. Per configurare i filtri di abbonamento con Firehose. Questo filtro definisce quali eventi di log vengono distribuiti al tuo flusso Firehose.

Invio dei log delle funzioni Lambda a Firehose

Nella console Lambda, è possibile inviare i log delle funzioni direttamente a Firehose dopo aver creato una nuova funzione. Per farlo, completa le seguenti fasi.

1. Accedi alla console di gestione e apri laconsole .
2. Scegli il nome della tua funzione.
3. Scegli la scheda Configurazione.
4. Scegliete la scheda Strumenti operativi e di monitoraggio.
5. Nel riquadro Configurazione della registrazione, scegli Modifica.
6. Nella sezione «Contenuto del registro», seleziona un formato di registro.
7. Nella sezione Step 1 (Fase 1) completare i passaggi seguenti.
   a. Seleziona un servizio di destinazione.
   b. Scegli un gruppo di log esistente o creane uno nuovo.

   > **Note**
   >
   > Se scegli un gruppo di log esistente per una destinazione Firehose, assicurati che il gruppo di log che scegli sia un tipo di gruppo di `Delivery` log.

   c. Scegliete uno stream Firehose.
   d. Verrà visualizzato il gruppo di log `Delivery` CloudWatch.
8. Selezionare Salva.

> **Note**
>
> Se il ruolo IAM fornito nella console non dispone dell'autorizzazione richiesta, la configurazione della destinazione avrà esito negativo. Per risolvere questo problema, fare riferimento a Autorizzazioni richieste per la destinazione del registro di Firehose per fornire le autorizzazioni richieste.

Registri multi-account

È possibile configurare Lambda per inviare i log al flusso di distribuzione di Firehose in un account diverso. AWS Ciò richiede l'impostazione di una destinazione e la configurazione delle autorizzazioni appropriate in entrambi gli account.

Per istruzioni dettagliate sulla configurazione della registrazione tra account, inclusi i ruoli e le policy IAM richiesti, consulta Configurazione di un nuovo abbonamento tra account nella documentazione di CloudWatch Logs.

Invio di log delle funzioni Lambda ad Amazon S3

Puoi configurare la tua funzione Lambda per inviare i log direttamente ad Amazon S3 utilizzando la console Lambda. Questa funzionalità offre una soluzione economica per l'archiviazione dei log a lungo termine e consente potenti opzioni di analisi utilizzando servizi come Athena.

> **Note**
>
> Puoi configurare i log delle funzioni Lambda da inviare ad Amazon S3 utilizzando la console AWS CLI Lambda e tutti gli SDK. AWS CloudFormation AWS

Prezzi

Per informazioni sui prezzi, consulta i prezzi di Amazon CloudWatch.

Autorizzazioni necessarie per la destinazione del log Amazon S3

Quando utilizzi la console Lambda per configurare Amazon S3 come destinazione dei log della tua funzione, devi:

1. Le autorizzazioni IAM richieste per utilizzare CloudWatch Logs with Lambda.
2. Da a Configura un filtro di sottoscrizioni CloudWatch Logs per inviare i log delle funzioni Lambda ad Amazon S3. Questo filtro definisce quali eventi di log vengono distribuiti al bucket Amazon S3.

Configura un filtro di sottoscrizioni CloudWatch Logs per inviare i log delle funzioni Lambda ad Amazon S3

Per inviare i log da CloudWatch Logs ad Amazon S3, devi creare un filtro di sottoscrizione. Questo filtro definisce quali eventi di log vengono distribuiti al bucket Amazon S3. L'argomento deve trovarsi nella stessa Regione del bucket Amazon S3.

Creazione di un filtro di sottoscrizione per Amazon S3

1. Crea un bucket Amazon Simple Storage Service (Amazon S3). Consigliamo di utilizzare un bucket creato appositamente per CloudWatch Logs. Tuttavia, se intendi utilizzare un bucket esistente, puoi passare alla fase 2.

   Esegui il comando seguente, sostituendo il segnaposto Regione con la Regione che desideri utilizzare:

   ```
   aws s3api create-bucket --bucket amzn-s3-demo-bucket2 --create-bucket-configuration
    LocationConstraint=region
   ```

   > **Note**
   >
   > `amzn-s3-demo-bucket2`è un esempio del nome del bucket Amazon S3. È riservato. Affinché questa procedura funzioni, è necessario sostituirla con il nome univoco del bucket Amazon S3.

   Di seguito è riportato un output di esempio:

   ```
   {
       "Location": "/amzn-s3-demo-bucket2"
   }
   ```

2. Crea il ruolo IAM che concede a File di log CloudWatch l'autorizzazione per inserire dati nel flusso. Questa policy include una chiave di contesto della condizione globale per prevenire il problema di sicurezza noto come "confused deputy". Per ulteriori informazioni sulla prevenzione del problema del "confused deputy", consulta .

   a. In primo luogo, utilizza un editor di testo per creare una policy di attendibilità in un file `~/TrustPolicyForCWL.json`, come segue:

```
{
    "Statement": {
        "Effect": "Allow",
        "Principal": { "Service": "logs.amazonaws.com" },
        "Condition": {
            "StringLike": {
                "aws:SourceArn": "arn:aws:logs:region:123456789012:*"
            }
         },
        "Action": "sts:AssumeRole"
    }
}
```

b. Utilizza il comando create-role per creare un ruolo IAM, specificando il file della policy di attendibilità. Annota il valore Role.Arn restituito, poiché ne avrai bisogno anche in una fase successiva:

```
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. Crea una policy di autorizzazioni per definire che operazioni possono essere eseguite da CloudWatch Logs nel tuo account. In primo luogo, utilizza un editor di testo per creare una policy di autorizzazione in un file `~/PermissionsForCWL.json`:

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:PutObject"],
      "Resource": ["arn:aws:s3:::amzn-s3-demo-bucket2/*"]
    }
  ]
}
```

   Associa la policy di autorizzazioni al ruolo inserendo il comando seguente:

```
aws iam put-role-policy --role-name CWLtoS3Role --policy-name Permissions-Policy-For-
S3 --policy-document file://~/PermissionsForCWL.json
```

4. Scegli un gruppo di log esistente o creane uno nuovo.

```
aws logs create-log-group --log-group-name my-logs --log-group-class DELIVERY --
region REGION_NAME
```

5. Configurazione della destinazione

```
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
```

Invio di log delle funzioni Lambda ad Amazon S3

Nella console Lambda, puoi inviare i log delle funzioni direttamente ad Amazon S3 dopo aver creato una nuova funzione. Per farlo, completa le seguenti fasi.

1. Accedi alla console di gestione e apri laconsole .
2. Scegli il nome della tua funzione.
3. Scegli la scheda Configurazione.
4. Scegli la scheda Strumenti operativi e di monitoraggio.
5. Nel riquadro Configurazione della registrazione, scegli Modifica.
6. Nella sezione «Contenuto del registro», seleziona un formato di registro.
7. Nella sezione Step 1 (Fase 1) completare i passaggi seguenti.
   a. Seleziona un servizio di destinazione.
   b. Scegli un gruppo di log esistente o creane uno nuovo.

   > **Note**
   >
   > Se scegli un gruppo di log esistente per una destinazione Amazon S3, assicurati che il gruppo di log che scegli sia un tipo di gruppo di `Delivery` log.

   c. Scegli un bucket Amazon S3 come destinazione per i log delle funzioni.
   d. Verrà visualizzato il gruppo di log `Delivery` CloudWatch.
8. Selezionare Salva.

> **Note**
>
> Se il ruolo IAM fornito nella console non dispone delle autorizzazioni richieste, la configurazione della destinazione avrà esito negativo. Per risolvere questo problema, consulta Autorizzazioni richieste per la destinazione dei log di Amazon S3.

Registrazione multi-account

È possibile configurare Lambda per inviare i log a un bucket Amazon S3 in un account. AWS Ciò richiede l'impostazione di una destinazione e la configurazione delle autorizzazioni appropriate in entrambi gli account.

Per istruzioni dettagliate sulla configurazione della registrazione tra account, inclusi i ruoli e le policy IAM richiesti, consulta Configurazione di un nuovo abbonamento tra account nella documentazione di CloudWatch Logs.

Registrazione delle chiamate API di Esploratore di risorse AWS tramite AWS CloudTrail

AWS Lambda si integra con AWS CloudTrail, un servizio che offre un record delle operazioni eseguite da un utente, un ruolo o un Servizio AWS. CloudTrail acquisisce le chiamate API per Lambda come eventi. Le chiamate acquisite includono le chiamate dalla console di Lambda e le chiamate di codice alle operazioni delle API di Lambda. Con le informazioni raccolte da CloudTrail è possibile determinare la richiesta effettuata a Lambda, l'indirizzo IP da cui è partita la richiesta, il momento in cui è stata eseguita e altri dettagli.

Ogni evento o voce di log contiene informazioni sull'utente che ha generato la richiesta. Le informazioni di identità consentono di determinare quanto segue:

- Se la richiesta è stata effettuata con le credenziali utente root o utente.
- Se la richiesta è stata effettuata per conto di un utente del Centro identità IAM.
- Se la richiesta è stata effettuata con le credenziali di sicurezza temporanee per un ruolo o un utente federato.
- Se la richiesta è stata effettuata da un altro Servizio AWS.

CloudTrail è attivo nel tuo Account AWS quando create l'account e hai automaticamente accesso alla cronologia degli eventi di CloudTrail. La cronologia degli eventi di CloudTrail fornisce una registrazione visualizzabile, ricercabile, scaricabile e immutabile degli eventi di gestione verificatisi negli ultimi 90 giorni in una Regione AWS. Per ulteriori informazioni, consulta Working with CloudTrail Event history nella Guida per l'utente di AWS CloudTrail. Non sono previsti costi CloudTrail per la visualizzazione della cronologia degli eventi.

Per una registrazione continua degli eventi nell'Account AWS oltre i 90 giorni, creare un trail o un datastore di eventi Data Lake CloudTrail.

Trail CloudTrail

> Un trail abilita la distribuzione da parte di CloudTrail dei file di log in un bucket Amazon S3. Tutti i trail creati utilizzando la Console di gestione AWS sono multi-regione. È possibile creare un trail per una singola Regione o per più Regioni tramite AWS CLI. Si consiglia di creare un trail per più Regioni in quanto consente di acquisire l'attività in tutte le Regioni AWS dell'account. Se si crea un trail per una singola Regione, è possibile visualizzare solo gli eventi registrati nella Regione

AWS del trail. Per ulteriori informazioni sui trail, consulta Creating a trail for your Account AWS e Creating a trail for an organization nella Guida per l'utente di AWS CloudTrail.

Puoi fornire gratuitamente una copia dei tuoi eventi di gestione in corso al tuo bucket Amazon S3 da CloudTrail creando un percorso, tuttavia dovranno essere considerati i costi di archiviazione di Amazon S3. Per maggiori informazioni sui prezzi di CloudTrail, consultare Prezzi di AWS CloudTrail. Per informazioni sui prezzi di Amazon S3, consulta Prezzi di Amazon S3.

Datastore di eventi CloudTrail Lake

Data Lake CloudTrail consente di eseguire query SQL sugli eventi. CloudTrail Lake converte gli eventi esistenti in formato JSON basato su righe in formato Apache ORC. ORC è un formato di archiviazione a colonne ottimizzato per il recupero rapido dei dati. Gli eventi vengono aggregati in archivi di dati degli eventi, che sono raccolte di eventi immutabili basate sui criteri selezionati applicando i selettori di eventi avanzati. I selettori applicati a un archivio di dati degli eventi controllano quali eventi persistono e sono disponibili per l'esecuzione della query. Per ulteriori informazioni su Data Lake CloudTrail, consulta Working with AWS CloudTrail Lake nella Guida per l'utente di AWS CloudTrail.

I datastore di eventi e le query di Data Lake CloudTrail comportano costi. Quando crei un datastore di eventi, scegli l'opzione di prezzo da utilizzare per tale datastore. L'opzione di prezzo determina il costo per l'importazione e l'archiviazione degli eventi, nonché il periodo di conservazione predefinito e quello massimo per il datastore di eventi. Per maggiori informazioni sui prezzi di CloudTrail, consultare Prezzi di AWS CloudTrail.

Eventi di dati Lambda in CloudTrail

Gli eventi di dati forniscono informazioni sulle operazioni delle risorse eseguite su o in una risorsa (ad esempio, lettura o scrittura su un oggetto Amazon S3). Queste operazioni sono definite anche operazioni del piano dei dati. Gli eventi di dati sono spesso attività che interessano volumi elevati di dati. Per impostazione predefinita, CloudTrail non registra la maggior parte degli eventi relativi ai dati e la cronologia degli eventi CloudTrail non li registra.

Un evento di dati CloudTrail che viene registrato per impostazione predefinita per i servizi supportati è `LambdaESMDisabled`. Per ulteriori informazioni sull'utilizzo di questo evento per risolvere i problemi relativi agli strumenti di mappatura dell'origine degli eventi Lambda, consulta the section called “Utilizzo di CloudTrail per risolvere i problemi relativi alle origini eventi Lambda disabilitate”.

Per gli eventi di dati sono previsti costi aggiuntivi. Per maggiori informazioni sui prezzi di CloudTrail, consultare Prezzi di AWS CloudTrail.

È possibile registrare gli eventi di dati per il tipo di risorse `AWS::Lambda::Function` utilizzando la console CloudTrail, AWS CLI o le operazioni dell'API CloudTrail. Per ulteriori informazioni su come registrare gli eventi di dati, consulta Registrazione degli eventi di dati con Console di gestione AWS e Registrazione degli eventi di dati con AWS Command Line Interface nella Guida per l'utente di AWS CloudTrail.

La tabella seguente elenca il tipo di risorse Lambda per i quali è possibile registrare gli eventi relativi ai dati. La colonna Tipo di evento di dati (console) mostra il valore da scegliere dall'elenco Tipo di evento di dati sulla console CloudTrail. La colonna resources.type value mostra il valore `resources.type`, da specificare quando si configurano selettori di eventi avanzati utilizzando le API AWS CLI o CloudTrail. La colonna Dati API registrati su CloudTrail mostra le chiamate API registrate su CloudTrail per il tipo di risorsa.

Tipo di evento di dati (console)	Valore resources.type	API di dati registrate su CloudTrail
Lambda	`AWS::Lambda::Function`	Invoke

È possibile configurare selettori di eventi avanzati per filtrare in base ai campi `eventName`, `readOnly`, e `resources.ARN` per registrare solo gli eventi che sono importanti per l'utente. L'esempio seguente è la vista JSON di una configurazione di eventi di dati che registra gli eventi solo per una funzione specifica. Per ulteriori informazioni su questi campi, consulta AdvancedFieldSelector nella Documentazione di riferimento delle API di AWS CloudTrail.

```
[
  {
    "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"
        ]
      }
    ]
  }
]
```

Eventi di gestione Lambda in CloudTrail

Gli eventi di gestione forniscono informazioni sulle operazioni di gestione eseguite sulle risorse nell'Account AWS. Queste operazioni sono definite anche operazioni del piano di controllo (control-plane). Per impostazione predefinita, CloudTrail registra gli eventi di gestione.

Lambda supporta la registrazione delle operazioni seguenti come eventi di gestione nei file di log CloudTrail.

> **Note**
>
> Nel file di log CloudTrail, `eventName` potrebbe includere informazioni sulla data e sulla versione, ma si riferisce comunque alla stessa API pubblica. Ad esempio, l'operazione `GetFunction` potrebbe apparire come `GetFunction20150331v2`. L'elenco seguente specifica quando il nome dell'evento è diverso dal nome dell'operazione API.

- AddLayerVersionPermission
- AddPermission (nome evento: `AddPermission20150331v2`)
- CreateAlias (nome evento: `CreateAlias20150331`)
- CreateEventSourceMapping (nome evento: `CreateEventSourceMapping20150331`)
- CreateFunction (nome evento: `CreateFunction20150331`)

  (I parametri `Environment` e `ZipFile` sono omessi dai log di CloudTrail per `CreateFunction`.)

- CreateFunctionUrlConfig
- DeleteAlias (nome evento: `DeleteAlias20150331`)
- DeleteCodeSigningConfig
- DeleteEventSourceMapping (nome evento: `DeleteEventSourceMapping20150331`)
- DeleteFunction (nome evento: `DeleteFunction20150331`)
- DeleteFunctionConcurrency (nome evento: `DeleteFunctionConcurrency20171031`)
- DeleteFunctionUrlConfig
- DeleteProvisionedConcurrencyConfig
- GetAlias (nome evento: `GetAlias20150331`)
- GetEventSourceMapping
- GetFunction
- GetFunctionUrlConfig
- GetFunctionConfiguration
- GetLayerVersionPolicy
- GetPolicy
- ListEventSourceMappings
- ListFunctions
- ListFunctionUrlConfigs
- PublishLayerVersion (nome evento: `PublishLayerVersion20181031`)

  (Il parametro `ZipFile` è omesso dai log di CloudTrail per `PublishLayerVersion`.)
- PublishVersion (nome evento: `PublishVersion20150331`)
- PutFunctionConcurrency (nome evento: `PutFunctionConcurrency20171031`)
- PutFunctionCodeSigningConfig
- PutFunctionEventInvokeConfig
- PutProvisionedConcurrencyConfig
- PutRuntimeManagementConfig
- RemovePermission (nome evento: `RemovePermission20150331v2`)
- TagResource (nome evento: `TagResource20170331v2`)
- UntagResource (nome evento: `UntagResource20170331v2`)
- UpdateAlias (nome evento: `UpdateAlias20150331`)

- UpdateCodeSigningConfig
- UpdateEventSourceMapping (nome evento: `UpdateEventSourceMapping20150331`)
- UpdateFunctionCode (nome evento: `UpdateFunctionCode20150331v2`)

  (Il parametro `ZipFile` è omesso dai log di CloudTrail per `UpdateFunctionCode`.)
- UpdateFunctionConfiguration (nome evento: `UpdateFunctionConfiguration20150331v2`)

  (Il parametro `Environment` è omesso dai log di CloudTrail per `UpdateFunctionConfiguration`.)
- UpdateFunctionEventInvokeConfig
- UpdateFunctionUrlConfig

Utilizzo di CloudTrail per risolvere i problemi relativi alle origini eventi Lambda disabilitate

Quando si modifica lo stato di uno strumento di mappatura dell'origine degli eventi utilizzando l'operazione API UpdateEventSourceMapping, la chiamata API viene registrata come evento di gestione in CloudTrail. Gli strumenti di mappatura dell'origine degli eventi possono anche passare direttamente allo stato `Disabled` a causa di errori.

Per i seguenti servizi, Lambda pubblica l'evento di dati `LambdaESMDisabled` su CloudTrail quando l'origine eventi passa allo stato Disabilitato:

- Amazon Simple Queue Service (Amazon SQS)
- Amazon DynamoDB
- Amazon Kinesis

Lambda non supporta questo evento per alcun altro tipo di strumento di mappatura dell'origine degli eventi.

Per ricevere avvisi quando gli strumenti di mappatura dell'origine degli eventi per i servizi supportati passano allo stato `Disabled`, configura un allarme in Amazon CloudWatch utilizzando l'evento CloudTrail `LambdaESMDisabled`. Per ulteriori informazioni sulla configurazione di un allarme CloudWatch, consulta Creazione di allarmi CloudWatch per eventi CloudTrail: esempi.

L'entità `serviceEventDetails` nel messaggio dell'evento `LambdaESMDisabled` contiene uno dei seguenti codici di errore.

RESOURCE_NOT_FOUND

La risorsa specificata nella richiesta non esiste.

FUNCTION_NOT_FOUND

La funzione associata all'origine eventi non esiste.

REGION_NAME_NOT_VALID

Il nome di una regione fornito all'origine o alla funzione evento non è valido.

AUTHORIZATION_ERROR

Le autorizzazioni non sono state impostate o non sono configurate correttamente.

FUNCTION_IN_FAILED_STATE

Il codice della funzione non viene compilato, ha rilevato un'eccezione irrecuperabile o si è verificata una distribuzione non valida.

Esempi di eventi Lambda

Un evento rappresenta una singola richiesta da qualsiasi fonte e include informazioni sull'operazione API richiesta, la data e l'ora dell'operazione, i parametri della richiesta e così via. I file di log di CloudTrail non sono uno stack trace ordinato delle chiamate API pubbliche, quindi gli eventi non appaiono in un ordine specifico.

L'esempio seguente mostra le voci di log CloudTrail per le operazioni `GetFunction` e `DeleteFunction`.

> **Note**
>
> La voce `eventName` potrebbe includere informazioni sulla data e sulla versione, ad esempio `"GetFunction20150331"`, ma si riferisce comunque alla stessa API pubblica.

```
{
  "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"
    }
  ]
}
```

Per informazioni sui contenuti delle registrazioni CloudTrail, consulta i contenuti delle registrazioni CloudTrail nella Guida per l'utente AWS CloudTrail.

Visualizza le chiamate alla funzione Lambda utilizzando AWS X-Ray

Puoi utilizzarli AWS X-Ray per visualizzare i componenti dell'applicazione, identificare i punti deboli delle prestazioni e risolvere le richieste che hanno provocato un errore. Le funzioni Lambda inviano dati di traccia a X-Ray e X-Ray elabora i dati per generare una mappa di servizio e riepiloghi di traccia ricercabili.

Lambda supporta due modalità di tracciamento per X-Ray: e. `Active` `PassThrough` Con il `Active` tracciamento, Lambda crea automaticamente segmenti di traccia per le chiamate di funzioni e li invia a X-Ray. `PassThrough`mode, d'altra parte, propaga semplicemente il contesto di tracciamento ai servizi a valle. Se hai abilitato il `Active` tracciamento per la tua funzione, Lambda invia automaticamente le tracce a X-Ray per le richieste campionate. In genere, un servizio upstream, come Amazon API Gateway o un'applicazione ospitata su Amazon EC2 dotata di strumentazione con l'SDK X-Ray, decide se tracciare le richieste in entrata, quindi aggiunge la decisione di campionamento come intestazione di tracciamento. Lambda utilizza quell'intestazione per decidere se inviare tracce o meno. Le tracce dei produttori di messaggi upstream, come Amazon SQS, vengono automaticamente collegate alle tracce delle funzioni Lambda downstream, creando end-to-end una visualizzazione dell'intera applicazione. Per ulteriori informazioni, consulta Tracciamento delle applicazioni guidate dagli eventi nella Guida per gli sviluppatori di AWS X-Ray .

> **Note**
>
> Il tracciamento X-Ray non è attualmente supportato per le funzioni Lambda con Streaming gestito da Amazon per Apache Kafka (Amazon MSK), Apache Kafka gestito dal cliente, Amazon MQ con ActiveMQ e RabbitMQ attivi oppure mappature delle origini degli eventi Amazon DocumentDB.

Per attivare il tracciamento attivo sulla funzione Lambda con la console, attenersi alla seguente procedura:

Per attivare il tracciamento attivo

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.

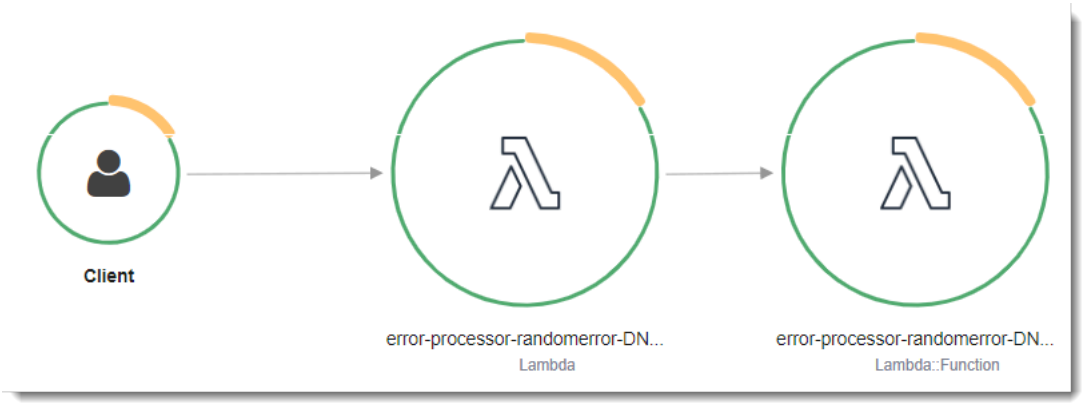
3. Scegliere Configuration (Configurazione) e quindi Monitoring and operations tools (Strumenti di monitoraggio e operazioni).
4. In Strumenti di monitoraggio aggiuntivi scegli Modifica.
5. In CloudWatch Application Signals e AWS X-Ray, scegli Enable for Lambda service trace.
6. Scegli Save (Salva).

La funzione ha bisogno dell'autorizzazione per caricare i dati di traccia su X-Ray. Quando si attiva il tracciamento nella console Lambda, Lambda aggiunge le autorizzazioni necessarie al ruolo di esecuzione della funzione. Altrimenti, aggiungi la AWSXRayDaemonWriteAccesspolicy al ruolo di esecuzione.

X-Ray non traccia tutte le richieste nell'applicazione. X-Ray applica un algoritmo di campionamento per garantire che il tracciamento avvenga in modo efficiente, continuando allo stesso tempo a fornire un campione rappresentativo di tutte le richieste. La frequenza di campionamento è di una richiesta al secondo e del 5% delle altre richieste. Non è possibile configurare la frequenza di campionamento di X-Ray per le funzioni.

Informazioni sui monitoraggi di X-Ray

In X-Ray, una traccia registra informazioni su una richiesta elaborata da uno o più servizi. Lambda registra 2 segmenti per traccia, che creano due nodi sul grafico del servizio. L'immagine seguente evidenzia questi due nodi:



Il primo nodo a sinistra rappresenta il servizio Lambda che riceve la richiesta di chiamata. Il secondo nodo rappresenta la specifica funzione Lambda.

Il segmento registrato per il servizio Lambda, `AWS::Lambda`, copre tutti i passaggi necessari per preparare l'ambiente di esecuzione Lambda. Ciò include la pianificazione di MicroVM, la creazione o

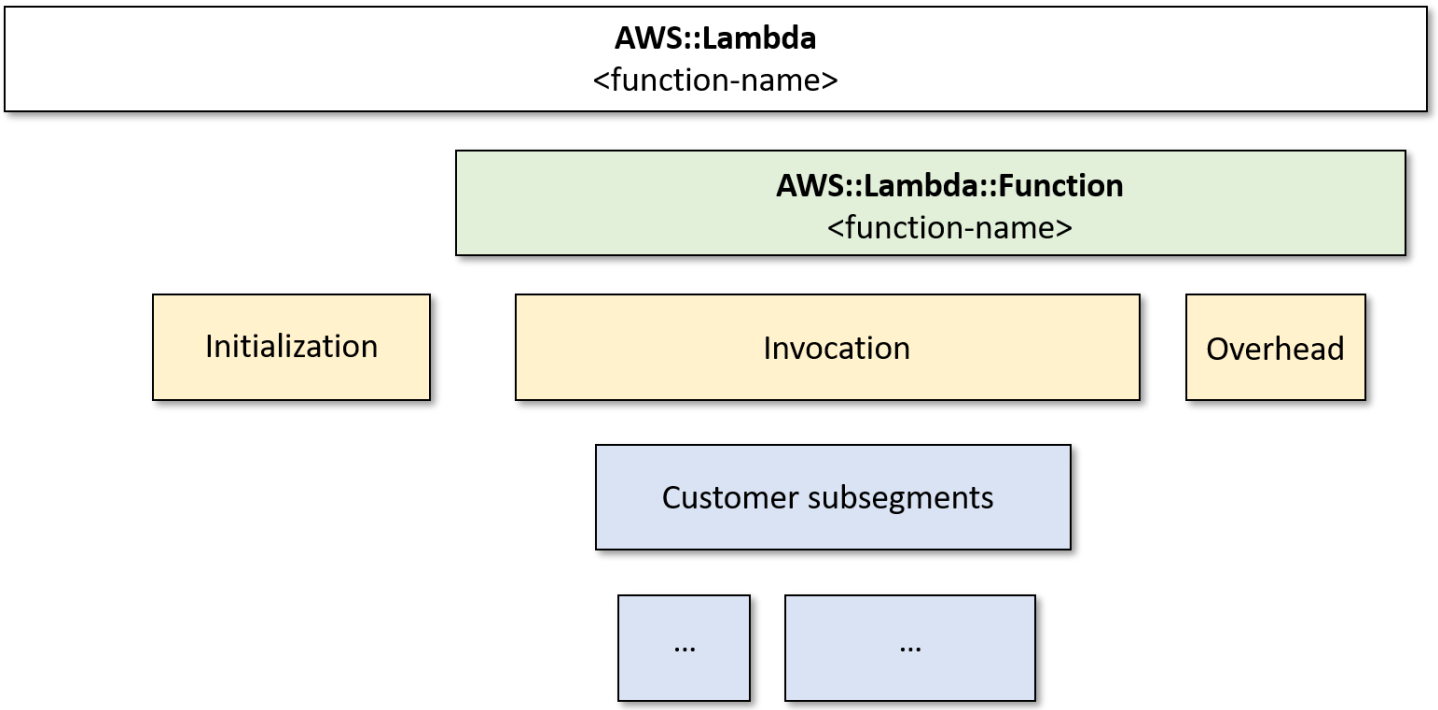
lo sblocco di un ambiente di esecuzione con le risorse configurate, nonché il download del codice per la funzione e di tutti i livelli.

Il segmento `AWS::Lambda::Function` è destinato al lavoro svolto dalla funzione.

> **Note**
>
> AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. Questa modifica influisce sui sottosegmenti del segmento della funzione. I paragrafi seguenti descrivono sia il vecchio che il nuovo formato per questi sottosegmenti.
> Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regioni, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia di nuovo formato.

Struttura a segmenti Lambda vecchio stile AWS X-Ray

La struttura X-Ray vecchio stile per il segmento `AWS::Lambda` è simile alla seguente:



In questo formato, il segmento della funzione ha sottosegmenti per `Initialization`, `Invocation` e `Overhead`. Solo per Lambda SnapStart, c'è anche un sottosegmento `Restore` (non mostrato in questo diagramma).

Il sottosegmento `Initialization` rappresenta la fase iniziale del ciclo di vita dell'ambiente di esecuzione Lambda. In questa fase, Lambda inizializza le estensioni, inizializza il runtime ed esegue il codice di inizializzazione della funzione.

Il sottosegmento `Invocation` rappresenta la fase di invocazione in cui Lambda richiama il gestore di funzioni. Questo inizia con la registrazione del runtime e dell'estensione e termina quando il runtime è pronto per inviare la risposta.

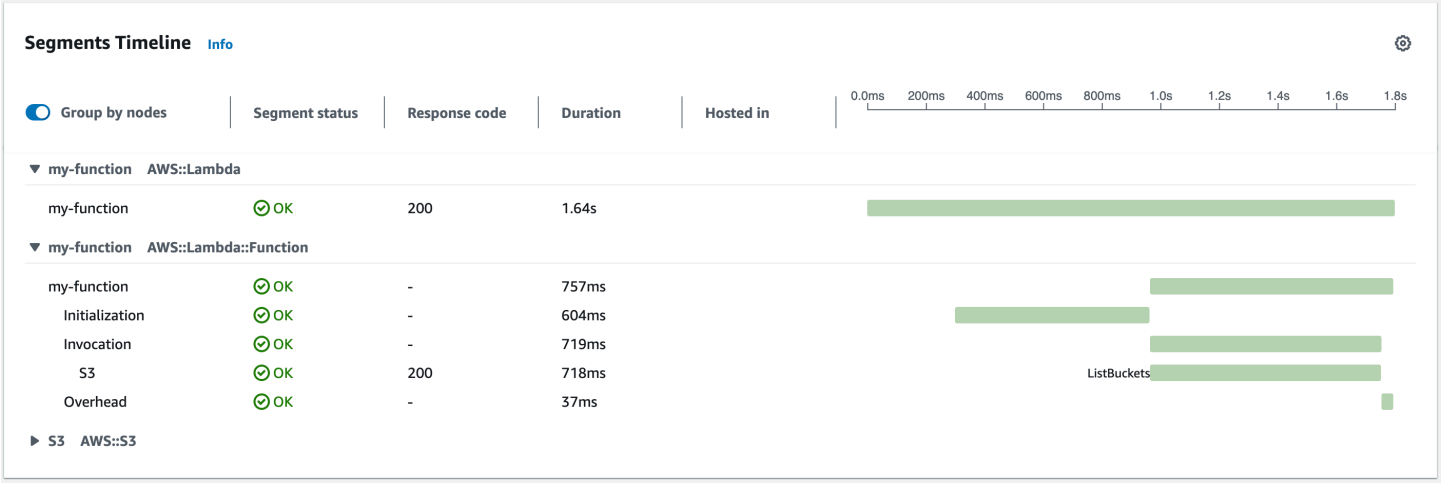
(SnapStart Solo Lambda) Il `Restore` sottosegmento mostra il tempo impiegato da Lambda per ripristinare un'istantanea, caricare il runtime ed eseguire eventuali hook di runtime successivi al ripristino. Il processo di ripristino degli snapshot può includere il tempo dedicato ad attività esterne alla MicroVM. Questa volta è riportato nel segmento secondario `Restore`. Non ti viene addebitato il tempo trascorso fuori dalla microVM per il ripristino di una snapshot.

Il sottosegmento `Overhead` rappresenta la fase che si verifica tra il momento in cui il runtime invia la risposta e il segnale per la successiva invocazione. Durante questo periodo, il runtime termina tutte le attività correlate a un'invocazione e si prepara a congelare la sandbox.

> **Important**
>
> È possibile utilizzare l'SDK X-Ray per estendere il sottosegmento `Invocation` con sottosegmenti aggiuntivi per chiamate a valle, annotazioni e metadati. Non è possibile accedere direttamente al segmento di funzione o registrare la parola eseguita al di fuori dell'ambito di chiamata del gestore.

Per ulteriori informazioni sulle fasi dell'ambiente di esecuzione Lambda, consulta the section called “Ambiente di esecuzione”.

Nel diagramma seguente è illustrato un esempio di traccia che utilizza la struttura di X-Ray vecchio stile.

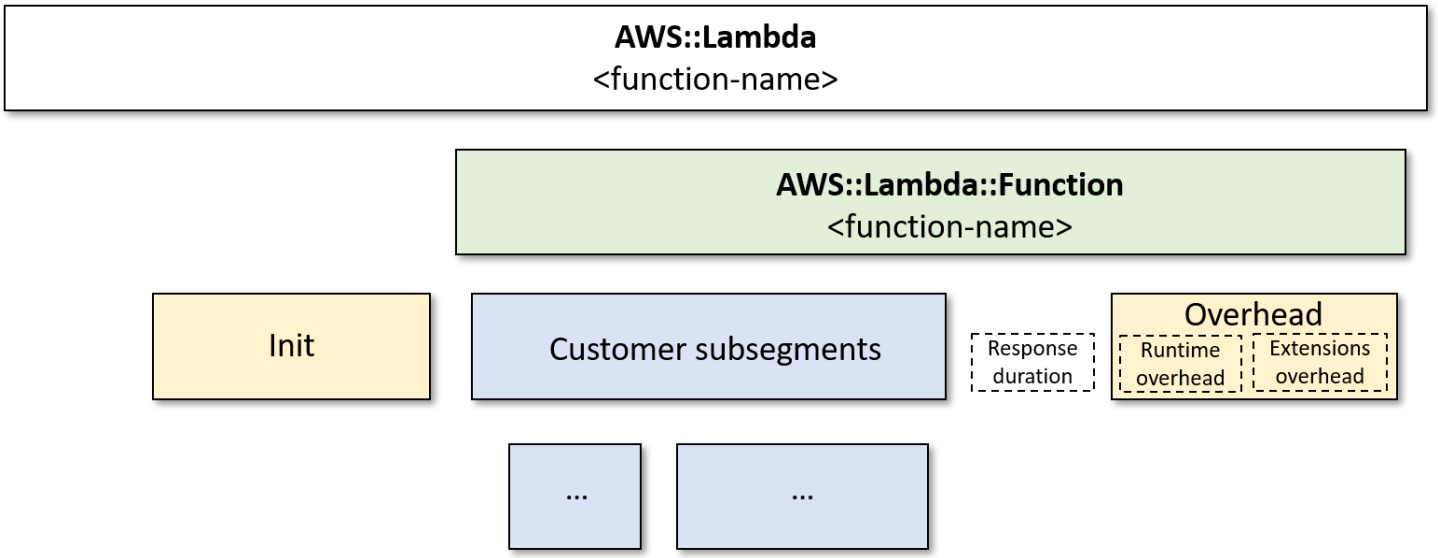


Nota i due segmenti dell'esempio. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.

> **Note**
>
> Occasionalmente, potresti notare un ampio divario tra le fasi di inizializzazione e invocazione della funzione nelle tracce X-Ray. Per le funzioni che utilizzano la simultaneità fornita, ciò è dovuto al fatto che Lambda inizializza le istanze della funzione con largo anticipo rispetto all'invocazione. Per le funzioni che utilizzano la concorrenza non riservata (su richiesta), Lambda può inizializzare in modo proattivo un'istanza di funzione, anche in assenza di chiamata. Visivamente, entrambi questi casi si presentano come un intervallo di tempo tra le fasi di inizializzazione e invocazione.

Struttura dei segmenti Lambda di nuova AWS X-Ray concezione

La struttura X-Ray nuovo stile per il segmento `AWS::Lambda` è simile alla seguente:

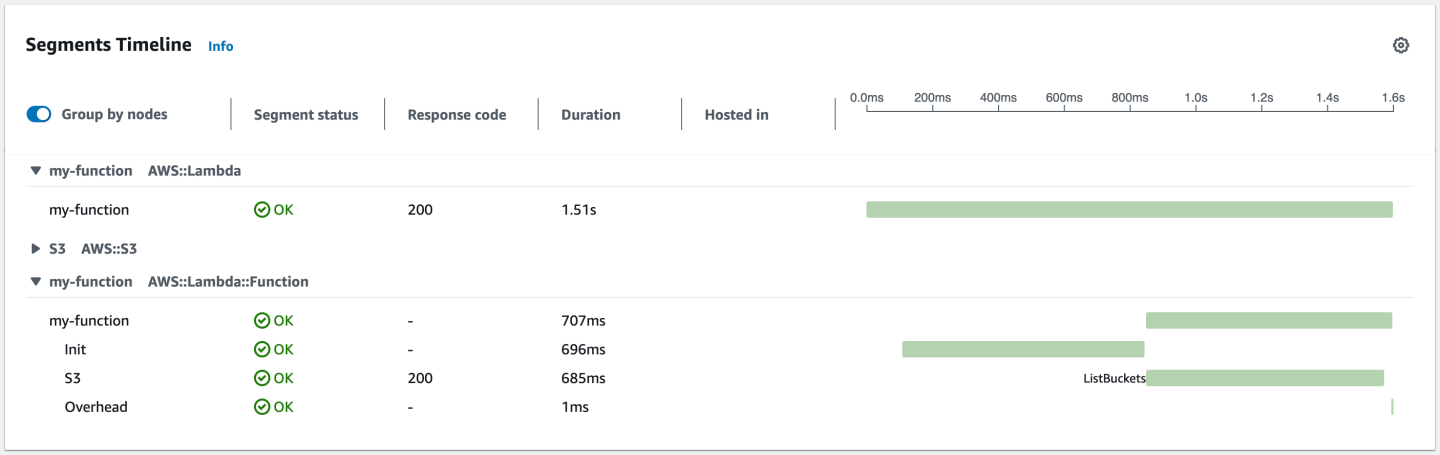


In questo nuovo formato, il sottosegmento `Init` rappresenta la fase iniziale del ciclo di vita dell'ambiente di esecuzione Lambda.

Nel nuovo formato non è presente alcun segmento di invocazione. I sottosegmenti dei clienti sono invece collegati direttamente al segmento `AWS::Lambda::Function`. Questo segmento contiene i seguenti parametri come annotazioni:

- `aws.responseLatency`: il tempo impiegato per l'esecuzione della funzione
- `aws.responseDuration`: il tempo impiegato per trasferire la risposta al cliente
- `aws.runtimeOverhead`: la quantità di tempo supplementare necessaria al completamento del runtime
- `aws.extensionOverhead`: la quantità di tempo supplementare necessaria al completamento delle estensioni

Nel diagramma seguente è illustrato un esempio di traccia che utilizza la struttura di X-Ray nuovo stile.



Nota i due segmenti dell'esempio. Entrambi sono nominati my-function, ma uno ha l'origine `AWS::Lambda` e l'altro ha l'origine `AWS::Lambda::Function`. Se il segmento `AWS::Lambda` mostra un errore, il servizio Lambda ha avuto un problema. Se il `AWS::Lambda::Function` segmento mostra un errore, la funzione ha avuto un problema.

Vedere i seguenti argomenti per un'introduzione specifica della lingua all'analisi in Lambda:

- Strumentazione del codice Node.js in AWS Lambda
- Strumentazione del codice Python in AWS Lambda
- Strumentazione del codice Ruby in AWS Lambda
- Strumentazione del codice Java in AWS Lambda
- Strumentazione del codice Go in AWS Lambda
- Strumentazione del codice C# in AWS Lambda

Per un elenco completo dei servizi che supportano la strumentazione attiva, consulta Servizi Servizi AWS supportati nella Guida per gli sviluppatori di AWS X-Ray .

Comportamento di tracciamento predefinito in Lambda

Se la funzione di `Active` tracciamento non è attivata, Lambda utilizza come impostazione predefinita `PassThrough` la modalità di tracciamento.

In `PassThrough` modalità, Lambda inoltra l'intestazione di tracciamento X-Ray ai servizi a valle, ma non invia le tracce automaticamente. Questo è vero anche se l'intestazione di tracciamento contiene la decisione di campionare la richiesta. Se il servizio upstream non fornisce un'intestazione

di tracciamento a raggi X, Lambda genera un'intestazione e decide di non campionare. Tuttavia, puoi inviare le tue tracce richiamando le librerie di tracciamento dal codice della funzione.

> **Note**
>
> In precedenza, Lambda inviava le tracce automaticamente quando i servizi upstream, come Amazon API Gateway, aggiungevano un'intestazione di tracciamento. Non inviando tracce automaticamente, Lambda ti offre il controllo necessario per tracciare le funzioni che ritieni importanti. Se la tua soluzione dipende da questo comportamento di tracciamento passivo, passa al `Active` tracciamento.

Autorizzazioni del ruolo di esecuzione

Lambda richiede le seguenti autorizzazioni per inviare i dati di traccia a X-Ray. Aggiungerle al ruolo di esecuzione della funzione.

- Radiografia: PutTraceSegments
- radiografia: PutTelemetryRecords

Queste autorizzazioni sono incluse nella politica AWSXRayDaemonWriteAccessgestita.

Abilitazione del **`Active`** tracciamento con l'API Lambda

Per gestire la configurazione di tracciamento con AWS CLI o AWS SDK, utilizza le seguenti operazioni API:

- UpdateFunctionConfiguration
- GetFunctionConfiguration
- CreateFunction

Il AWS CLI comando di esempio seguente abilita il tracciamento attivo su una funzione denominata my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

La modalità di tracciamento fa parte della configurazione specifica della versione quando si pubblica una versione della funzione. Non è possibile modificare la modalità di tracciamento in una versione pubblicata.

Abilitazione del tracciamento con **`Active`**CloudFormation

Per attivare il tracciamento su una `AWS::Lambda::Function` risorsa in un CloudFormation modello, utilizzate la `TracingConfig` proprietà.

Example function-inline.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
```

Per una `AWS::Serverless::Function` risorsa AWS Serverless Application Model (AWS SAM), utilizzate la `Tracing` proprietà.

Example template.yml – Configurazione del tracciamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
```

Monitorare le prestazioni delle funzioni con Lambda Insights di Amazon CloudWatch

Amazon CloudWatch Lambda Insights raccoglie e aggrega parametri e registri delle prestazioni del runtime delle funzioni Lambda per le applicazioni serverless. In questa pagina viene descritto come abilitare e utilizzare Lambda Insights per eseguire la diagnosi dei problemi relativi alle funzioni Lambda.

Sezioni

- Come Lambda Insights monitora le applicazioni serverless
- Prezzi
- Runtime supportati
- Attivazione di Lambda Insights nella console Lambda
- Attivazione di Lambda Insights a livello di programmazione
- Uso del pannello di controllo di Lambda Insights
- Esempio di flusso di lavoro per rilevare anomalie delle funzioni
- Esempio di flusso di lavoro utilizzando query per risolvere i problemi di una funzione
- Fasi successive

Come Lambda Insights monitora le applicazioni serverless

CloudWatch Lambda Insights è una soluzione di monitoraggio e risoluzione dei problemi per applicazioni serverless in esecuzione su AWS Lambda. La soluzione raccoglie, aggrega e riassume le metriche a livello di sistema, tra cui il tempo della CPU, la memoria, l'utilizzo del disco e della rete. Vengono inoltre raccolte, aggregate e riepilogate informazioni diagnostiche quali avvii a freddo e arresti del worker Lambda per aiutare a isolare i problemi con le funzioni Lambda e risolverli rapidamente.

Lambda Insights utilizza una nuova estensione CloudWatch Lambda Insights che viene fornita come livello Lambda. Quando si abilita questa estensione su una funzione Lambda per un runtime supportato, essa raccoglie parametri a livello di sistema ed emette un singolo evento di log delle prestazioni per ogni chiamata di tale funzione Lambda. CloudWatch utilizza la formattazione dei parametri incorporata per estrarre i parametri dagli eventi di log. Per ulteriori informazioni, consulta l'argomento relativo all'Uso delle estensioni AWS Lambda.

Il livello Lambda Insights estende `CreateLogStream` e `PutLogEvents` per il gruppo di log `/aws/lambda-insights/`.

Prezzi

Quando so abilita Lambda Insights per la funzione Lambda, Lambda Insights riporta 8 parametri per funzione e ogni chiamata di funzione invia circa 1 KB di dati di log a CloudWatch. Paghi solo in base ai parametri e ai registri riportati per la tua funzione da Lambda Insights. Non sono previste tariffe minime né policy di utilizzo del servizio obbligatorie. Non si paga per Lambda Insights se la funzione non viene richiamata. Per informazioni sui prezzi, consulta la pagina Prezzi di Amazon CloudWatch.

Runtime supportati

È possibile utilizzare Lambda Insights con qualsiasi runtime che supporta le estensioni Lambda.

Attivazione di Lambda Insights nella console Lambda

È possibile abilitare il monitoraggio Lambda Insights avanzato su funzioni Lambda nuove ed esistenti. Quando si abilita Lambda Insights su una funzione nella console Lambda per un runtime supportato, Lambda aggiunge l'estensione Lambda Insights come livello alla funzione e verifica o prova ad allegare la policy `CloudWatchLambdaInsightsExecutionRolePolicy` al ruolo di esecuzione della funzione.

Per attivare Lambda Insights nella console Lambda

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere la funzione.
3. Scegli la scheda Configurazione.
4. Dal menu a sinistra, scegli Strumenti di monitoraggio e operazioni.
5. Nel riquadro Strumenti di monitoraggio aggiuntivi scegli Modifica.
6. In CloudWatch Lambda Insights, attiva Monitoraggio avanzato.
7. Selezionare Salva.

Attivazione di Lambda Insights a livello di programmazione

È inoltre possibile abilitare Lambda Insights utilizzando la AWS Command Line Interface (AWS CLI), la CLI (SAM) AWS Serverless Application Model, CloudFormation o il AWS

Cloud Development Kit (AWS CDK). Quando si abilita Lambda Insights a livello di programmazione su una funzione per un runtime supportato, CloudWatch collega la policy `CloudWatchLambdaInsightsExecutionRolePolicy` al ruolo di esecuzione della funzione.

Per ulteriori informazioni, consulta le Nozioni di base su Lambda Insights nella Guida per l'utente di Amazon CloudWatch.

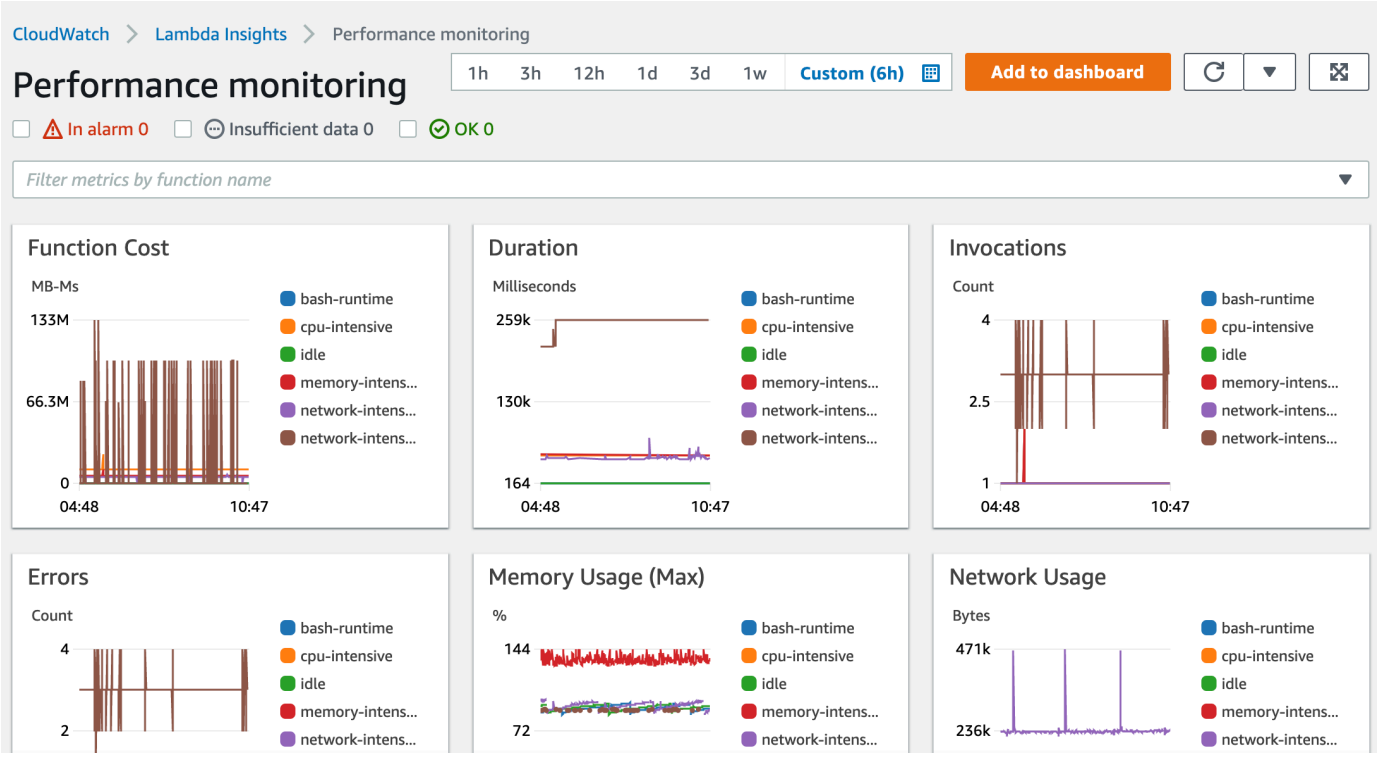
Uso del pannello di controllo di Lambda Insights

Il pannello di controllo Lambda Insights ha due visualizzazioni nella console CloudWatch: la panoramica multifunzione e la vista a funzione singola. La panoramica multi-funzione aggrega i parametri di runtime per le funzioni Lambda nell'account AWS attuale e nella regione. La vista a funzione singola mostra i parametri di runtime disponibili per una singola funzione Lambda.

È possibile utilizzare la panoramica multifunzione del pannello di controllo Lambda Insights nella console CloudWatch per identificare le funzioni Lambda sovrautilizzate e sottoutilizzate. È possibile utilizzare la visualizzazione a funzione singola del pannello di controllo Lambda Insights nella console CloudWatch per risolvere i problemi delle singole richieste.

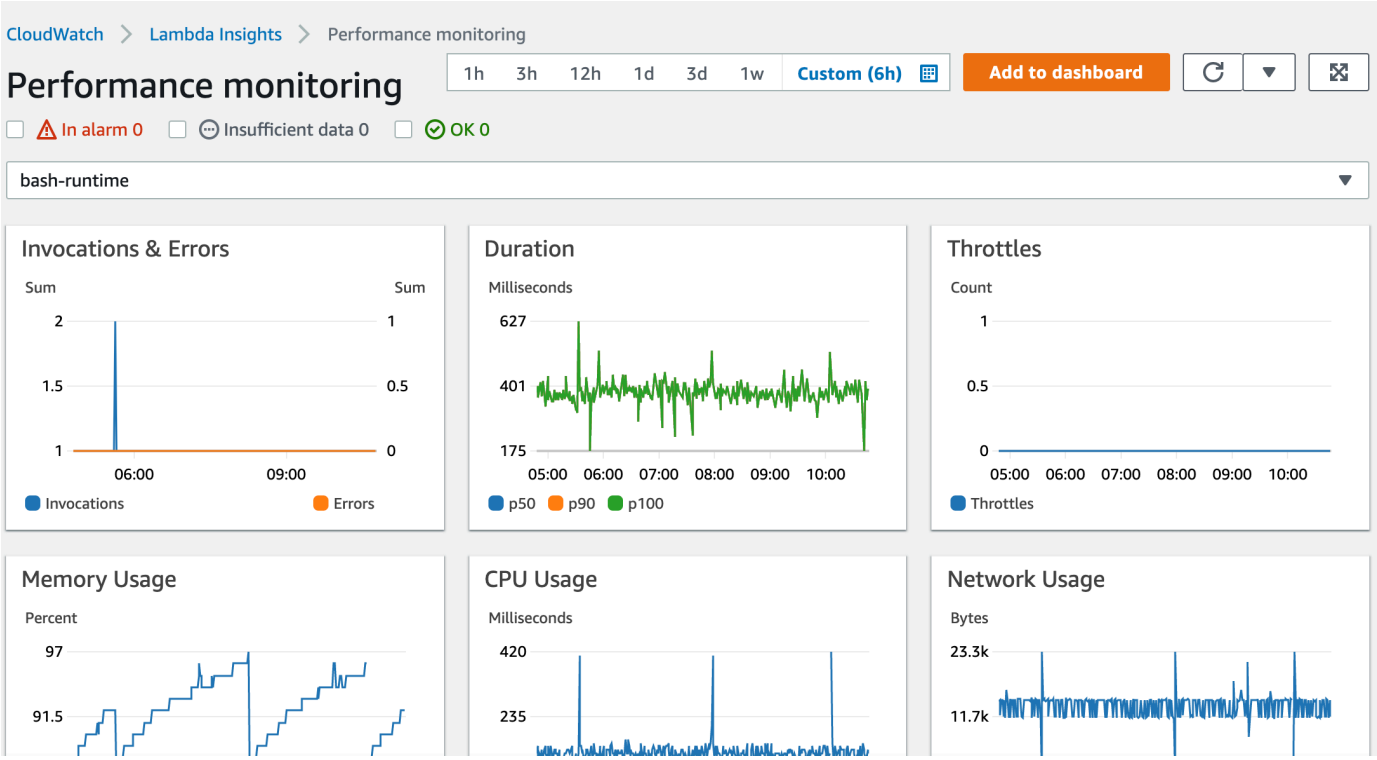
Per visualizzare i parametri di runtime per tutte le funzioni

1. Aprire la pagina Multifunzione nella console CloudWatch.
2. Scegli tra gli intervalli di tempo predefiniti oppure scegli un intervallo di tempo personalizzato.
3. (Facoltativo) Scegliere Add to dashboard (Aggiungi a pannello di controllo) per aggiungere i widget al dashboard CloudWatch.



Per visualizzare le metriche di runtime di una singola funzione

1. Aprire la pagina Funzione singola nella console CloudWatch.
2. Scegli tra gli intervalli di tempo predefiniti oppure scegli un intervallo di tempo personalizzato.
3. (Facoltativo) Scegliere Add to dashboard (Aggiungi a pannello di controllo) per aggiungere i widget al dashboard CloudWatch.



Per ulteriori informazioni, vedere Creazione e utilizzo dei widget nei pannelli di controllo CloudWatch.

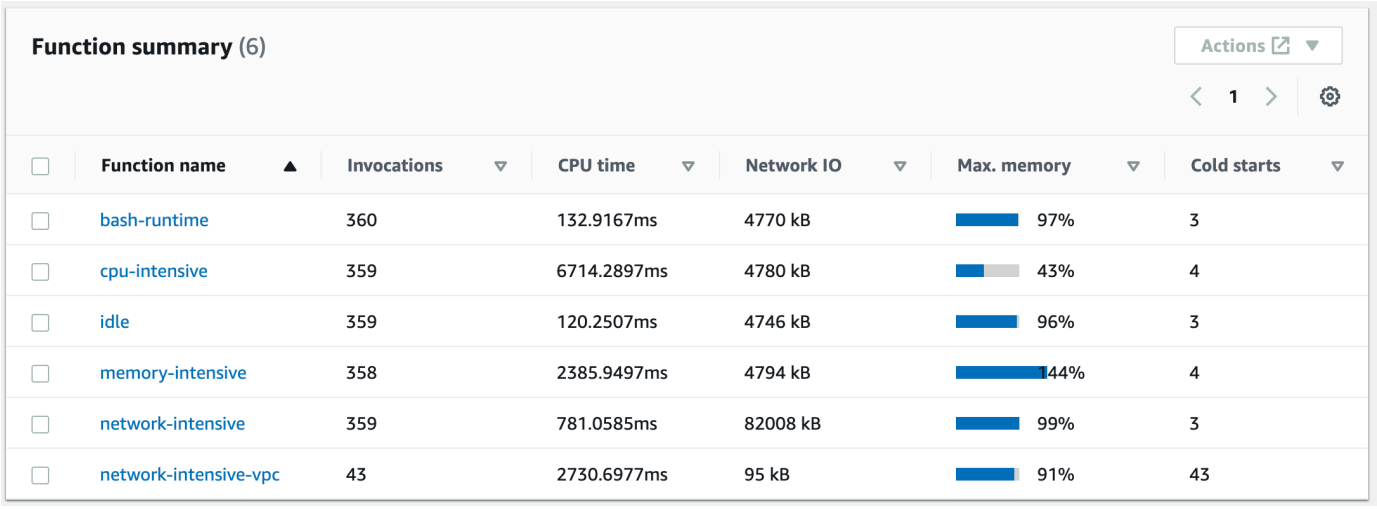
Esempio di flusso di lavoro per rilevare anomalie delle funzioni

È possibile utilizzare la panoramica multifunzione sul pannello di controllo Lambda Insights per identificare e rilevare anomalie della memoria di calcolo con la funzione. Ad esempio, se la panoramica multifunzione indica che una funzione utilizza una grande quantità di memoria, è possibile visualizzare metriche dettagliate sull'utilizzo della memoria nel riquadro Uso memoria. Puoi quindi accedere al pannello di controllo Parametri per abilitare il rilevamento delle anomalie o creare un allarme.

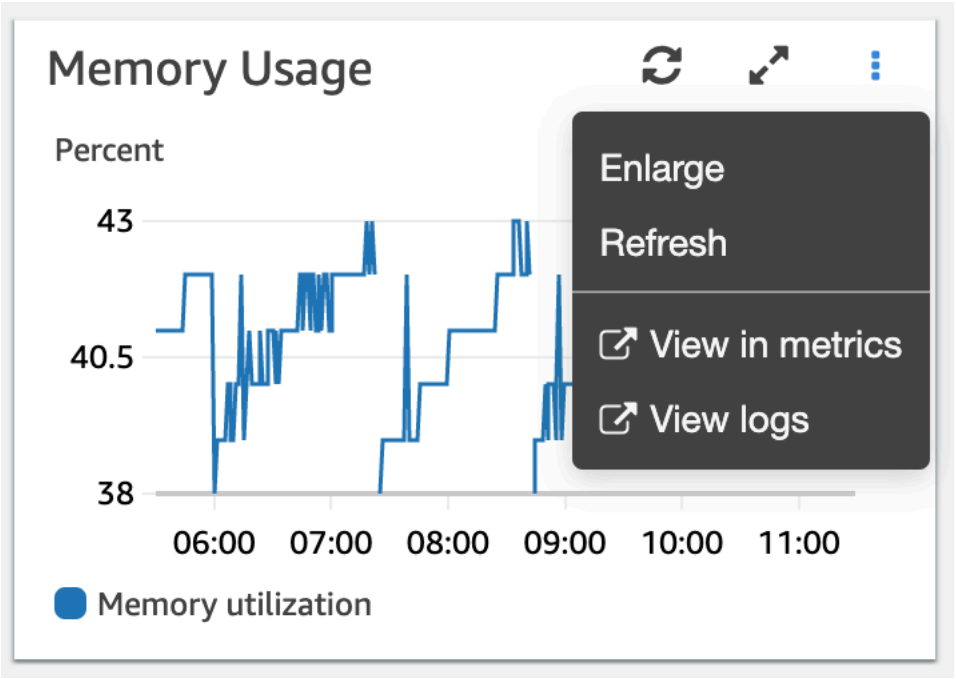
Per abilitare il rilevamento delle anomalie per una funzione

1. Aprire la pagina Multifunzione nella console CloudWatch.
2. In Riepilogo funzione, scegliere il nome della funzione.

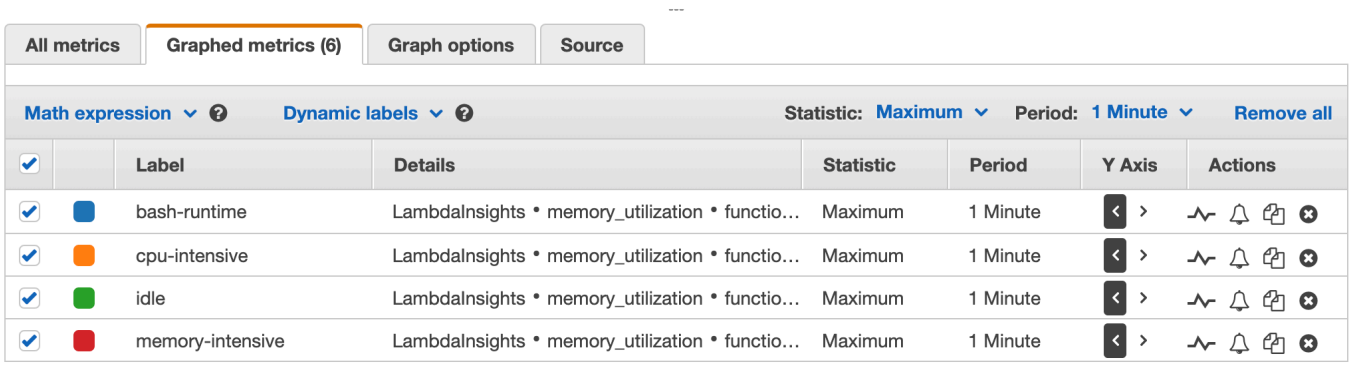
   La vista a funzione singola si apre con le metriche di runtime della funzione.



3. Nel riquadro Uso memoria scegliere i tre punti verticali, quindi scegliere Visualizza in metriche per aprire il pannello di controllo Parametri .



4. Nella scheda Graphed metrics (Parametri grafici), nella colonna Actions (Operazioni), scegli la prima icona per abilitare il rilevamento delle anomalie per la funzione.



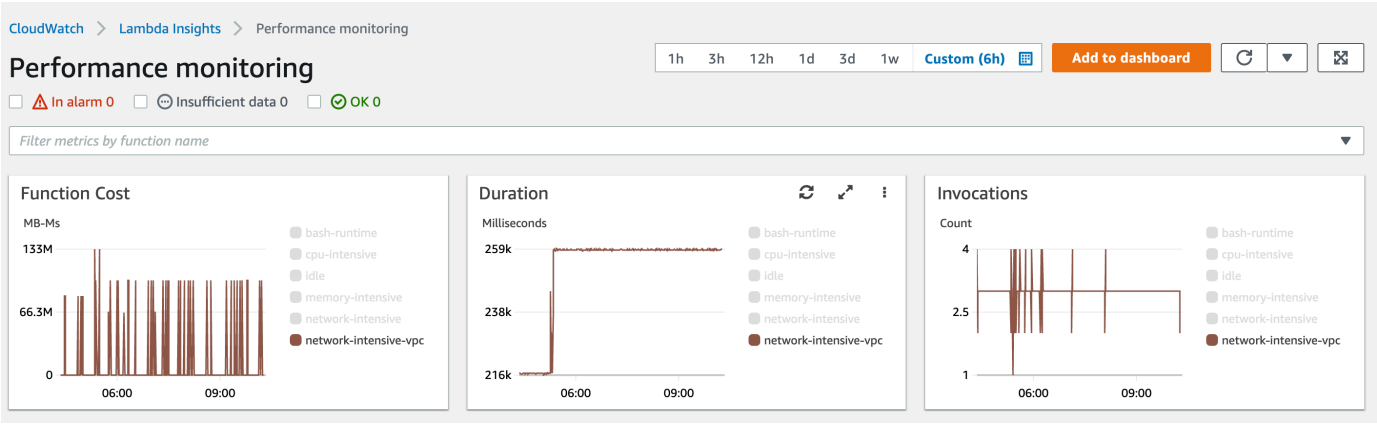
Per ulteriori informazioni, consulta Utilizzo del rilevamento delle anomalie di CloudWatch.

Esempio di flusso di lavoro utilizzando query per risolvere i problemi di una funzione

È possibile utilizzare la visualizzazione a funzione singola nel pannello di controllo Lambda Insights per identificare la causa principale di un picco nella durata della funzione. Ad esempio, se la panoramica multifunzione indica un notevole aumento della durata della funzione, è possibile sospendere o scegliere ciascuna funzione nel riquadro Durata per determinare quale funzione sta causando l'aumento. È quindi possibile accedere alla visualizzazione a funzione singola ed esaminare i registri applicazioni per determinare la causa principale.

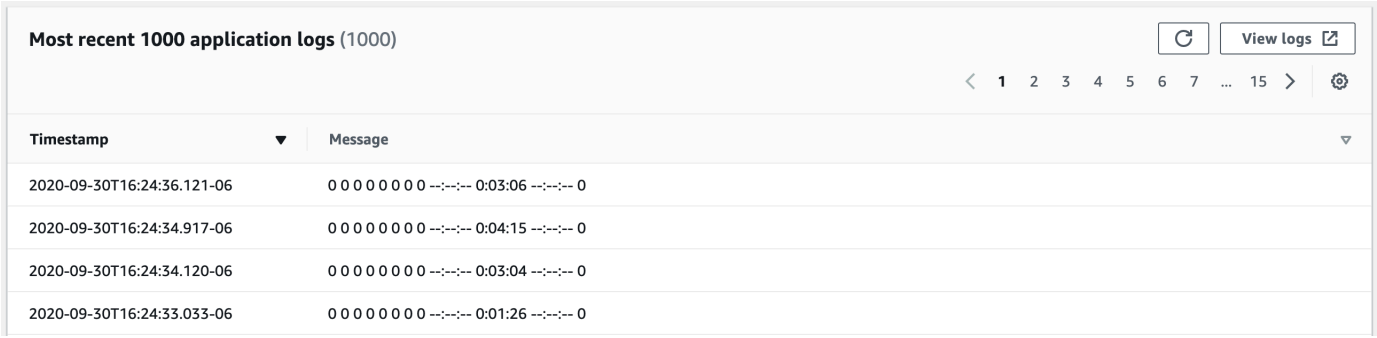
Per eseguire query su una funzione

1. Aprire la pagina Multifunzione nella console CloudWatch.
2. Nel riquadro Durata scegliere la funzione per filtrare le metriche della durata.

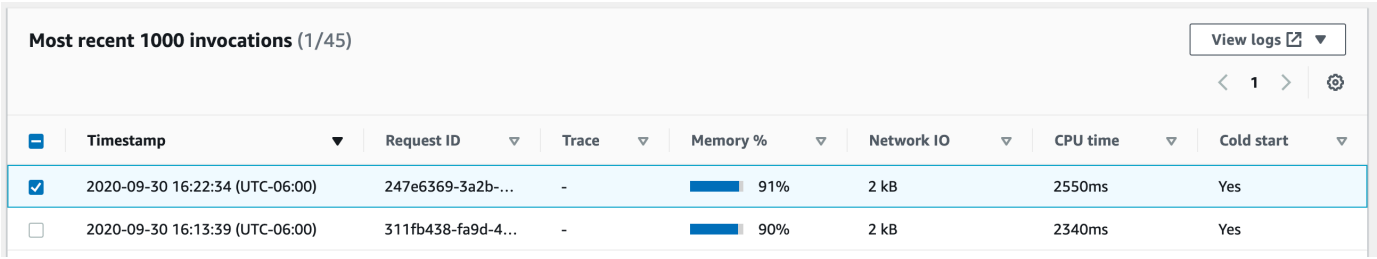


3. Aprire la pagina Funzione singola.
4. Scegli l'elenco a discesa Filtra metriche per nome funzione, quindi scegli la funzione.

5. Per visualizzare i 1000 registri applicazioni più recenti, scegliere la scheda Registri applicazioni.
6. Esaminare il timestamp e il messaggio per identificare la richiesta di chiamata per cui si desidera risolvere i problemi.

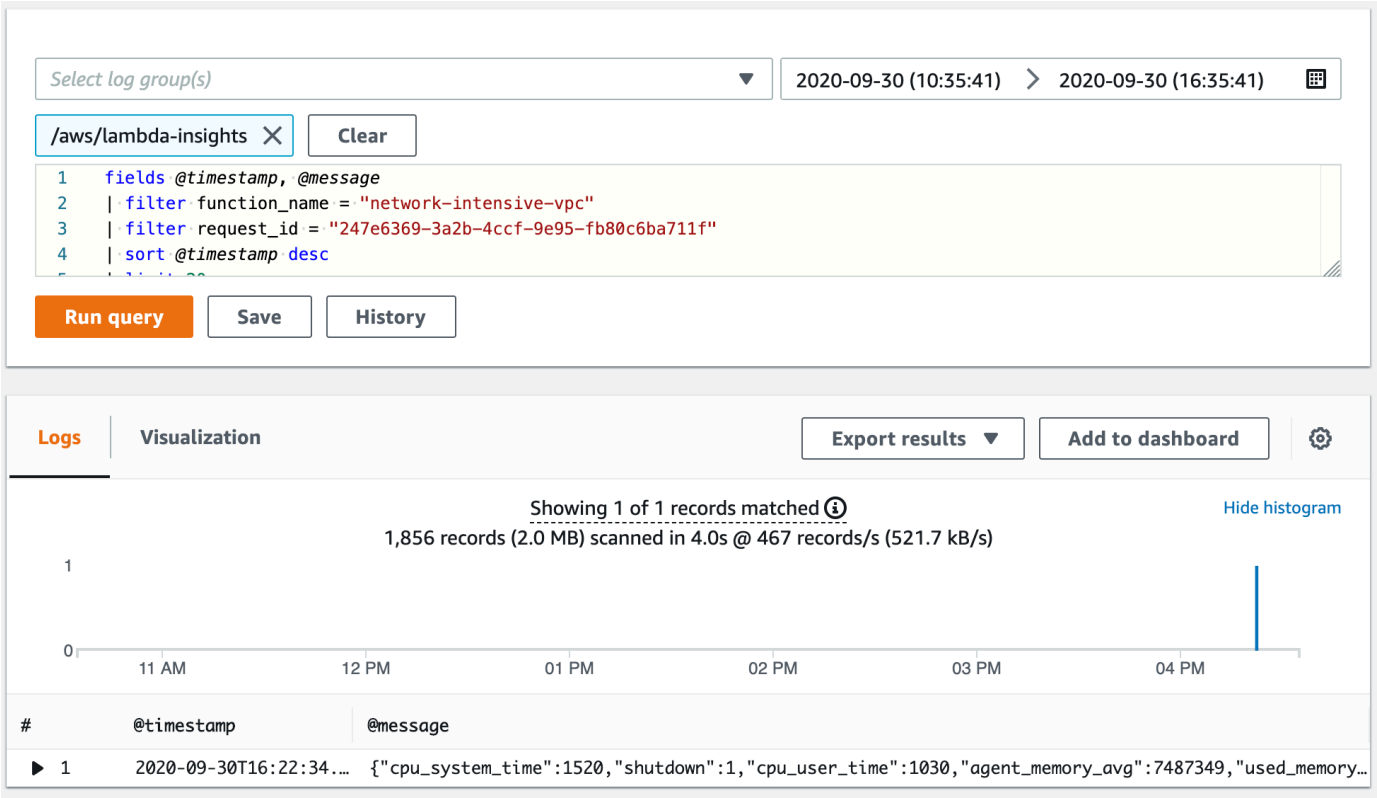


7. Per visualizzare le 1000 invocazioni più recenti, scegliere la scheda Invocazioni .
8. Selezionare il timestamp o il messaggio per la richiesta di invocazione per cui si desidera risolvere i problemi.



9. Scegliere l'elenco a discesa Visualizza registri quindi scegliere Visualizza registri prestazioni.

   Viene visualizzata una query generata automaticamente per la funzione nel pannello di controllo Logs Insights (Analisi registri) .
10. Scegliere Esegui query per generare un messaggio Log per la richiesta di chiamata.



Fasi successive

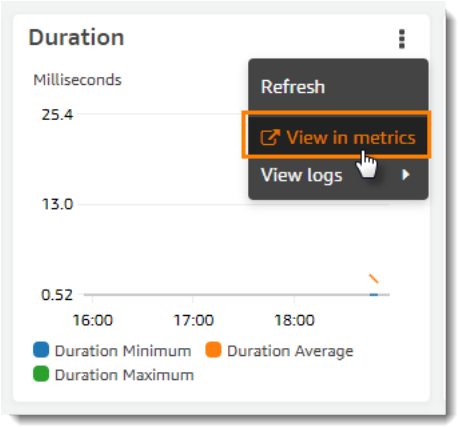
- Informazioni su come creare un pannello di controllo di CloudWatch Logs in Creazione di un pannello di controllo nella Guida per l'utente di Amazon CloudWatch.
- Scopri come aggiungere query a un pannello di controllo di CloudWatch Logs in Aggiunta di query a pannello di controllo o esportazione dei risultati della query nella Guida per l'utente di Amazon CloudWatch.

Monitoraggio delle applicazioni Lambda

La sezione Applicazioni della console Lambda include una scheda Monitoraggio in cui è possibile esaminare un pannello di controllo di Amazon CloudWatch con parametri aggregati per le risorse dell'applicazione.

Per monitorare un'applicazione Lambda

1. Aprire la pagina Applicazioni della console Lambda.
2. Selezionare Monitoring (Monitoraggio).
3. Per maggiori dettagli sui parametri in qualsiasi grafico, scegli Visualizza nei parametri dal menu a discesa.



   Il grafico viene visualizzato in una nuova scheda, con i parametri pertinenti elencati sotto il grafico. Puoi personalizzare la visualizzazione di questo grafico, modificando i parametri e le risorse mostrate, le statistiche, il periodo e altri fattori per comprendere meglio la situazione attuale.

Per impostazione predefinita, la console Lambda mostra un pannello di controllo di base. Puoi personalizzare questa pagina aggiungendo uno o più pannelli di controllo Amazon CloudWatch al modello dell'applicazione con il tipo di risorsa AWS::CloudWatch::Dashboard. Quando il modello include uno o più pannelli di controllo, la pagina mostra i pannelli di controllo anziché il pannello di controllo predefinito. Puoi passare da un pannello di controllo all'altro con il menu a discesa in alto a destra della pagina. Nell'esempio seguente viene creato un pannello di controllo con un singolo widget che rappresenta graficamente il numero di chiamate di una funzione denominata `my-function`.

Example Modello di pannello di controllo delle funzioni

```
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
                    }
                }
            ]
        }
```

Per ulteriori informazioni sulla creazione di pannelli di controllo e widget di CloudWatch, consulta Struttura e sintassi del corpo del pannello di controllo nella Documentazione di riferimento delle API di Amazon CloudWatch.

Monitoraggio delle prestazioni delle applicazioni con Amazon CloudWatch Application Signals

Amazon CloudWatch Application Signals è una soluzione di monitoraggio delle prestazioni delle applicazioni (APM) che consente a sviluppatori e operatori di monitorare lo stato e le prestazioni delle loro applicazioni serverless create con Lambda. Puoi abilitare Application Signals con un clic dalla console Lambda e non è necessario aggiungere alcun codice di strumentazione o dipendenze esterne alla funzione Lambda. Dopo aver abilitato Application Signals, è possibile visualizzare tutti i parametri e le tracce raccolte nella console CloudWatch. In questa pagina viene descritto come abilitare e visualizzare i dati di telemetria di Application Signals per le applicazioni.

Argomenti

- In che modo Application Signals si integra con Lambda
- Prezzi
- Runtime supportati
- Abilitazione di Application Signals nella console Lambda
- Utilizzo del pannello di controllo di Application Signals

In che modo Application Signals si integra con Lambda

Application Signals strumenta automaticamente le funzioni Lambda utilizzando le librerie AWS Distro for OpenTelemetry (ADOT) avanzate, fornite tramite un livello Lambda. Application Signals legge i dati raccolti dal livello e genera pannelli di controllo con parametri delle prestazioni chiave per le applicazioni.

Puoi collegare questo livello con un clic abilitando Application Signals nella console Lambda. Quando attivi Application Signals dalla console, Lambda esegue le seguenti operazioni per conto tuo:

- Aggiorna il ruolo di esecuzione della funzione per includere `CloudWatchLambdaApplicationSignalsExecutionRolePolicy`. Questa policy fornisce l'accesso in scrittura a AWS X-Ray e ai gruppi di log CloudWatch utilizzati per Application Signals.
- Aggiunge un livello alla funzione che la strumenta automaticamente per acquisire dati di telemetria come richieste, disponibilità, latenza, errori e guasti. Per garantire che Application Signals funzioni correttamente, rimuovi qualsiasi codice di strumentazione dell'SDK X-Ray esistente dalla tua funzione. Il codice di strumentazione dell'SDK X-Ray personalizzato può interferire con la strumentazione fornita dal livello.

- Aggiunge la variabile di ambiente `AWS_LAMBDA_EXEC_WRAPPER` alla funzione e ne imposta il valore su `/opt/otel-instrument`. Questa variabile di ambiente modifica il comportamento di avvio della funzione per utilizzare il livello di Application Signals ed è necessaria per una corretta strumentazione. Se questa variabile di ambiente esiste già, assicurati che sia impostata sul valore richiesto.

Prezzi

L'utilizzo di Application Signals per le funzioni Lambda comporta dei costi. Per informazioni sui prezzi, consulta la pagina Prezzi di Amazon CloudWatch.

Runtime supportati

L'integrazione di Application Signals con Lambda funziona con i seguenti runtime:

- .NET 8
- Java 11
- Java 17
- Java 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

Abilitazione di Application Signals nella console Lambda

Puoi abilitare Application Signals su qualsiasi funzione Lambda esistente utilizzando un runtime supportato. La procedura seguente descrive come abilitare Application Signals nella console Lambda.

Per abilitare Application Signals nella console Lambda

1. Aprire la pagina Funzioni della console Lambda.

2. Scegliere la funzione.
3. Scegli la scheda Configurazione.
4. Dal menu a sinistra, scegli Strumenti di monitoraggio e operazioni.
5. Nel riquadro Strumenti di monitoraggio aggiuntivi scegli Modifica.
6. In CloudWatch Application Signals e AWS X-Ray e in Application Signals, seleziona Abilita.
7. Selezionare Salva.

Se è la prima volta che abiliti Application Signals per la tua funzione, devi anche configurare una sola volta il rilevamento servizi per Application Signals nella console CloudWatch. Dopo aver completato questa configurazione una tantum del rilevamento servizi, Application Signals rileva automaticamente tutte le funzioni Lambda aggiuntive per le quali abiliti Application Signals, in tutte le regioni.

> **Note**
>
> Dopo aver richiamato la funzione aggiornata, possono essere necessari fino a 10 minuti prima che i dati di servizio inizino a comparire nel pannello di controllo di Application Signals nella console CloudWatch.

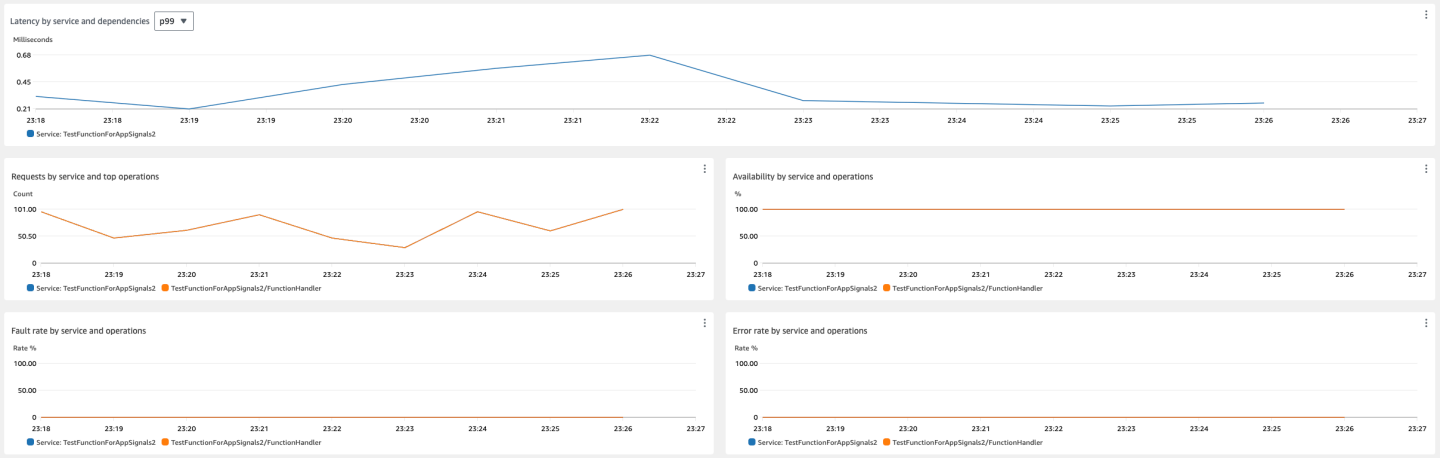
Utilizzo del pannello di controllo di Application Signals

Dopo aver abilitato Application Signals per la tua funzione, puoi visualizzare i parametri dell'applicazione nella console CloudWatch. Puoi visualizzare rapidamente il pannello di controllo di Application Signals associato dalla console Lambda con i seguenti passaggi:

Per visualizzare il pannello di controllo di Application Signals relativo alla tua funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere la funzione.
3. Selezionare la scheda Monitor (Monitora).
4. Scegli il pulsante Visualizza Application Signals. Verrà aperta direttamente la panoramica di Application Signals per il tuo servizio nella console CloudWatch.

Ad esempio, la schermata seguente mostra i parametri relativi a latenza, numero di richieste, disponibilità, tasso di guasti e tasso di errore per una funzione in una finestra temporale di 10 minuti.



Per sfruttare al meglio l'integrazione con Application Signals, puoi creare obiettivi a livello di servizio (SLO) per la tua applicazione. Ad esempio, puoi creare SLO di latenza per garantire che l'applicazione risponda rapidamente alle richieste degli utenti e SLO di disponibilità per monitorare il tempo di attività. Gli SLO possono aiutarti a rilevare il peggioramento delle prestazioni o le interruzioni prima che abbiano un impatto sugli utenti. Per ulteriori informazioni, consulta Service level objectives (SLOs) nella Guida per l'utente di Amazon CloudWatch.

Esecuzione da remoto del debug delle funzioni Lambda con Visual Studio Code

Con la funzionalità di debug remoto nel AWS Toolkit for Visual Studio Code, puoi eseguire il debug delle funzioni Lambda in esecuzione direttamente nel cloud AWS. Ciò è utile quando si esaminano problemi difficili da replicare localmente o da diagnosticare solo con i log.

Con il debug da remoto, puoi:

- Impostare punti di interruzione nel codice della funzione Lambda.
- Eseguire il codice in tempo reale.
- Ispezionare le variabili e lo stato durante il runtime.
- Funzioni di debug Lambda implementate su AWS, incluse quelle in VPC o con autorizzazioni IAM specifiche.

Runtime supportati

Il debug da remoto è supportato per i seguenti runtime:

- Python (AL2023)
- Java
- JavaScript/Node.js (AL2023)

> **Note**
>
> Il debug da remoto è supportato per le architetture sia x86_64 sia arm64.

Sicurezza e debug da remoto

Il debug da remoto opera entro i limiti di sicurezza Lambda esistenti. Gli utenti possono collegare livelli a una funzione utilizzando l'autorizzazione `UpdateFunctionConfiguration`, che ha già la possibilità di accedere alle variabili di ambiente e alla configurazione della funzione. Il debug da remoto non va oltre queste autorizzazioni esistenti. Aggiunge invece ulteriori controlli di sicurezza attraverso il tunneling sicuro e la gestione automatica delle sessioni. Inoltre, il debug da remoto è interamente una funzionalità controllata dal cliente che richiede autorizzazioni e azioni esplicite:

- Creazione di tunnel IoT sicuri: il Toolkit AWS deve creare un tunnel IoT sicuro, cosa che si verifica solo con l'autorizzazione esplicita dell'utente che usa `iot:OpenTunnel`.
- Gestione di allegati e token del livello di debug: il processo di debug mantiene la sicurezza attraverso questi controlli:
  - Il livello di debug deve essere collegato alla funzione Lambda e questo processo richiede le seguenti autorizzazioni: `lambda:UpdateFunctionConfiguration` e `lambda:GetLayerVersion`.
  - Un token di sicurezza (generato tramite `iot:OpenTunnel`) deve essere aggiornato nella variabile di ambiente della funzione prima di ogni sessione di debug, il che richiede anch'esso `lambda:UpdateFunctionConfiguration`.
  - Per motivi di sicurezza, questo token viene ruotato automaticamente e il livello di debug viene rimosso automaticamente alla fine di ogni sessione di debug e non può essere riutilizzato.

> **Note**
>
> Il debug da remoto è supportato per le architetture sia x86_64 sia arm64.

Prerequisiti

Prima di iniziare il debug da remoto, assicurati di disporre di quanto riportato di seguito:

1. Una funzione Lambda implementata sul tuo account AWS.
2. AWS Toolkit for Visual Studio Code. Consulta Configurazione del AWS Toolkit for Visual Studio Code per le istruzioni di installazione.
3. La versione del Toolkit AWS installata deve essere la 3.69.0 o successiva.
4. Credenziali AWS configurate in AWS Toolkit for Visual Studio Code. Per ulteriori informazioni, consulta Autenticazione e controllo degli accessi.

Debug da remoto delle funzioni Lambda

Esegui questi passaggi per avviare una sessione di debug da remoto:

1. Apri l'Esploratore AWS in VS Code selezionando l'icona AWS nella barra laterale sinistra.
2. Espandi la sezione Lambda per vedere le tue funzioni.

3. Fai clic con il pulsante destro del mouse sulla funzione di cui desideri eseguire il debug.
4. Dal menu contestuale, seleziona Richiama da remoto.
5. Nella finestra di invocazione che si apre, seleziona la casella Abilita debug.
6. Fai clic su Richiama per avviare la sessione di debug remota.

> **Note**
>
> Le funzioni Lambda hanno un limite combinato di 250 MB per il codice della funzione e tutti i livelli collegati. Il livello di debug da remoto aggiunge circa 40 MB alle dimensioni della funzione.

Una sessione di debug da remoto termina quando:

- Scegliere Rimuovi configurazione debug dalla schermata di configurazione Richiama da remoto.
- Selezionare l'icona di disconnessione nei controlli di debug di VS Code.
- Selezionare il file del gestore nell'editor VS Code.

> **Note**
>
> Il livello di debug viene rimosso automaticamente dopo 60 secondi di inattività dopo l'ultima invocazione.

Disabilitazione de debug da remoto

Puoi disabilitare questa funzionalità in tre modi:

- Nega aggiornamenti delle funzioni: imposta `lambda:UpdateFunctionConfiguration` su `deny`.
- Limita autorizzazioni IoT: nega le autorizzazioni relative all'IoT.
- Blocca livelli di debug: nega `lambda:GetLayerVersion` per i seguenti ARN:
  - `arn:aws:lambda:*:*:layer:LDKLayerX86:*`
  - `arn:aws:lambda:*:*:layer:LDKLayerArm64:*`

> **Note**
>
> La disabilitazione di questa funzionalità impedisce l'aggiunta del livello di debug durante gli aggiornamenti della configurazione delle funzioni.

Informazioni aggiuntive

Per ulteriori informazioni sull'utilizzo di Lambda in VS Code, consulta Sviluppo locale di funzioni Lambda con VS Code.

Per istruzioni dettagliate sulla risoluzione dei problemi, sui casi d'uso avanzati e sulla disponibilità delle regioni, consulta Debug remoto delle funzioni Lambda nella Guida per l'utente di AWS Toolkit for Visual Studio Code.

Gestione delle dipendenze Lambda con i livelli

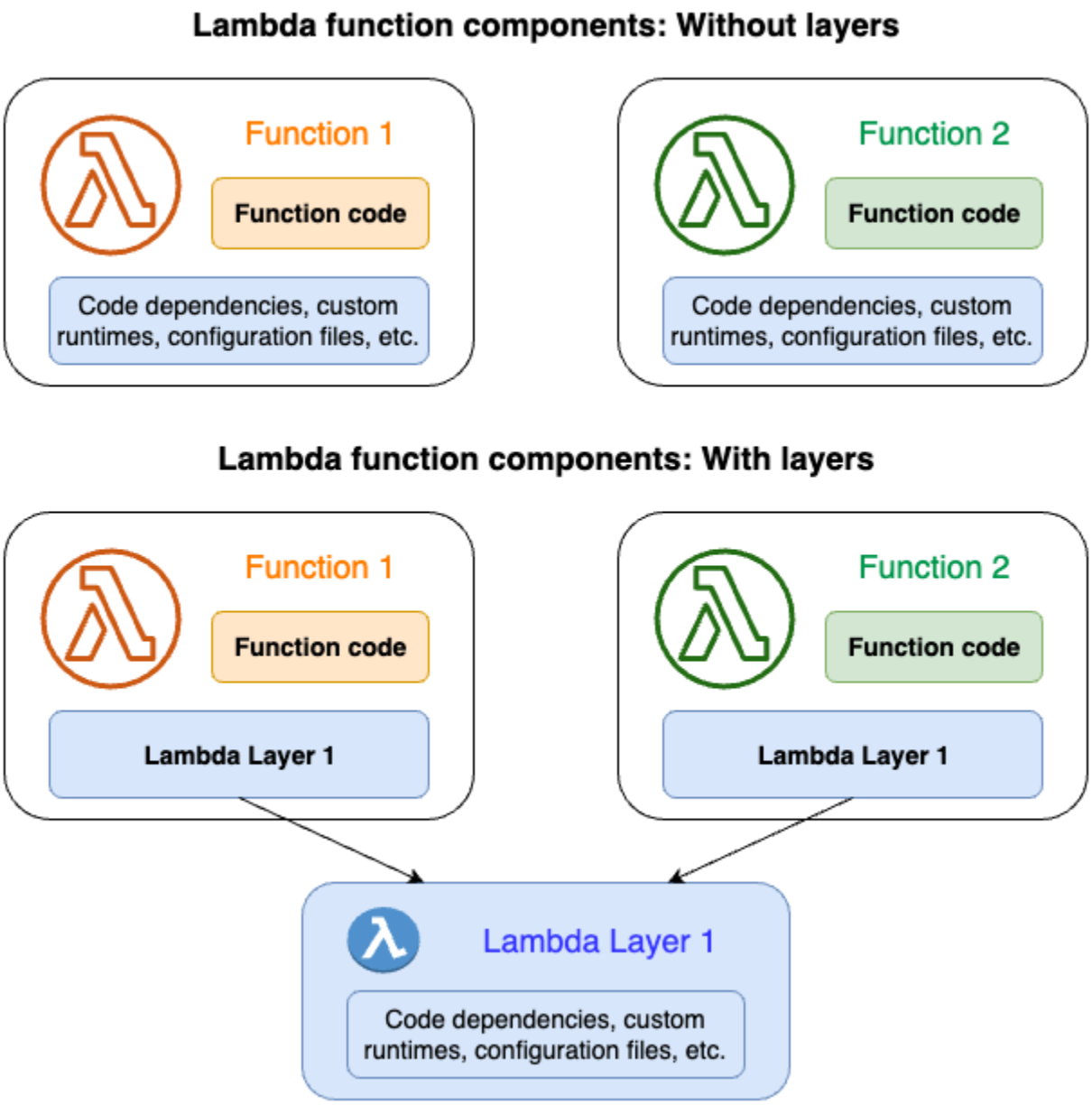
Un livello Lambda è un archivio di file .zip che può contenere codice o dati aggiuntivi. I livelli di solito contengono dipendenze dalla libreria, un runtime personalizzato o file di configurazione.

Esistono diversi motivi per cui potresti prendere in considerazione l'utilizzo dei livelli:

- Per ridurre le dimensioni dei pacchetti di implementazione. Invece di includere tutte le dipendenze delle funzioni insieme al codice della funzione nel pacchetto di implementazione, inseriscile in un livello. Ciò mantiene i pacchetti di implementazione piccoli e organizzati.
- Per separare la logica delle funzioni di base dalle dipendenze. Con i livelli, puoi aggiornare le dipendenze delle funzioni indipendentemente dal codice della funzione e viceversa. Ciò favorisce la separazione dei problemi e ti aiuta a concentrarti sulla logica funzionale.
- Per condividere le dipendenze tra più funzioni. Dopo aver creato un livello, puoi applicarlo a qualsiasi numero di funzioni del tuo account. Senza livelli, è necessario includere le stesse dipendenze in ogni singolo pacchetto di implementazione.
- Per utilizzare l'editor di codice della console Lambda. L'editor di codice è uno strumento utile per testare rapidamente aggiornamenti minori del codice funzionale. Tuttavia, non è possibile utilizzare l'editor se la dimensione del pacchetto di implementazione è troppo grande. L'uso dei livelli riduce le dimensioni del pacchetto e può sbloccare l'utilizzo dell'editor di codice.
- Per bloccare una versione SDK incorporata.Gli SDK incorporati possono cambiare senza preavviso man mano che AWS rilascia nuovi servizi e funzionalità. Puoi bloccare una versione dell'SDK creando un livello Lambda con la versione specifica necessaria. La funzione utilizza quindi sempre la versione nel livello, anche se la versione incorporata nel servizio cambia.

Se utilizzi le funzioni Lambda in Go o Rust, ti consigliamo di non utilizzare i livelli. Per le funzioni Go e Rust, fornisci il codice della funzione come eseguibile, che include il codice di funzione compilato insieme a tutte le sue dipendenze. L'inserimento delle dipendenze in un livello impone alla funzione di caricare manualmente gli assembly aggiuntivi durante la fase di inizializzazione, per cui i tempi di avvio a freddo possono aumentare. Per prestazioni ottimali per le funzioni Go e Rust, includi le tue dipendenze insieme al pacchetto di implementazione.

Il diagramma seguente illustra le principali differenze di architettura tra due funzioni che condividono dipendenze. Una utilizza i livelli Lambda e l'altra no.



Quando si include un livello in una funzione Lambda, Lambda estrae il contenuto del livello nella directory `/opt` nell'ambiente di esecuzione della funzione. Tutti i runtime Lambda supportati in modo nativo includono percorsi a directory specifiche all'interno della directory `/opt`. Ciò consente alla funzione di accedere al contenuto dei livelli. Per ulteriori informazioni su questi percorsi specifici e su come creare correttamente i pacchetti per i livelli, consulta the section called “Creazione di pacchetti dei livelli”.

Puoi includere fino a cinque livelli per funzione. Inoltre, è possibile utilizzare i livelli solo con funzioni Lambda implementate come archivio di file con estensione zip. Per funzioni definite come immagine

del container, quando si crea l'immagine del container viene creato un pacchetto del runtime preferito e tutte le dipendenze del codice. Per ulteriori informazioni, consulta Operazioni con i livelli e le estensioni Lambda nelle immagini di container nel Compute Blog AWS.

Argomenti

- Come usare i livelli
- Livelli e versioni di livelli
- Creazione di pacchetti del contenuto dei livelli
- Creazione ed eliminazione di livelli in Lambda
- Aggiunta di livelli alle funzioni
- Utilizzo di AWS CloudFormation con i livelli
- Utilizzo di AWS SAM con i livelli

Come usare i livelli

Per creare un livello, raccogli le tue dipendenze in un file .zip, in modo simile a come crei un normale pacchetto di implementazione. Più specificamente, il processo generale di creazione e utilizzo dei livelli prevede questi tre passaggi:

- Innanzitutto, crea un pacchetto per il contenuto del livello. Ciò significa creare un archivio di file .zip. Per ulteriori informazioni, consulta the section called “Creazione di pacchetti dei livelli”.
- Quindi, crea il livello in Lambda. Per ulteriori informazioni, consulta the section called “Creazione ed eliminazione di livelli”.
- Aggiungi il livello alla tua funzione). Per ulteriori informazioni, consulta the section called “Aggiunta di livelli”.

Livelli e versioni di livelli

Una versione di livello è un'istantanea immutabile di una versione specifica di un livello. Quando si crea un nuovo livello, Lambda crea una nuova versione del livello con un numero di versione pari a 1. Ogni volta che si pubblica un aggiornamento del livello, Lambda incrementa il numero di versione e crea una nuova versione.

Ogni flusso è identificato in modo univoco da un nome della risorsa Amazon (ARN) univoco. Quando si aggiunge un livello alla funzione, è necessario specificare la versione esatta del livello che si desidera utilizzare (ad es. `arn:aws:lambda:us-east-1:123456789012:layer:my-layer:1`).

Creazione di pacchetti del contenuto dei livelli

Un livello Lambda è un archivio di file .zip che può contenere codice o dati aggiuntivi. I livelli di solito contengono dipendenze dalla libreria, un runtime personalizzato o file di configurazione.

In questa sezione viene descritto come creare pacchetti del contenuto dei livelli. Per ulteriori informazioni concettuali sui livelli e su come e perché utilizzarli, consulta Livelli Lambda.

Il primo passaggio per creare un livello consiste nel raggruppare tutto il contenuto del livello in un archivio di file .zip. Perché le funzioni Lambda vengano eseguite su Amazon Linux, il contenuto del livello deve essere in grado di compilare e creare in un ambiente Linux.

Per garantire che il contenuto del livello funzioni correttamente in un ambiente Linux, si consiglia di creare il contenuto utilizzando uno strumento come Docker.

Argomenti

- Percorsi dei livelli per ciascun runtime Lambda

Percorsi dei livelli per ciascun runtime Lambda

Quando si aggiunge un livello a una funzione, Lambda carica il contenuto del livello nella directory `/opt` di quell'ambiente di esecuzione. Per ogni runtime Lambda, la variabile `PATH` include percorsi di cartelle specifici nella directory `/opt`. Per garantire che la variabile raccolga il contenuto del livello, il file .zip del livello dovrebbe avere le sue dipendenze nei seguenti percorsi di cartella:

Runtime	Percorso
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`
	`python/lib/` *`python3.x`* `/site-packages` (directory del sito)

Runtime	Percorso
Java	`java/lib` (`CLASSPATH`)
Ruby	`ruby/gems/3.4.0` (`GEM_PATH`)
	`ruby/lib` (`RUBYLIB`)
Tutti i runtime	`bin` (`PATH`)
	`lib` (`LD_LIBRARY_PATH`)

Negli esempi seguenti viene illustrato come strutturare le cartelle per l'archivio .zip del livello.

Node.js

Example struttura di file per l'SDK AWS X-Ray per 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 struttura dei file per gem JSON

```
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 struttura dei file per il file JAR Jackson

```
layer_content.zip
# java
    # lib
        # jackson-core-2.17.0.jar
        # <other potential dependencies>
        # ...
```

All

Example struttura dei file per la libreria JQ

```
jq.zip
# bin/jq
```

Per istruzioni specifiche del linguaggio sull'impacchettamento, la creazione e l'aggiunta di un livello, consulta le pagine seguenti:

- Node.js – the section called “Livelli”
- Python – the section called “Livelli”
- Ruby – the section called “Livelli”
- Java: the section called “Livelli”

Consigliamo di non utilizzare i livelli per gestire le dipendenze per le funzioni Lambda scritte in Go. Questo perché le funzioni Lambda in Go vengono compilate in un unico eseguibile, che viene fornito a Lambda quando si distribuisce la funzione. Questo eseguibile contiene il codice di funzione compilato, insieme a tutte le sue dipendenze. L'uso dei livelli non solo complica questo processo, ma comporta anche un aumento dei tempi di avvio a freddo, poiché le funzioni devono caricare manualmente assiemi aggiuntivi in memoria durante la fase di inizializzazione.

Per utilizzare dipendenze esterne con i gestori Go, includile direttamente nel pacchetto di implementazione.

Creazione ed eliminazione di livelli in Lambda

Un livello Lambda è un archivio di file .zip che può contenere codice o dati aggiuntivi. I livelli di solito contengono dipendenze dalla libreria, un runtime personalizzato o file di configurazione.

In questa sezione viene descritto come creare ed eliminare livelli in Lambda. Per ulteriori informazioni concettuali sui livelli e su come e perché utilizzarli, consulta Livelli Lambda.

Dopo aver creato i pacchetti del contenuto dei livelli, il passaggio successivo consiste nel creare il livello in Lambda. In questa sezione viene descritto come creare ed eliminare i livelli utilizzando solo la console Lambda o l'API Lambda. Per creare un livello utilizzando AWS CloudFormation, consulta the section called “Livelli con CloudFormation”. Per creare un livello utilizzando AWS Serverless Application Model (AWS SAM), consulta the section called “Livelli con AWS SAM”.

Argomenti

- Creazione di un livello
- Eliminazione della versione di un livello

Creazione di un livello

Per creare un livello, puoi caricare l'archivio di file con estensione .zip dal computer locale o da Amazon Simple Storage Service (Amazon S3). Lambda estrae il contenuto del livello nella directory `/opt` durante la configurazione dell'ambiente di esecuzione per la funzione.

I livelli possono avere una o più versioni. Quando si crea un livello, Lambda imposta la versione del livello su 1. È possibile modificare le autorizzazioni su una versione di livello esistente in qualsiasi momento. Tuttavia, per aggiornare il codice o apportare altre modifiche alla configurazione, è necessario creare una nuova versione del livello.

Creazione di un livello (console)

1. Apri la pagina Layers (Livelli) nella console Lambda.
2. Scegli Create layer (Crea livello).
3. In Layer configuration (Configurazione livello), per Nome, immettere un nome del livello.
4. (Facoltativo) In Description (Descrizione), immetti una descrizione per il livello.
5. Per caricare il codice del livello, esegui una delle seguenti operazioni:

- Per caricare un file con estensione .zip dal computer, scegliere Carica un file .zip. Scegli Carica per selezionare il file con estensione .zip locale.
- Per caricare un file da Amazon S3, scegli Upload a file from Amazon S3 (Carica un file da Amazon S3). Quindi, per l'URL del link Amazon S3, immetti un collegamento al file.

6. (Facoltativo) Per Architetture compatibili, scegli un valore o entrambi i valori. Per ulteriori informazioni, consulta the section called “Set di istruzioni (ARM/x86)”.
7. (Facoltativo) Per Runtime compatibili, scegli i runtime con cui il tuo livello è compatibile.
8. (Facoltativo) Per Licenza, inserisci tutte le informazioni necessarie sulla licenza.
9. Scegli Create (Crea).

In alternativa, puoi eseguire il comando publish-layer-version AWS Command Line Interface (CLI). Esempio:

```
aws lambda publish-layer-version --layer-name my-layer --zip-file fileb://layer.zip --
compatible-runtimes python3.14
```

Ogni volta che esegui `publish-layer-version`, viene creata una nuova versione del livello.

Eliminazione della versione di un livello

Per eliminare una versione di livello, utilizzate l'operazione DeleteLayerVersionAPI. Ad esempio, eseguite il delete-layer-version AWS CLI comando con il nome e la versione del livello specificati.

```
aws lambda delete-layer-version --layer-name my-layer --version-number 1
```

Quando elimini la versione di un livello, non puoi più configurare una funzione Lambda per utilizzarla. Tuttavia, qualsiasi funzione che già utilizza la versione continua ad averne accesso. Inoltre, Lambda non riutilizza mai i numeri di versione per il nome di un livello.

Quando si calcolano le quote, l'eliminazione di una versione di livello significa che non viene più conteggiata come parte della quota predefinita di 75 GB per l'archiviazione di funzioni e livelli. Tuttavia, per le funzioni che utilizzano una versione di livello eliminata, il contenuto del livello viene comunque conteggiato ai fini della quota di dimensione del pacchetto di implementazione della funzione (ad esempio 250 MB per gli archivi di file .zip).

Aggiunta di livelli alle funzioni

Un livello Lambda è un archivio di file .zip che può contenere codice o dati aggiuntivi. I livelli di solito contengono dipendenze dalla libreria, un runtime personalizzato o file di configurazione.

In questa sezione viene spiegato come aggiungere un livello a una funzione Lambda. Per ulteriori informazioni concettuali sui livelli e su come e perché utilizzarli, consulta Livelli Lambda.

Prima di poter configurare una funzione Lambda per utilizzare un livello, è necessario:

- Crea un pacchetto per il contenuto del livello
- Crea un livello in Lambda
- Assicurati di avere l'autorizzazione per chiamare l'GetLayerVersionAPI nella versione del layer. Per le funzioni incluse nella tua Account AWS, devi avere questa autorizzazione nella tua politica utente. Per utilizzare un livello in un altro account, il proprietario di quell'account deve concedere l'autorizzazione per l'account in una policy basata sulle risorse. Per alcuni esempi, consulta the section called “Accesso ai livelli per altri account”.

Puoi aggiungere fino a cinque livelli a una funzione Lambda. La dimensione totale non decompressa della funzione e di tutti i livelli non può superare la quota della dimensione del pacchetto di distribuzione non compresso di 250 MB. Per ulteriori informazioni, consulta Quote di Lambda.

Le tue funzioni possono continuare a utilizzare qualsiasi versione del livello che hai già aggiunto, anche dopo che la versione del livello è stata eliminata o dopo la revoca del tuo permesso di accesso al livello. Non è tuttavia possibile creare una nuova funzione che utilizza la versione di un livello eliminato.

Aggiunta di un livello a una funzione

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la funzione.
3. Scorri verso il basso fino alla sezione Livelli, quindi scegli Aggiungi un livello.
4. In Scegli un livello, scegli un'origine del livello:
    a. AWS livelli: scegli dall'elenco delle estensioni AWS gestite.
    b. Livelli personalizzati: scegli un livello creato nel tuo Account AWS.

   c. Specificare un ARN: per utilizzare un layer di un altro livello Account AWS, ad esempio un'estensione di terze parti, inserisci Amazon Resource Name (ARN).

5. Scegliere Aggiungi.

L'ordine in cui si aggiungono i livelli è l'ordine in cui Lambda unisce il contenuto del livello nell'ambiente di esecuzione. Puoi modificare l'ordine di unione dei livelli utilizzando la console.

Aggiornamento dell'ordine di unione dei livelli per la tua funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegli la funzione da configurare.
3. In Layers (Livelli), scegli Edit (Modifica)
4. Scegli uno dei livelli.
5. Scegli Merge earlier (Unisci prima) o Merge later (Unisci in un secondo momento) per regolare l'ordine dei livelli.
6. Scegli Save (Salva).

I livelli sono suddivisi in versioni. Il contenuto di ogni versione di livello non è modificabile. Il proprietario del livello può rilasciare nuove versioni del livello in modo da fornire contenuto appropriato. È possibile utilizzare la console per aggiornare la versione del livello collegata alle funzioni.

Aggiornamento delle versioni del livello per la funzione (console)

1. Apri la pagina Layers (Livelli) nella console Lambda.
2. Scegli il livello per il quale desideri aggiornare la versione.
3. Seleziona la scheda Funzioni che utilizzano questa versione.
4. Scegli le funzioni che desideri modificare, quindi scegli Modifica.
5. Da Versione livello, seleziona la versione del livello a cui passare.
6. Scegliere Update functions (Aggiorna funzioni).

Non è possibile aggiornare le versioni del Function Layer tra più account AWS .

Ricerca di informazioni sul livello

Per trovare livelli nel tuo account compatibili con il runtime della tua funzione, usa l'ListLayersAPI. Ad esempio, è possibile utilizzare il seguente comando list-layers (AWS Command Line Interface CLI):

```
aws lambda list-layers --compatible-runtime python3.14
```

Verrà visualizzato un output simile al seguente:

```
{
    "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"
                ]
            }
        }
    ]
}
```

Per elencare tutti i livelli nell'account, ometti l'opzione `--compatible-runtime`. I dettagli della risposta mostrano la versione più recente di ogni livello.

È inoltre possibile ottenere la versione più recente di un layer utilizzando l'API. ListLayerVersions Ad esempio, puoi utilizzare il seguente comando `list-layer-versions` della CLI:

```
aws lambda list-layer-versions --layer-name my-layer
```

Verrà visualizzato un output simile al seguente:

```
{
    "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"
            ]
        }
    ]
}
```

Utilizzo di AWS CloudFormation con i livelli

È possibile utilizzare CloudFormation per creare un livello e associarlo alla funzione Lambda. Nel modello di esempio seguente viene creato un livello denominato `my-lambda-layer` che viene collegato alla funzione Lambda utilizzando la proprietà Layers (Livelli).

In questo esempio, il modello specifica il nome della risorsa Amazon (ARN) di un ruolo di esecuzione IAM esistente. Puoi anche creare un nuovo ruolo di esecuzione nel modello utilizzando la risorsa CloudFormation AWS::IAM::Role.

La funzione non necessita di autorizzazioni speciali per utilizzare i livelli.

```
---
Description: CloudFormation Template for Lambda 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
```

Utilizzo di AWS SAM con i livelli

È possibile utilizzare AWS Serverless Application Model (AWS SAM) per automatizzare la creazione dei livelli nella tua applicazione. Il tipo di risorsa `AWS::Serverless::LayerVersion` crea una versione del layer a cui è possibile fare riferimento dalla configurazione della funzione Lambda.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: AWS SAM Template for Lambda 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
```

Aumentare le funzioni Lambda utilizzando le estensioni Lambda

È possibile utilizzare le estensioni Lambda per potenziare le funzioni Lambda. Ad esempio, utilizza le estensioni Lambda per integrare le funzioni con i tuoi strumenti di monitoraggio, osservabilità, sicurezza e governance preferiti. È possibile scegliere tra un ampio set di strumenti forniti dai Partner AWS Lambda oppure creare estensioni Lambda personalizzate.

Lambda supporta estensioni interne ed esterne. Un'estensione esterna viene eseguita come processo indipendente nell'ambiente di esecuzione e continua a funzionare dopo che l'invocazione della funzione è completamente elaborata. Poiché le estensioni vengono eseguite come processi separati, è possibile scriverle in un linguaggio diverso da quello della funzione. Tutti i Runtime Lambda supportano le estensioni.

Un'estensione interna viene eseguita come parte del processo di runtime. La funzione accede alle estensioni interne utilizzando script wrapper o meccanismi in-process come `JAVA_TOOL_OPTIONS`. Per ulteriori informazioni, consulta Modifica dell'ambiente di runtime.

È possibile aggiungere estensioni a una funzione utilizzando la console Lambda, AWS CLI (CloudFormation) o l'Infrastructure as Code e strumenti (IaC) di codice quali AWS Command Line Interface, AWS Serverless Application Model (AWS SAM) e Terraform.

Ti viene addebitato il runtime consumato dall'estensione (in incrementi di 1 ms). Non c'è alcun costo per installare le proprie estensioni. Per ulteriori informazioni sui prezzi delle estensioni, consulta la sezione Prezzi AWS Lambda. Per informazioni sui prezzi per le estensioni dei partner, consulta i siti Web dei partner. Consulta the section called “Partner con estensioni” per l'elenco delle estensioni ufficiali dei partner.

Per un tutorial sulle estensioni e su come usarle con le funzioni Lambda, consultare il Workshop sulle estensioni di AWS Lambda.

Argomenti

- Ambiente di esecuzione
- Impatto su prestazioni e risorse
- Autorizzazioni
- Configurazione delle estensioni Lambda
- Partner con estensioni AWS Lambda

- Utilizzo dell'API Extensions di Lambda per creare estensioni
- Accesso ai dati di telemetria in tempo reale per le estensioni tramite l'API Telemetry

Ambiente di esecuzione

Lambda richiama la funzione in un ambiente di esecuzione, che fornisce un ambiente di runtime sicuro e isolato. L'ambiente di esecuzione gestisce le risorse necessarie per eseguire la funzione e fornisce il supporto del ciclo di vita per il runtime e le estensioni della funzione.

Il ciclo di vita dell'ambiente di esecuzione prevede le seguenti fasi:

- **`Init`**: in questa fase, Lambda crea o sblocca un ambiente di esecuzione con le risorse configurate, scarica il codice per la funzione e tutti i livelli, inizializza le estensioni, inizializza il runtime e quindi esegue il codice di inizializzazione della funzione (il codice al di fuori del gestore principale). La fase `Init` si verifica durante la prima invocazione o prima delle invocazioni di funzione se è stata abilitata la concorrenza con provisioning.

  La fase `Init` è suddivisa in tre sottofasi: `Extension init`, `Runtime init` e `Function init`. Queste sottofasi assicurano che tutte le estensioni e il runtime completino le loro attività di configurazione prima dell'esecuzione del codice della funzione.

  Quando Lambda SnapStart è attivato, la fase `Init` si verifica quando si pubblica una versione della funzione. Lambda salva uno snapshot della memoria e dello stato del disco dell'ambiente di esecuzione inizializzato, mantiene lo snapshot crittografato e lo memorizza nella cache per l'accesso a bassa latenza. Se disponi di un hook di runtime prima del checkpoint, il codice viene eseguito alla fine della fase `Init`.
- **`Restore`** (solo SnapStart): quando si invoca per la prima volta una funzione SnapStart e man mano che la funzione viene aumentata, Lambda riprende i nuovi ambienti di esecuzione dallo snapshot persistente invece di inizializzare la funzione da zero. Se disponi di un hook di runtime dopo il ripristino, il codice viene eseguito alla fine della fase `Restore`. Ti sarà addebitata la durata degli hook di runtime dopo il ripristino. Il runtime deve essere caricato e gli hook di runtime dopo il ripristino devono essere completati entro il limite di timeout (10 secondi). Altrimenti, otterrai un'eccezione SnapStartTimeoutException. Al termine della fase `Restore`, Lambda chiama il gestore della funzione (Invoca fase).
- **`Invoke`**: in questa fase, Lambda invoca il gestore della funzione. Dopo che la funzione è stata completata, Lambda si prepara a gestire un'altra invocazione di funzione.

- **`Shutdown`**: questa fase viene attivata se la funzione Lambda non riceve alcuna invocazione per un certo periodo. Nella fase `Shutdown`, Lambda chiude il runtime, avvisa le estensioni per farle fermare in modo pulito, e poi rimuove l'ambiente. Lambda invia un evento `Shutdown` a ogni estensione; l'evento comunica all'estensione che l'ambiente sta per essere chiuso.

Durante la fase `Init`, Lambda estrae i livelli contenenti estensioni nella directory `/opt` nell'ambiente di esecuzione. Lambda cerca le estensioni nella directory `/opt/extensions/`, interpreta ogni file come un bootstrap eseguibile per avviare l'estensione e avvia tutte le estensioni in parallelo.

Impatto su prestazioni e risorse

Le dimensioni delle estensioni della funzione vengono conteggiate per il limite di dimensioni del pacchetto di distribuzione. Per un archivio di file .zip, la dimensione totale decompressa della funzione e di tutte le estensioni non può superare il limite della dimensione decompressa del pacchetto di distribuzione pari a 250 MB.

Le estensioni possono influire sulle prestazioni della funzione perché condividono risorse funzionali quali CPU, memoria e archiviazione. Ad esempio, se un'estensione esegue operazioni ad alta intensità di calcolo, è possibile che la durata dell'esecuzione della funzione aumenti.

Ogni estensione deve completare la sua inizializzazione prima che Lambda richiami la funzione. Pertanto, un'estensione che consuma tempo di inizializzazione significativo può aumentare la latenza della chiamata di funzione.

Per misurare il tempo aggiuntivo impiegato dall'estensione dopo l'esecuzione della funzione, è possibile utilizzare la `PostRuntimeExtensionsDuration` metrica della funzione. Per misurare l'aumento della memoria utilizzata, è possibile utilizzare la metrica `MaxMemoryUsed`. Per comprendere l'impatto di un'estensione specifica, è possibile eseguire diverse versioni delle funzioni affiancate.

> **Note**
>
> La metrica `MaxMemoryUsed` è una delle Metriche raccolte da Lambda Insights e non una metrica nativa di Lambda.

Autorizzazioni

Le estensioni hanno accesso alle stesse risorse delle funzioni. Poiché le estensioni vengono eseguite nello stesso ambiente della funzione, le autorizzazioni vengono condivise tra la funzione e l'estensione.

Per un archivio di file .zip, è possibile creare un modello CloudFormation per semplificare l'operazione di allegare la stessa configurazione dell'estensione, incluse le autorizzazioni AWS Identity and Access Management (IAM) a più funzioni.

Configurazione delle estensioni Lambda

Configurazione delle estensioni (archivio di file .zip)

È possibile aggiungere un'estensione alla funzione come livello Lambda. L'utilizzo dei livelli consente di condividere le estensioni all'interno dell'organizzazione o all'intera community di sviluppatori Lambda. È possibile aggiungere una o più estensioni a un livello. È possibile registrare fino a 10 estensioni per una funzione.

Si aggiunge l'estensione alla funzione utilizzando lo stesso metodo che si farebbe per qualsiasi livello. Per ulteriori informazioni, consulta Livelli Lambda.

Aggiungi un'estensione alla tua funzione (console)

1. Aprire la pagina Funzioni della console Lambda.
2. Scegliere una funzione.
3. Seleziona la scheda Codice se non è già selezionata.
4. In Livelli, scegli Modifica.
5. Per Scegli un livello, scegliere Specifica un ARN.
6. In Specifica un ARN, inserire l'Amazon Resource Name (ARN) di un livello di estensione.
7. Scegli Aggiungi.

Utilizzo delle estensioni nelle immagini di container

È possibile aggiungere estensioni all'immagine di container. L'impostazione ENTRYPOINT dell'immagine di container specifica il processo principale per la funzione. Configurare l'impostazione ENTRYPOINT nel Dockerfile o come sostituzione nella configurazione della funzione.

È possibile eseguire più processi all'interno di un container. Lambda gestisce il ciclo di vita del processo principale ed eventuali processi aggiuntivi. Lambda utilizza l'API Estensioni per gestire il ciclo di vita dell'estensione.

Esempio: aggiunta di un'estensione esterna

Un'estensione esterna viene eseguita in un processo separato dalla funzione Lambda. Lambda avvia un processo per ogni estensione nella directory `/opt/extensions/`. Lambda utilizza

l'API Estensioni per gestire il ciclo di vita dell'estensione. Dopo che la funzione è stata eseguita completamente, Lambda invia un evento `Shutdown` a ciascuna estensione esterna.

Example di aggiungere un'estensione esterna a un'immagine di base 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
```

Passaggi successivi

Per ulteriori informazioni sulle estensioni, si consiglia di utilizzare le seguenti risorse:

- Per un esempio di lavoro di base, consulta Creazione di estensioni per AWS Lambda nel blog Compute di AWS.
- Per informazioni sulle estensioni fornite dai partner AWS Lambda, consulta Introduzione delle estensioni AWS Lambda nel blog Compute di AWS.
- Per visualizzare le estensioni di esempio e gli script wrapper disponibili, consulta le Estensioni AWS Lambda negli esempi del repository AWS GitHub.

Partner con estensioni AWS Lambda

AWS Lambda ha collaborato con diverse entità di terze parti per fornire estensioni da integrare con le funzioni Lambda. L'elenco seguente descrive le estensioni di terze parti pronte all'uso in qualsiasi momento.

- AppDynamics: fornisce la strumentazione automatica delle funzioni Node.js o Python Lambda, fornendo visibilità e avvisi sulle prestazioni delle funzioni.
- Axiom: fornisce i pannelli di controllo per il monitoraggio delle prestazioni della funzione Lambda e i parametri aggregati a livello di sistema.
- Datadog: fornisce visibilità completa e in tempo reale alle applicazioni serverless tramite l'uso di metriche, tracce e log.
- Dynatrace: fornisce visibilità su tracce e metriche e sfrutta l'intelligenza artificiale per il rilevamento automatico degli errori e l'analisi delle cause principali nell'intero stack di applicazioni.
- Elastic: fornisce il monitoraggio delle prestazioni delle applicazioni (APM) per identificare e risolvere i problemi principali utilizzando tracce, parametri e log correlati.
- Epsagon – attende gli eventi di richiamo, memorizza le tracce e le invia in parallelo alle esecuzioni delle funzioni Lambda.
- Fastly– protegge le funzioni Lambda da attività sospette, come attacchi in stile iniezione, acquisizione di account tramite furto di credenziali, bot dannosi e abuso delle API.
- HashiCorp Vault – gestisce i segreti e li rende disponibili affinché gli sviluppatori possano utilizzarli all'interno del codice della funzione, senza rendere le funzioni consapevoli di Vault.
- Honeycomb – strumento di osservabilità per il debug dello stack di app.
- Lumigo – profila i richiami delle funzioni Lambda e raccoglie le metriche per la risoluzione dei problemi in ambienti serverless e di microservizi.
- New Relic – funziona insieme alle funzioni Lambda, raccogliendo, migliorando e trasportando automaticamente la telemetria alla piattaforma di osservabilità unificata di New Relic.
- Sedai – una piattaforma di gestione cloud autonoma, basata su AI/ML, che offre un'ottimizzazione continua per i team operativi del cloud per massimizzare i risparmi sui costi, le prestazioni e la disponibilità del cloud su larga scala.
- Sentry – esegue la diagnostica, corregge e ottimizza le prestazioni delle funzioni Lambda.
- Site24x7 – ottiene osservabilità in tempo reale negli ambienti Lambda
- Splunk – raccoglie parametri ad alta risoluzione e bassa latenza per un monitoraggio efficiente ed efficace delle funzioni Lambda.

- Sumo Logic – fornisce visibilità sullo stato e sulle prestazioni delle applicazioni serverless.
- Salt Security: semplifica la governance dello stato delle API e la sicurezza delle API per le funzioni Lambda attraverso la configurazione e il supporto automatizzati per diversi runtime.

AWSEstensioni gestite da

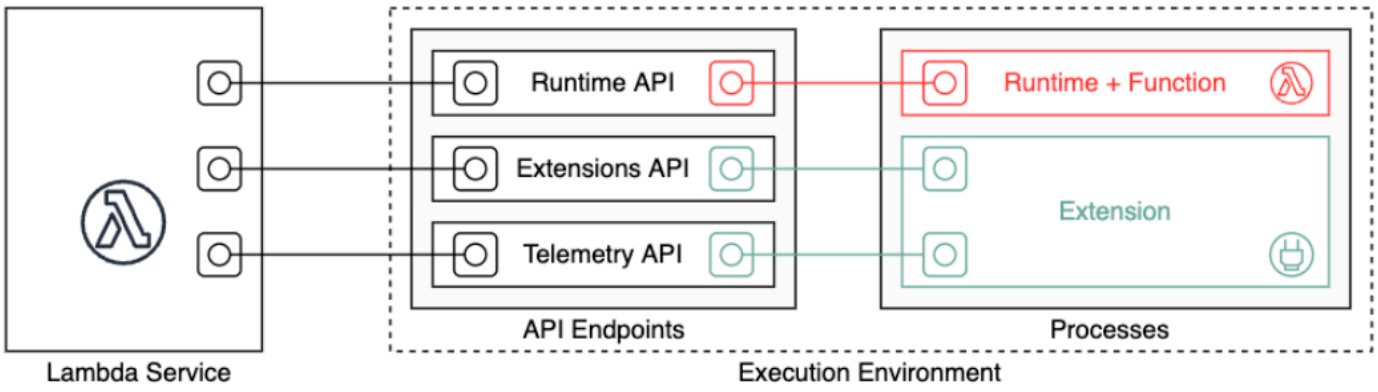
AWS fornisce le proprie estensioni gestite, tra cui:

- AWSAppConfig: utilizza flag di funzionalità e dati dinamici per aggiornare le funzioni Lambda. È inoltre possibile utilizzare questa estensione per aggiornare altre configurazioni dinamiche, come la limitazione e l'ottimizzazione delle operazioni.
- Amazon CodeGuru Profiler: migliora le prestazioni delle applicazioni e riduce i costi individuando la linea di codice più costosa di un'applicazione e fornendo consigli per migliorare il codice.
- CloudWatch Lambda Insights: monitora, risolve i problemi e ottimizza le prestazioni delle funzioni Lambda tramite dashboard automatizzati.
- AWS Distro for OpenTelemetry (ADOT): consente alle funzioni di inviare dati di traccia ai servizi di monitoraggio di AWS come AWS X-Ray e alle destinazioni che supportano OpenTelemetry, come Honeycomb e Lightstep.
- AWS Parametri e segreti : recupera in modo sicuro i parametri da AWS Systems Manager Parameter Store e i segreti da Gestione dei segreti AWS Funzioni Lambda.

Per ulteriori esempi di estensioni e progetti dimostrativi, consultare Estensioni AWS Lambda.

Utilizzo dell'API Extensions di Lambda per creare estensioni

Gli autori di funzioni Lambda utilizzano le estensioni per integrare Lambda con gli strumenti preferiti per il monitoraggio, l'osservabilità, la sicurezza e la governance. Gli autori delle funzioni possono utilizzare estensioni di AWS, AWS partner e progetti open source. Per ulteriori informazioni sull'utilizzo delle estensioni, consulta Introducing AWS Lambda Extensions on the AWS Compute Blog. In questa sezione viene descritto come utilizzare l'API di estensione Lambda, il ciclo di vita dell'ambiente di esecuzione Lambda e la documentazione di riferimento delle API di estensione Lambda.



In qualità di autore dell'estensione, l'utente utilizzare l'API estensioni per ottenere un'integrazione profonda nell'ambiente di esecuzione Lambda. L'estensione può registrarsi per funzioni ed eventi del ciclo di vita dell'ambiente di esecuzione. In risposta a questi eventi, è possibile avviare nuovi processi, eseguire la logica e controllare e partecipare a tutte le fasi del ciclo di vita di Lambda: inizializzazione, invocazione e arresto. Inoltre, è possibile utilizzare l'API Log di runtime per ricevere un flusso di log.

Un'estensione esterna viene eseguita come processo indipendente nell'ambiente di esecuzione e continua a funzionare anche dopo che la chiamata della funzione è stata completamente elaborata. Poiché le estensioni esterne vengono eseguite come processi, è possibile scriverle in un linguaggio diverso da quella della funzione. Si consiglia di implementare le estensioni utilizzando un linguaggio compilato. In questo caso, l'estensione è un binario autonomo compatibile con tutti i tempi di esecuzione supportati. Tutti i Runtime Lambda supportano le estensioni. Se si utilizza un linguaggio non compilato, assicurarsi di includere un runtime compatibile nell'estensione.

Lambda supporta anche le estensioni interne. Un'estensione interna viene eseguita come thread separato nel processo di runtime. Il runtime avvia e arresta l'estensione interna. Un modo alternativo per integrarsi con l'ambiente Lambda consiste nell'utilizzare variabili d'ambiente e script wrapper

specifici del linguaggio. Le estensioni interne consentono di configurare l'ambiente di runtime e modificare il comportamento di startup del processo di runtime.

È possibile aggiungere estensioni a una funzione in due modi. Per una funzione distribuita come archivio di file con estensione zip, è possibile distribuire l'estensione come livello. Per una funzione definita come immagine di container, le estensioni vengono aggiunte all'immagine del container.

> **Note**
>
> Per esempio estensioni e script wrapper, consulta AWS Lambda Extensions on the Samples repository. AWS GitHub

Argomenti

- Ciclo di vita dell'ambiente di esecuzione Lambda
- Riferimento all'API delle estensioni

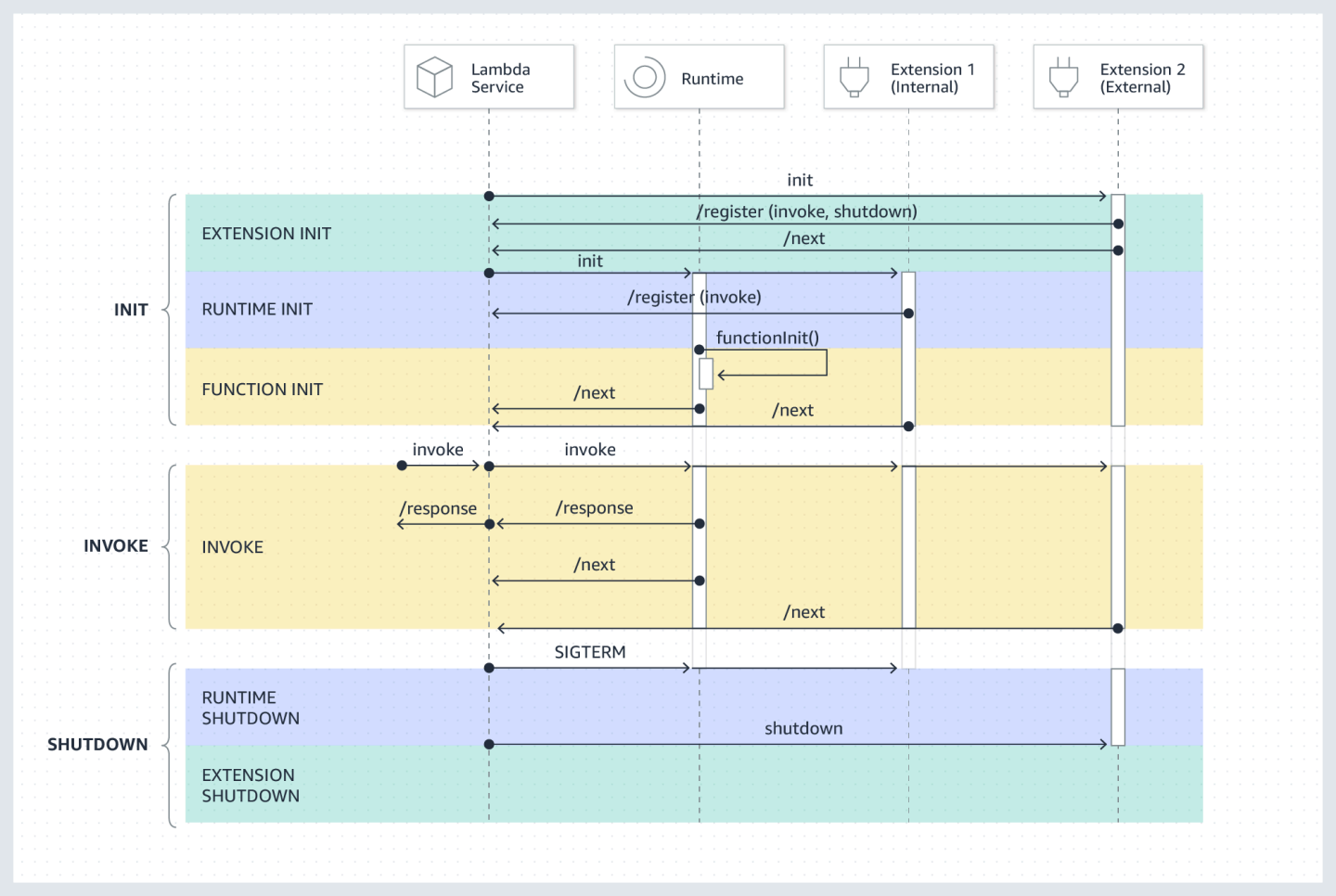
Ciclo di vita dell'ambiente di esecuzione Lambda

Il ciclo di vita dell'ambiente di esecuzione prevede le seguenti fasi:

- **`Init`**: in questa fase, Lambda crea o sblocca un ambiente di esecuzione con le risorse configurate, scarica il codice per la funzione e tutti i livelli, inizializza le estensioni, inizializza il runtime e quindi esegue il codice di inizializzazione della funzione (il codice al di fuori del gestore principale). La fase `Init` si verifica durante la prima invocazione o prima delle invocazioni di funzione se è stata abilitata la concorrenza con provisioning.

  La fase `Init` è suddivisa in tre sottofasi: `Extension init`, `Runtime init` e `Function init`. Queste sottofasi assicurano che tutte le estensioni e il runtime completino le loro attività di configurazione prima dell'esecuzione del codice della funzione.
- **`Invoke`**: in questa fase, Lambda invoca il gestore della funzione. Dopo che la funzione è stata completata, Lambda si prepara a gestire un'altra invocazione di funzione.
- **`Shutdown`**: questa fase viene attivata se la funzione Lambda non riceve alcuna invocazione per un certo periodo. Nella fase `Shutdown`, Lambda chiude il runtime, avvisa le estensioni per farle fermare in modo pulito, e poi rimuove l'ambiente. Lambda invia un evento `Shutdown` a ogni estensione; l'evento comunica all'estensione che l'ambiente sta per essere chiuso.

Ogni fase inizia con un evento da Lambda al runtime e a tutte le estensioni registrate. Il runtime e ogni estensione segnalano il completamento inviando una richiesta API `Next`. Lambda congela l'ambiente di esecuzione quando ogni processo è stato completato e non ci sono eventi in sospeso.



Argomenti

- Fase di init
- Invoca fase
- Fase di arresto
- Autorizzazioni e configurazione
- Gestione dei guasti
- Risoluzione dei problemi delle estensioni

Fase di init

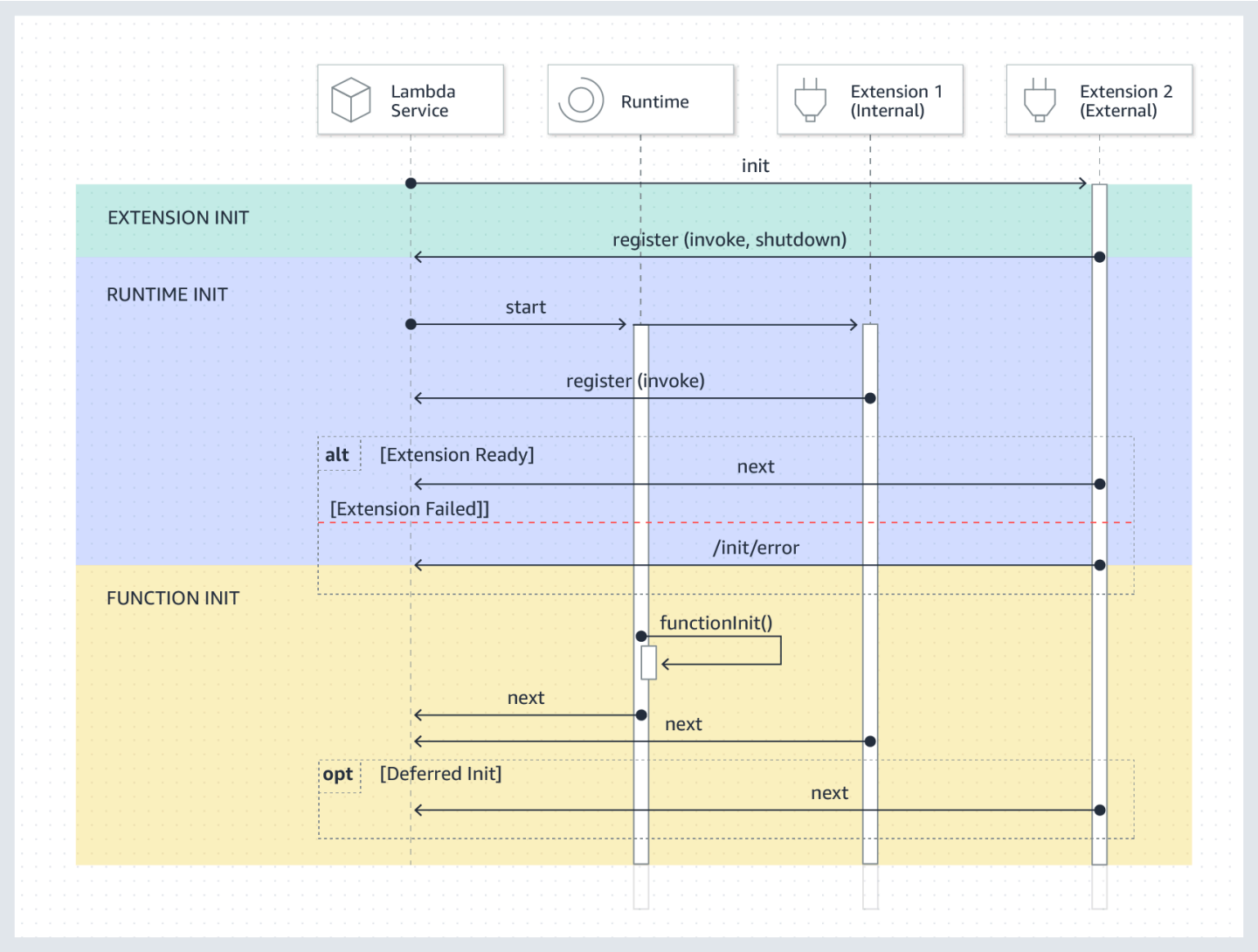
Durante la fase `Extension init`, per ricevere gli eventi ogni estensione deve registrarsi con Lambda. Lambda utilizza il nome file completo dell'estensione per confermare che l'estensione abbia completato la sequenza di bootstrap. Pertanto, ogni chiamata `Register` API deve includere l'intestazione `Lambda-Extension-Name` con il nome file completo dell'estensione.

È possibile registrare fino a 10 estensioni per una funzione. Questo limite viene applicato tramite la chiamata `Register` API.

Dopo la registrazione di ogni estensione, Lambda inizia la fase `Runtime init`. Il processo di runtime chiama `functionInit` per avviare la fase `Function init`.

La fase `Init` viene completata dopo il runtime e ogni estensione registrata indica il completamento inviando una richiesta `Next` API.

> **Note**
>
> Le estensioni possono completare la loro inizializzazione in qualsiasi punto della fase `Init`.



Invoca fase

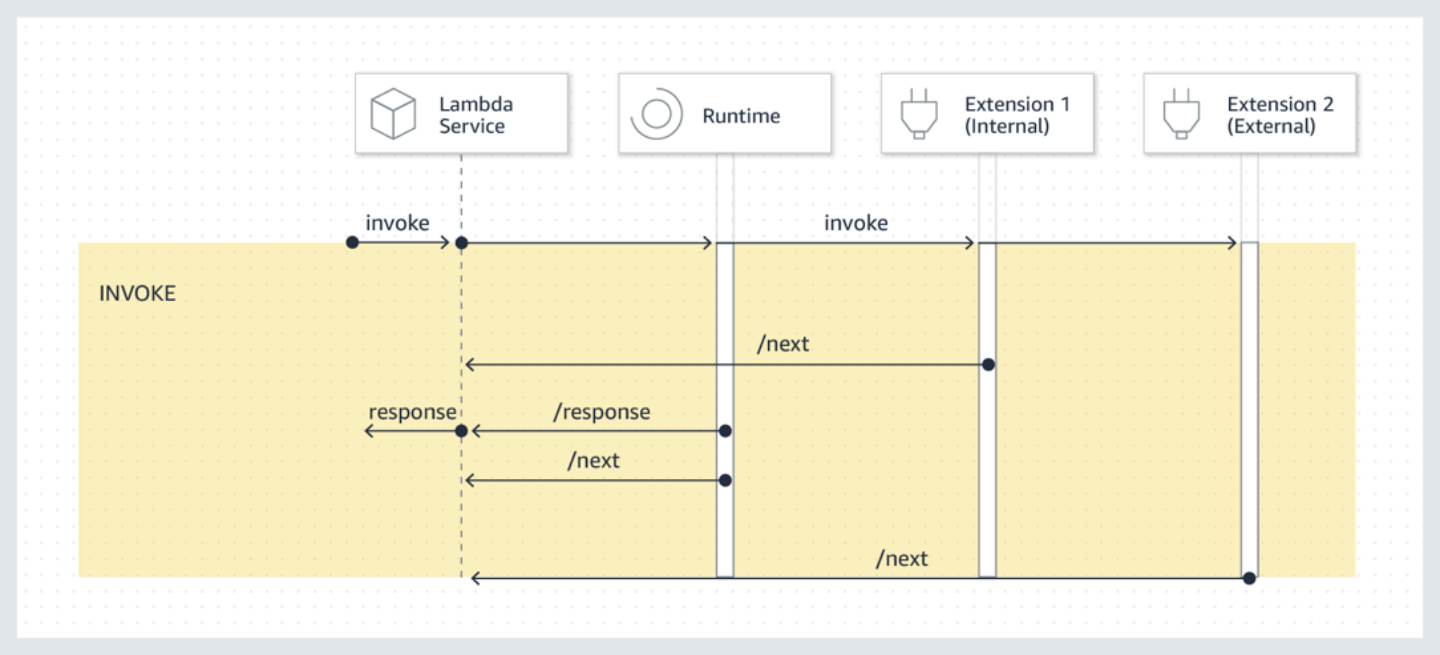
Quando viene richiamata una funzione Lambda in risposta a una richiesta API `Next`, Lambda invia un evento `Invoke` al runtime e a ogni estensione registrata per l'evento `Invoke`.

> **Note**
>
> Istanze gestite Lambda: le funzioni Extensions for Lambda Managed Instances non possono registrarsi per l'evento. `Invoke` Poiché Lambda Managed Instances supporta chiamate simultanee all'interno di un singolo ambiente di esecuzione, l'evento non è supportato. `Invoke` Le estensioni possono registrarsi solo per l'evento. `Shutdown` Se devi tenere traccia dell'inizio e della fine delle chiamate, utilizza l'evento della `platform.report` piattaforma tramite l'API di telemetria.

Durante la chiamata, le estensioni esterne vengono eseguite in parallelo con la funzione. Continuano anche a funzionare dopo che la funzione è stata completata. In questo modo è possibile acquisire informazioni diagnostiche o inviare log, parametri e tracce in una posizione di propria scelta.

Dopo aver ricevuto la risposta della funzione dal runtime, Lambda restituisce la risposta al client, anche se le estensioni sono ancora in esecuzione.

La fase `Invoke` termina dopo il runtime e tutte le estensioni segnalano che vengono eseguite inviando una richiesta `Next` API.



Payload evento: l'evento inviato al runtime (e alla funzione Lambda) contiene l'intera richiesta, le intestazioni (ad esempio `RequestId`) e il payload. L'evento inviato a ciascuna estensione contiene metadati che descrivono il contenuto dell'evento. Questo evento del ciclo di vita include il tipo di evento, l'ora di timeout della funzione (`deadlineMs`), il `requestId`, la funzione richiamata Amazon Resource Name (ARN) e le intestazioni di traccia.

Le estensioni che desiderano accedere al corpo dell'evento funzione possono utilizzare un SDK in runtime che comunica con l'estensione. Gli sviluppatori di funzioni utilizzano l'SDK in runtime per inviare il payload all'estensione quando viene richiamata la funzione.

Ecco un esempio di payload:

```
{
    "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"
    }
 }
```

Duration limit (Limite di durata): l'impostazione di timeout della funzione limita la durata dell'intera fase `Invoke`. Ad esempio, se si imposta il timeout della funzione come 360 secondi, la funzione e tutte le estensioni devono essere completate entro 360 secondi. Si noti che non esiste una fase post-richiamo indipendente. La durata è la somma di tutto il tempo necessario per il completamento del runtime e di tutte le invocazioni delle estensioni e non viene calcolata fino a quando la funzione e tutte le estensioni non hanno terminato l'esecuzione.

Impatto sulle prestazioni e sovraccarico di estensione: le estensioni possono influire sulle prestazioni delle funzioni In qualità di autore dell'estensione, hai il controllo sull'impatto sulle prestazioni della tua estensione. Ad esempio, se una estensione svolge operazioni con uso intensivo dell'elaborazione, la durata della funzione aumenta poiché l'estensione e il codice della funzione condividono le stesse risorse della CPU. Inoltre, se l'estensione esegue operazioni estese al termine dell'invocazione della funzione, la durata della funzione aumenta perché la fase `Invoke` continua fino a quando tutte le estensioni non segnalano che sono state completate.

> **Note**
>
> Lambda alloca la potenza della CPU in proporzione all'impostazione della memoria della funzione. Potresti avere una maggiore durata di esecuzione e inizializzazione a impostazioni di memoria inferiori perché i processi di funzione e estensione sono in concorrenza per le stesse risorse della CPU. Per ridurre la durata di esecuzione e inizializzazione, prova ad aumentare l'impostazione della memoria.

Per aiutare a identificare l'impatto sulle prestazioni introdotto dalle estensioni sulla fase `Invoke`, Lambda esegue l'output del parametro `PostRuntimeExtensionsDuration`. Questa metrica misura il tempo cumulativo trascorso tra la richiesta `Next` API runtime e l'ultima richiesta `Next` API di estensione. Per misurare l'aumento della memoria utilizzata, utilizzare la metrica `MaxMemoryUsed`.

Per ulteriori informazioni sugli stati delle funzioni, consultare Utilizzo dei parametri di CloudWatch Logs con Lambda.

Gli sviluppatori di funzioni possono eseguire diverse versioni delle loro funzioni fianco a fianco per comprendere l'impatto di un'estensione specifica. Si consiglia agli autori di estensioni di pubblicare il consumo previsto di risorse per semplificare la scelta di un'estensione adatta per gli sviluppatori di funzioni.

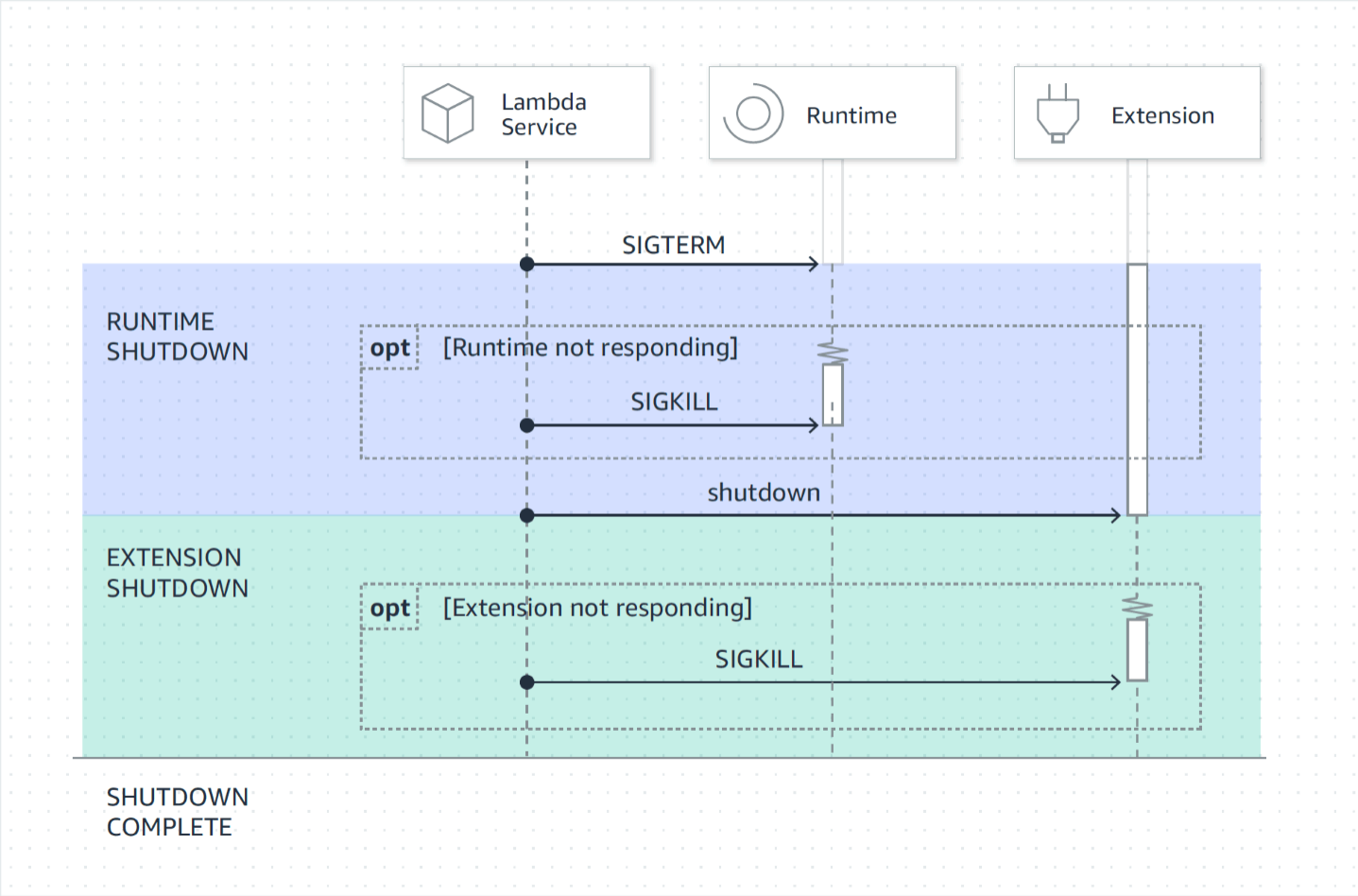
Fase di arresto

Quando Lambda sta per chiudere il runtime, invia uno `Shutdown` a ciascuna estensione esterna registrata. Le estensioni possono utilizzare questo tempo per le attività di pulizia finali. L'evento `Shutdown` viene inviato in risposta a una richiesta `Next` API.

Limite di durata: la durata massima della fase `Shutdown` dipende dalla configurazione delle estensioni registrate:

- 0 ms: funzione senza estensioni registrate
- 500 ms: funzione con estensione interna registrata
- 2.000 ms: funzione con una o più estensioni esterne registrate

Se il runtime o un'estensione non risponde all'evento `Shutdown` entro il limite, Lambda termina il processo utilizzando un segnale `SIGKILL`.



Payload evento: l'evento `Shutdown` contiene il motivo dell'arresto e il tempo rimanente in millisecondi.

`shutdownReason` include quanto segue:

- SPINDOWN – Arresto normale
- TIMEOUT – Timeout del limite di durata
- FAILURE – Condizione di errore, ad esempio un evento `out-of-memory`

```
{
  "eventType": "SHUTDOWN",
  "shutdownReason": "reason for shutdown",
  "deadlineMs": "the time and date that the function times out in Unix time
 milliseconds"
}
```

Autorizzazioni e configurazione

Le estensioni vengono eseguite nello stesso ambiente di esecuzione della funzione Lambda. Inoltre, le estensioni condividono con la funzione risorse quali CPU, memoria e archiviazione `/tmp` su disco. Inoltre, le estensioni utilizzano lo stesso ruolo AWS Identity and Access Management (IAM) e lo stesso contesto di sicurezza della funzione.

Autorizzazioni di accesso al file system e alla rete: le estensioni vengono eseguite nello stesso file system e nello stesso spazio dei nomi dei nomi di rete del runtime della funzione. Ciò significa che le estensioni devono essere compatibili con il sistema operativo associato. Se un'estensione richiede ulteriori regole di traffico di rete in uscita, è necessario applicare tali regole alla configurazione della funzione.

> **Note**
>
> Poiché la directory del codice funzione è di sola lettura, le estensioni non possono modificare il codice della funzione.

Variabili di ambiente: le estensioni possono accedere alle variabili di ambiente, della funzione, ad eccezione delle seguenti variabili specifiche del processo di runtime:

- `AWS_EXECUTION_ENV`
- `AWS_LAMBDA_LOG_GROUP_NAME`
- `AWS_LAMBDA_LOG_STREAM_NAME`
- `AWS_LAMBDA_MAX_CONCURRENCY`(Solo istanze gestite Lambda)
- `AWS_XRAY_CONTEXT_MISSING`
- `AWS_XRAY_DAEMON_ADDRESS`
- `LAMBDA_RUNTIME_DIR`
- `LAMBDA_TASK_ROOT`
- `_AWS_XRAY_DAEMON_ADDRESS`
- `_AWS_XRAY_DAEMON_PORT`
- `_HANDLER`

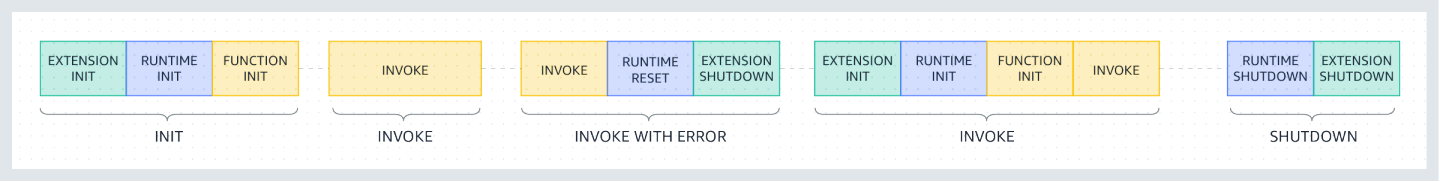
> **Note**
>
> Rilevamento delle istanze gestite Lambda: le estensioni possono controllare `AWS_LAMBDA_INITIALIZATION_TYPE` la variabile di ambiente per determinare se sono in esecuzione su istanze gestite Lambda anziché su funzioni Lambda (predefinite). Questo è il metodo consigliato alle estensioni per adattare il proprio comportamento in base al tipo di ambiente di esecuzione.

Gestione dei guasti

Errori di inizializzazione: se un'estensione genera un errore, Lambda riavvia l'ambiente di esecuzione per imporre un comportamento coerente e per incoraggiare la risposta immediata agli errori per le estensioni. Inoltre, per alcuni clienti, le estensioni devono soddisfare esigenze mission-critical quali registrazione, sicurezza, governance e raccolta di dati di telemetria.

Invocare errori (come memoria esaurita, timeout funzione): poiché le estensioni condividono le risorse con il runtime, sono influenzata dall'esaurimento della memoria. Quando il runtime fallisce, tutte le estensioni e il runtime stesso partecipano alla fase `Shutdown`. Inoltre, il runtime viene riavviato automaticamente come parte dell'invocazione corrente o tramite un meccanismo di reinizializzazione differita.

Se si verifica un errore (ad esempio un timeout della funzione o un errore di runtime) durante `Invoke`, il servizio Lambda esegue un reset. Il reset si comporta come un evento `Shutdown`. Innanzitutto Lambda chiude il runtime, poi invia un evento `Shutdown` a ogni estensione esterna registrata. L'evento include il motivo dell'arresto. Se questo ambiente viene utilizzato per una nuova chiamata, l'estensione e il runtime vengono reinizializzati come parte della chiamata successiva.



Per una spiegazione più approfondita del diagramma precedente, consulta Errori durante la fase di richiamo.

Registri delle estensioni: Lambda invia l'output di registro delle estensioni CloudWatch a Logs. Lambda genera anche un evento di log aggiuntivo per ogni estensione durante `Init`. L'evento di

log registra il nome e la preferenza di registrazione (evento, configurazione) in caso di successo o il motivo dell'errore in caso di errore.

Risoluzione dei problemi delle estensioni

- Se una richiesta `Register` non riesce, assicurarsi che l'intestazione `Lambda-Extension-Name` nella chiamata `Register` API contenga il nome file completo dell'estensione.
- Se la richiesta `Register` non riesce per un'estensione interna, assicurarsi che la richiesta non si registra per l'evento `Shutdown`.

Riferimento all'API delle estensioni

La specifica OpenAPI per le estensioni API versione 2020-01-01 è disponibile qui: extensions-api.zip

È possibile recuperare il valore dell'endpoint API dalla variabile di ambiente `AWS_LAMBDA_RUNTIME_API`. Per inviare una richiesta `Register`, utilizzare il prefisso `2020-01-01/` prima di ogni percorso API. Ad esempio:

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-01-01/extension/register
```

Metodi API

- Registrati
- Next
- Errore di init
- Errore di uscita

Registrati

Durante `Extension init`, per ricevere gli eventi tutte le estensioni devono registrarsi con Lambda. Lambda utilizza il nome file completo dell'estensione per confermare che l'estensione abbia completato la sequenza di bootstrap. Pertanto, ogni chiamata `Register` API deve includere l'intestazione `Lambda-Extension-Name` con il nome file completo dell'estensione.

Le estensioni interne vengono avviate e arrestate dal processo di runtime, pertanto non sono autorizzate a registrarsi per l'evento `Shutdown`.

Percorso – `/extension/register`

Metodo – POST

Intestazioni della richiesta

- `Lambda-Extension-Name` – Il nome file completo dell'estensione. Campo obbligatorio: sì Tipo: stringa
- `Lambda-Extension-Accept-Feature`: si utilizza per specificare le funzionalità opzionali delle estensioni durante la registrazione. Campo obbligatorio: no. Tipo: stringa separata da virgole. Funzionalità disponibili da specificare tramite questa impostazione:
  - `accountId`: se specificato, la risposta alla registrazione dell'estensione conterrà l'ID dell'account associato alla funzione Lambda per cui si sta registrando l'estensione.

Parametri del corpo della richiesta

- `events` – Array degli eventi a cui registrarsi. Campo obbligatorio: no. Tipo: array di stringhe Stringhe valide: `INVOKE`, `SHUTDOWN`.

> **Note**
>
> Istanze gestite Lambda: le funzioni Extensions for Lambda Managed Instances possono registrarsi solo per l'evento. `SHUTDOWN` Il tentativo di registrazione all'`INVOKE`evento genererà un errore. Questo perché Lambda Managed Instances supporta chiamate simultanee all'interno di un unico ambiente di esecuzione.

Intestazioni di risposta

- `Lambda-Extension-Identifier` – Identificativo univoco dell'agente generato (stringa UUID) necessario per tutte le richieste successive.

Codice di risposta

- 200 – Il corpo della risposta contiene il nome della funzione, la versione della funzione e il nome del gestore.
- 400 – Richiesta non valida
- 403 – Non consentito
- 500 – Errore del container. Stato non recuperabile. L'estensione dovrebbe uscire tempestivamente.

Example Esempio di corpo della richiesta

```
{
    'events': [ 'INVOKE', 'SHUTDOWN']
}
```

Example Esempio di corpo della risposta

```
{
    "functionName": "helloWorld",
    "functionVersion": "$LATEST",
    "handler": "lambda_function.lambda_handler"
}
```

Example Esempio di corpo della risposta con funzione accountId facoltativa

```
{
    "functionName": "helloWorld",
    "functionVersion": "$LATEST",
    "handler": "lambda_function.lambda_handler",
    "accountId": "123456789012"
}
```

Next

Le estensioni inviano una richiesta `Next` API per ricevere l'evento successivo, che può essere un evento `Invoke` o un evento `Shutdown`. Il corpo della risposta contiene il payload, che è un documento JSON che contiene i dati degli eventi.

L'estensione invia una richiesta `Next` API per segnalare che è pronta per ricevere nuovi eventi. Questa è una chiamata di blocco.

Non impostare un timeout sulla chiamata GET, poiché l'estensione può essere sospesa per un periodo di tempo fino a quando non c'è un evento da restituire.

Percorso – `/extension/event/next`

Metodo – GET

Intestazioni della richiesta

- `Lambda-Extension-Identifier` – Identificativo univoco dell'estensione (stringa UUID). Campo obbligatorio: sì Tipo: stringa UUID.

Intestazioni di risposta

- `Lambda-Extension-Event-Identifier`: identificatore univoco per l'evento (stringa UUID).

Codice di risposta

- 200 – La risposta contiene informazioni sull'evento successivo (`EventInvoke` o `EventShutdown`).
- 403 – Non consentito
- 500 – Errore del container. Stato non recuperabile. L'estensione dovrebbe uscire tempestivamente.

Errore di init

L'estensione utilizza questo metodo per segnalare un errore di inizializzazione a Lambda. Chiamalo quando l'estensione non riesce a inizializzare dopo che si è registrata. Dopo che Lambda riceve l'errore, le chiamate API successive hanno esito negativo. L'estensione dovrebbe terminare dopo aver ricevuto la risposta da Lambda.

Percorso – `/extension/init/error`

Metodo – POST

Intestazioni della richiesta

- `Lambda-Extension-Identifier` – Identificativo univoco dell'estensione. Campo obbligatorio: sì Tipo: stringa UUID.
- `Lambda-Extension-Function-Error-Type` – Tipo di errore rilevato dall'estensione. Campo obbligatorio: sì L'intestazione è costituita da un valore stringa. Lambda accetta qualsiasi stringa, ma si consiglia di utilizzare il formato <categoria.motivo>. Esempio:
  - Estensione. NoSuchHandler
  - Estensione. APIKeyNotFound
  - Estensione. ConfigInvalid

- Estensione. UnknownReason

Parametri del corpo della richiesta

- `ErrorRequest` – Informazioni sull'errore. Campo obbligatorio: no.

Questo campo è un oggetto JSON con la seguente struttura:

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

NB: Lambda accetta qualsiasi valore per `errorType`.

Nell'esempio seguente viene mostrato un messaggio di errore della funzione Lambda in cui la funzione non è stata in grado di analizzare i dati evento forniti nell'invocazione.

Example Errore di funzione

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Codice di risposta

- 202 – Accettato
- 400 – Richiesta non valida
- 403 – Non consentito
- 500 – Errore del container. Stato non recuperabile. L'estensione dovrebbe uscire tempestivamente.

Errore di uscita

L'estensione utilizza questo metodo per segnalare un errore a Lambda prima di uscire. Chiamalo quando incontri un errore inaspettato. Dopo che Lambda riceve l'errore, le chiamate API successive hanno esito negativo. L'estensione dovrebbe terminare dopo aver ricevuto la risposta da Lambda.

Percorso – `/extension/exit/error`

Metodo – POST

Intestazioni della richiesta

- `Lambda-Extension-Identifier` – Identificativo univoco dell'estensione. Campo obbligatorio: sì Tipo: stringa UUID.
- `Lambda-Extension-Function-Error-Type` – Tipo di errore rilevato dall'estensione. Campo obbligatorio: sì L'intestazione è costituita da un valore stringa. Lambda accetta qualsiasi stringa, ma si consiglia di utilizzare il formato <categoria.motivo>. Esempio:
  - Estensione. NoSuchHandler
  - Estensione. APIKeyNotFound
  - Estensione. ConfigInvalid
  - Estensione. UnknownReason

Parametri del corpo della richiesta

- `ErrorRequest` – Informazioni sull'errore. Campo obbligatorio: no.

Questo campo è un oggetto JSON con la seguente struttura:

```
{
      errorMessage: string (text description of the error),
      errorType: string,
      stackTrace: array of strings
}
```

NB: Lambda accetta qualsiasi valore per `errorType`.

Nell'esempio seguente viene mostrato un messaggio di errore della funzione Lambda in cui la funzione non è stata in grado di analizzare i dati evento forniti nell'invocazione.

Example Errore di funzione

```
{
      "errorMessage" : "Error parsing event data.",
      "errorType" : "InvalidEventDataException",
      "stackTrace": [ ]
}
```

Codice di risposta

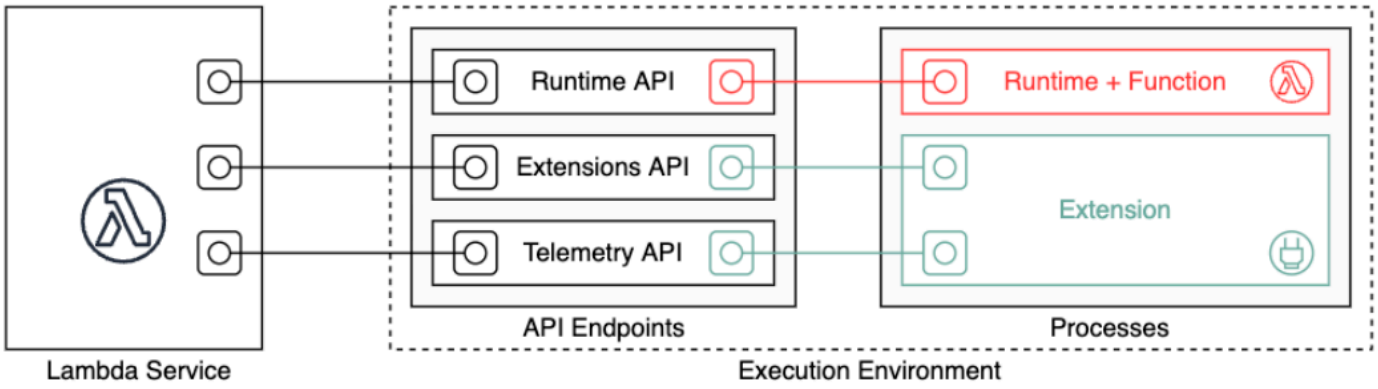
- 202 – Accettato
- 400 – Richiesta non valida
- 403 – Non consentito
- 500 – Errore del container. Stato non recuperabile. L'estensione dovrebbe uscire tempestivamente.

Accesso ai dati di telemetria in tempo reale per le estensioni tramite l'API Telemetry

L'API Telemetry consente alle estensioni di ricevere dati di telemetria direttamente da Lambda. Durante l'inizializzazione e l'invocazione, Lambda acquisisce in automatico i dati di telemetria, tra cui log, parametri della piattaforma e tracce della piattaforma. L'API Telemetry consente alle estensioni di accedere a tali dati di telemetria direttamente da Lambda quasi in tempo reale.

All'interno dell'ambiente di esecuzione Lambda, puoi abbonare le estensioni Lambda ai flussi di telemetria. Dopo l'abbonamento, Lambda invia automaticamente tutti i dati di telemetria alle estensioni. Avrai quindi la flessibilità necessaria per elaborare, filtrare e inviare i dati alla tua destinazione preferita, come un bucket Amazon Simple Storage Service (Amazon S3) o un fornitore di strumenti di osservabilità di terze parti.

Il diagramma seguente mostra come l'API Extensions e l'API Telemetry connettono le estensioni a Lambda dall'interno dell'ambiente di esecuzione. Inoltre L'API Runtime collega il runtime e la funzione a Lambda.



> **Important**
>
> L'API di telemetria Lambda sostituisce l'API Lambda Logs. Sebbene l'API Logs rimanga completamente funzionante, in futuro consigliamo di utilizzare solo l'API di telemetria. Puoi iscrivere la tua estensione a un flusso di telemetria utilizzando l'API di telemetria o l'API Logs. Dopo la sottoscrizione utilizzando uno di questi APIs, qualsiasi tentativo di sottoscrizione utilizzando l'altra API restituisce un errore.

Requisiti della versione dello schema Lambda Managed Instances

Le istanze gestite Lambda supportano solo la versione `2025-01-29` dello schema dell'API di telemetria. Quando ti iscrivi ai flussi di telemetria per le funzioni Managed Instance, devi utilizzarli nella richiesta di abbonamento. `"schemaVersion": "2025-01-29"` L'utilizzo di versioni precedenti dello schema comporterà il rifiuto degli eventi da parte di Lambda.
La versione `2025-01-29` dello schema è retrocompatibile e può essere utilizzata sia con le istanze gestite Lambda che con le funzioni Lambda (predefinite). Consigliamo di utilizzare questa versione per tutte le nuove estensioni per garantire la compatibilità tra entrambi i modelli di distribuzione.

Le estensioni possono utilizzare l'API di telemetria per sottoscrivere i tre diversi flussi di telemetria.

- Telemetria della piattaforma: registri, metriche e tracce, che descrivono eventi ed errori relativi al ciclo di vita del runtime dell'ambiente di esecuzione, al ciclo di vita delle estensioni e alle chiamate delle funzioni.
- Log delle funzioni: log personalizzati generati dal codice della funzione Lambda.
- Log di estensione: i log personalizzati generati dal codice di estensione Lambda.

Note

Lambda invia log e metriche e tracce a X-Ray (se hai attivato il tracciamento) CloudWatch, anche se un'estensione si abbona ai flussi di telemetria.

Sections

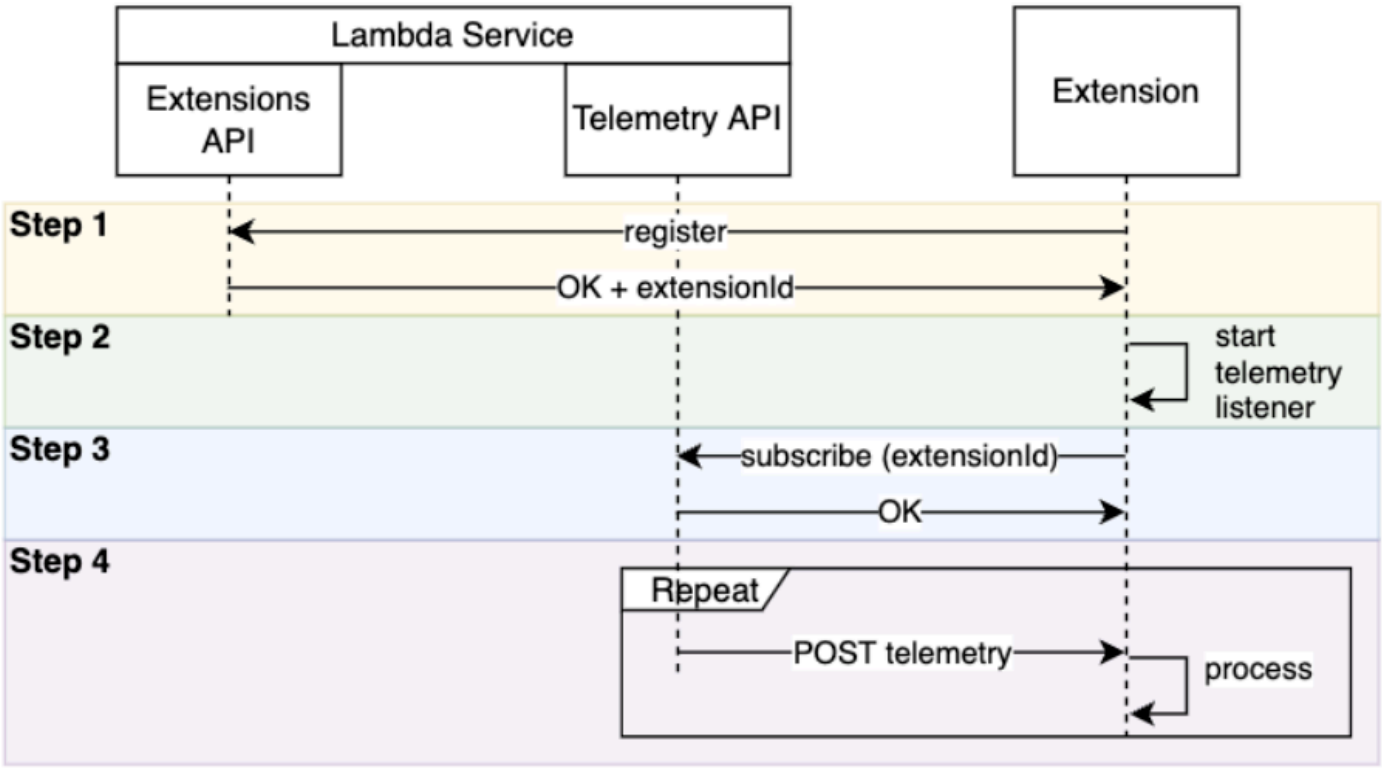
- Creazione di estensioni tramite l'API di telemetria
- Registrazione delle estensioni
- Creazione di un listener di telemetria
- Specifica di un protocollo di destinazione
- Configurazione dell'utilizzo della memoria e del buffering
- Invio di una richiesta di sottoscrizione all'API di telemetria
- Messaggi dell'API di telemetria in entrata
- Riferimento all'API di telemetria Lambda

- Riferimento allo schema Event dell'API di telemetria Lambda
- Conversione degli oggetti Event dell'API di telemetria Lambda in intervalli OpenTelemetry
- Utilizzo dell'API Logs di Lambda

Creazione di estensioni tramite l'API di telemetria

Le estensioni Lambda vengono eseguite come processi indipendenti nell'ambiente di esecuzione. L'esecuzione delle estensioni può proseguire dopo il completamento dell'invocazione della funzione. Poiché le estensioni vengono eseguite come processi separati, è possibile scriverle in un linguaggio diverso da quello del codice della funzione. Consigliamo di scrivere estensioni utilizzando un linguaggio compilato come Golang o Rust. In questo caso, l'estensione è un binario autonomo compatibile con qualsiasi runtime supportato.

Il diagramma seguente illustra un processo in quattro fasi per creare un'estensione che riceve ed elabora i dati di telemetria tramite l'API di telemetria.



Ecco ogni fase in modo più dettagliato:

1. Registra la tua estensione utilizzando l'the section called “API estensioni”. Questo ti fornisce un `Lambda-Extension-Identifier`, di cui avrai bisogno nei passaggi seguenti. Per ulteriori

informazioni su come eseguire la registrazione dell'estensione, consulta the section called “Registrazione delle estensioni”.

2. Crea un listener di telemetria. Può trattarsi di un server HTTP o TCP di base. Lambda utilizza l'URI del listener di telemetria per inviare dati di telemetria all'estensione. Per ulteriori informazioni, consulta the section called “Creazione di un listener di telemetria”.
3. Utilizzando l'API Subscribe nell'API Telemetry, esegui l'abbonamento all'estensione ai flussi di telemetria desiderati. Avrai bisogno dell'URI del tuo listener di telemetria per questo passaggio. Per ulteriori informazioni, consulta the section called “Invio di una richiesta di sottoscrizione all'API di telemetria”.
4. Ottieni dati di telemetria da Lambda tramite l'listener di telemetria. Puoi eseguire qualsiasi elaborazione personalizzata di questi dati, ad esempio inviandoli ad Amazon S3 o a un servizio di osservabilità esterno.

> **Note**
>
> L'ambiente di esecuzione di una funzione Lambda può avviarsi e interrompersi più volte nell'ambito del suo ciclo di vita. In generale, il codice dell'estensione viene eseguito durante le chiamate delle funzioni e anche fino a 2 secondi durante la fase di spegnimento. Consigliamo di raggruppare i dati di telemetria non appena arrivano all'listener. Quindi, utilizza gli eventi del ciclo di vita `Invoke` e `Shutdown` per inviare ogni batch alle destinazioni desiderate.

Registrazione delle estensioni

Prima di poter eseguire l'abbonamento ai dati di telemetria, devi registrare l'estensione Lambda. La registrazione avviene durante la fase di inizializzazione dell'estensione. L'esempio seguente mostra una richiesta HTTP per la registrazione di un'estensione.

```
POST http://${AWS_LAMBDA_RUNTIME_API}/2020-01-01/extension/register
 Lambda-Extension-Name: lambda_extension_name
{
    'events': [ 'INVOKE', 'SHUTDOWN']
}
```

Se la richiesta ha esito positivo, il sottoscrittore riceve una risposta riuscita HTTP 200. L'intestazione della risposta contiene l'`Lambda-Extension-Identifier`. Il corpo della risposta contiene altre proprietà della funzione.

```
HTTP/1.1 200 OK
Lambda-Extension-Identifier: a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
{
    "functionName": "lambda_function",
    "functionVersion": "$LATEST",
    "handler": "lambda_handler",
    "accountId": "123456789012"
}
```

Per ulteriori informazioni, consulta the section called “Riferimento all'API delle estensioni”.

Creazione di un listener di telemetria

L'estensione Lambda deve disporre di un listener che gestisca le richieste in entrata dall'API di telemetria. Il codice seguente mostra un esempio di implementazione dell'listener di telemetria in 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
}
```

Specifica di un protocollo di destinazione

Quando esegui la sottoscrizione per ricevere la telemetria tramite l'API di telemetria, puoi specificare un protocollo di destinazione in aggiunta all'URI di destinazione:

```
{
    "destination": {
        "protocol": "HTTP",
        "URI": "http://sandbox.localdomain:8080"
    }
}
```

Lambda accetta due protocolli per la ricezione della telemetria:

- HTTP (consigliato): Lambda consegna la telemetria a un endpoint HTTP locale (`http://sandbox.localdomain:${PORT}/${PATH}`) come matrice di record in formato JSON. Il parametro `$PATH` è facoltativo. Lambda supporta solo HTTP, non HTTPS. Lambda fornisce la telemetria tramite richieste POST.

- TCP: Lambda consegna la telemetria a una porta TCP in formato JSON delimitato da Newline (NDJSON).

> **Note**
>
> Si consiglia di utilizzare HTTP anziché TCP. Con TCP, la piattaforma Lambda non può riconoscere che la telemetria viene consegnata al livello dell'applicazione. Pertanto, se l'estensione si blocca si potrebbero perdere i dati di telemetria. HTTP non condivide questa limitazione.

Prima di eseguire l'abbonamento per i ricevere i dati di telemetria, definisci l'listener HTTP locale o la porta TCP. Durante l'installazione, tenere presente quanto segue:

- Lambda invia la telemetria solo alle destinazioni che si trovano all'interno dell'ambiente di esecuzione.
- Lambda riprova a inviare i dati di telemetria (con backoff) in assenza di un listener o se la richiesta POST subisce un errore. Se riporta un arresto anomalo, l'listener di telemetria continuerà a riceverla dopo che Lambda avrà riavviato l'ambiente di esecuzione.
- Lambda riserva la porta 9001. Non ci sono altre restrizioni o raccomandazioni sul numero di porta.

Configurazione dell'utilizzo della memoria e del buffering

L'utilizzo della memoria in un ambiente di esecuzione aumenta linearmente con il numero di abbonati. Gli abbonamenti consumano risorse di memoria, perché ognuno apre un nuovo buffer di memoria per archiviare i dati di telemetria. L'utilizzo della memoria buffer contribuisce al consumo di memoria complessivo nell'ambiente di esecuzione.

Quando effettui l'abbonamento per ricevere la telemetria attraverso l'API Telemetry, hai la possibilità di eseguire il buffer dei dati di telemetria e consegnarli agli abbonati in batch. Per ottimizzare l'utilizzo della memoria, puoi specificare una configurazione di buffering:

```
{
    "buffering": {
        "maxBytes": 256*1024,
        "maxItems": 1000,
        "timeoutMs": 100
```

```
    }
}
```

Parametro	Description	Valori predefiniti e limiti
`maxBytes`	Il volume massimo di dati di telemetria (in byte) da memorizzare nel buffer della memoria.	Predefinito: 262.144. Minimo: 262.144. Massimo: 1.048.576.
`maxItems`	Il numero massimo di eventi da memorizzare nel buffer.	Predefinito: 10.000 Minimo: 1.000 Massimo: 10.000.
`timeoutMs`	Il tempo massimo (in millisecondi) per memorizzare nel buffer un batch.	Predefinito: 1.000 Minimo: 25 Massimo: 30.000

Quando configuri il buffering, tieni presente i seguenti punti:

- Se uno qualsiasi dei flussi di input è chiuso, Lambda svuota i log. Ad esempio, ciò si può verificare quando il runtime subisce un arresto anomalo.
- Ogni abbonato può personalizzare la configurazione di buffering nella richiesta di abbonamento.
- Al momento di determinare la dimensione di buffer per la lettura dei dati, prevedi di ricevere payload di dimensioni pari a `2 * maxBytes + metadataBytes`, dove `maxBytes` è un componente della configurazione di buffering. Per valutare la quantità di `metadataBytes` da considerare, esamina i seguenti metadati. Lambda allega metadati simili a questi a ogni record:

```
{
   "time": "2022-08-20T12:31:32.123Z",
   "type": "function",
   "record": "Hello World"
}
```

- Se il sottoscrittore non è in grado di elaborare la telemetria in ingresso abbastanza rapidamente o se il codice di funzione genera un volume di log molto elevato, Lambda potrebbe eliminare i registri per mantenere limitato l'utilizzo della memoria. Quando ciò si verifica, Lambda invia un evento `platform.logsDropped`.

Invio di una richiesta di sottoscrizione all'API di telemetria

Un'estensione Lambda può eseguire la sottoscrizione per ricevere i dati di telemetria inviando una richiesta di sottoscrizione all'API di telemetria. La richiesta di sottoscrizione deve contenere informazioni sui tipi di eventi a cui si desidera che l'estensione si iscriva. Inoltre, la richiesta può contenere informazioni sulla destinazione della consegna e una configurazione del buffering.

Prima di inviare una richiesta di sottoscrizione, è necessario disporre di un ID di estensione (`Lambda-Extension-Identifier`). Quando registri l'estensione con l'API delle estensioni, ottieni un ID di estensione dalla risposta dell'API.

La registrazione avviene durante la fase di inizializzazione dell'estensione. L'esempio seguente mostra una richiesta HTTP di sottoscrizione a tutti e tre i flussi di telemetria: telemetria della piattaforma, log delle funzioni e log delle estensioni.

```
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"
   }
}
```

Se la richiesta ha esito positivo, il sottoscrittore riceve una risposta do operazione riuscita HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

Messaggi dell'API di telemetria in entrata

Dopo l'abbonamento con l'API Telemetry, un'estensione comincia in automatico a ricevere i dati di telemetria da Lambda attraverso richieste POST. Ogni corpo di richiesta POST contiene una serie di oggetti `Event`. Ogni `Event` presenta il seguente schema:

```
{
   time: String,
   type: String,
   record: Object
}
```

- La proprietà `time` definisce il momento in cui la piattaforma Lambda ha generato l'evento. Si tratta di un valore diverso da quando l'evento si è verificato effettivamente. Il valore della stringa di `time` è un timestamp nel formato ISO 8601.
- La proprietà `type` definisce il tipo di evento. La seguente tabella riporta tutti i valori possibili.
- La proprietà `record` definisce un oggetto JSON che contiene i dati di telemetria. Lo schema di questo oggetto JSON dipende dal `type`.

> **Ordinamento degli eventi con invocazioni simultanee**
>
> Per le istanze gestite Lambda, le chiamate di più funzioni possono essere eseguite contemporaneamente all'interno dello stesso ambiente di esecuzione. In questo caso, l'ordine degli `platform.start` `platform.report` eventi non è garantito tra diverse chiamate simultanee. Le estensioni devono gestire gli eventi di più chiamate in esecuzione in parallelo e non devono presupporre un ordinamento sequenziale.
> Per attribuire correttamente gli eventi a chiamate specifiche, le estensioni devono utilizzare il `requestId` campo presente in questi eventi della piattaforma. Ogni chiamata ha un ID di richiesta univoco che rimane coerente in tutti gli eventi di quella chiamata, consentendo alle estensioni di correlare correttamente gli eventi anche quando arrivano fuori ordine.

La tabella seguente riassume tutti i tipi di oggetti `Event` e i collegamenti al riferimento dello schema `Event` dell'API di telemetria per ogni tipo di evento.

Categoria	Tipo di evento	Description	Schema dei registri di eventi
Evento della piattaforma	`platform.initStart`	Inizializzazione della funzione avviata.	Schema the section called “`platform.initStart`”
Evento della piattaforma	`platform.initRuntimeDone`	Inizializzazione della funzione completata.	Schema the section called “`platform.initRuntimeDone`”
Evento della piattaforma	`platform.initReport`	Un rapporto sull'inizializzazione della funzione.	Schema the section called “`platform.initReport`”
Evento della piattaforma	`platform.start`	L'invocazione della funzione è stata avviata.	Schema the section called “`platform.start`”
Evento della piattaforma	`platform.runtimeDone`	Il runtime ha terminato l'elaborazione di un evento con esito positivo o negativo.	Schema the section called “`platform.runtimeDone`”
Evento della piattaforma	`platform.report`	Un rapporto sull'invocazione di una funzione.	Schema the section called “`platform.report`”
Evento della piattaforma	`platform.restoreStart`	Il ripristino del runtime è iniziato.	Schema the section called “`platform.restoreStart`”
Evento della piattaforma	`platform.restoreRuntimeDone`	Il ripristino del runtime è stato completato.	Schema the section called “`platform.restoreRuntimeDone`”

Categoria	Tipo di evento	Description	Schema dei registri di eventi
Evento della piattaforma	`platform.restoreReport`	Rapporto di ripristino del runtime.	Schema the section called “`platform.restoreReport`”
Evento della piattaforma	`platform.telemetrySubscription`	L'estensione che ha eseguito la sottoscrizione all'API di telemetria.	Schema the section called “`platform.telemetrySubscription`”
Evento della piattaforma	`platform.logsDropped`	Lambda ha eliminato le voci del log.	Schema the section called “`platform.logsDropped`”
Log delle funzioni	`function`	Una riga del log dal codice della funzione.	Schema the section called “`function`”
Log di estensioni	`extension`	Una riga di log dal codice dell'estensione.	Schema the section called “`extension`”

Riferimento all'API di telemetria Lambda

Utilizza l'endpoint dell'API di telemetria Lambda per sottoscrivere le estensioni ai flussi di telemetria. È possibile recuperare l'endpoint dell'API di telemetria dalla variabile di ambiente `AWS_LAMBDA_RUNTIME_API`. Per inviare una richiesta API, aggiungi la versione dell'API (`2022-07-01/`) e `telemetry/`. Esempio:

```
http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry/
```

Per la definizione della specifica OpenAPI (OAS) della versione delle risposte alla sottoscrizione `2025-01-29`, consulta quanto segue:

- HTTP — telemetry-api-http-schema.zip
- TCP — .zip telemetry-api-tcp-schema

operazioni API

- Subscribe

Subscribe

Per sottoscrivere un flusso di telemetria, un'estensione Lambda può inviare una richiesta API Subscribe.

- Percorso – `/telemetry`
- Metodo: PUT
- Headers
  - `Content-Type`: `application/json`
- Parametri del corpo della richiesta
  - schemaVersion
    - Obbligatorio: sì
    - Tipo: String
    - Valori validi: `"2025-01-29"`, `"2022-12-13"` o `"2022-07-01"`
    - Nota: le istanze gestite Lambda richiedono. `"2025-01-29"` Questa versione è retrocompatibile con le funzioni Lambda (predefinite).

- destinazione: le impostazioni di configurazione che definiscono la destinazione dell'evento di telemetria e il protocollo per la consegna degli eventi.
  - Obbligatorio: sì
  - Tipo: oggetto

```
{
    "protocol": "HTTP",
    "URI": "http://sandbox.localdomain:8080"
}
```

  - protocollo: il protocollo utilizzato da Lambda per inviare dati di telemetria.
    - Obbligatorio: sì
    - Tipo: String
    - Valori validi: `"HTTP"|"TCP"`
  - URI: l'URI a cui inviare i dati di telemetria.
    - Obbligatorio: sì
    - Tipo: String
  - Per ulteriori informazioni, consulta the section called “Specifica di un protocollo di destinazione”.
- tipi: i tipi di dati di telemetria che desideri siano sottoscritti dall'estensione.
  - Obbligatorio: sì
  - Tipo: array di stringhe
  - Valori validi: `"platform"|"function"|"extension"`
- buffering: le impostazioni di configurazione per il buffering degli eventi.
  - Obbligatorio: no
  - Tipo: oggetto

```
{
   "buffering": {
        "maxItems": 1000,
        "maxBytes": 256*1024,
        "timeoutMs": 100
   }
}
```


  - maxItems – Il numero massimo di eventi da memorizzare nel buffer.

    - Obbligatorio: no
    - Tipo: numero intero
    - Predefinito: 1.000
    - Minimo: 1.000
    - Massimo: 10.000.
- maxBytes: il volume massimo di dati di telemetria (in byte) da memorizzare nel buffer della memoria.
    - Obbligatorio: no
    - Tipo: numero intero
    - Predefinito: 262.144.
    - Minimo: 262.144.
    - Massimo: 1.048.576.
- timeoutMs – Il tempo massimo (in millisecondi) per il buffer di un batch.
    - Obbligatorio: no
    - Tipo: numero intero
    - Predefinito: 1.000
    - Minimo: 25
    - Massimo: 30.000
- Per ulteriori informazioni, consulta the section called “Configurazione dell'utilizzo della memoria e del buffering”.

Esempio di richiesta API Subscribe

```
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"
    }
}
```

Se la richiesta Subscribe ha esito positivo, il sottoscrittore riceve una risposta di operazione riuscita HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

Se la richiesta non riesce, l'estensione riceve una risposta di errore. Esempio:

```
HTTP/1.1 400 OK
{
    "errorType": "ValidationError",
    "errorMessage": "URI port is not provided; types should not be empty"
}
```

Ecco alcuni codici di risposta aggiuntivi che l'estensione può ricevere:

- 200 – Richiesta completata con successo
- 202 – Richiesta accettata. Risposta alla richiesta di sottoscrizione nell'ambiente di test locale
- 400: richiesta non valida
- 500 – Errore servizio

Riferimento allo schema **`Event`** dell'API di telemetria Lambda

Utilizza l'endpoint dell'API di telemetria Lambda per sottoscrivere le estensioni ai flussi di telemetria. È possibile recuperare l'endpoint dell'API di telemetria dalla variabile di ambiente `AWS_LAMBDA_RUNTIME_API`. Per inviare una richiesta API, aggiungi la versione dell'API (`2022-07-01/`) e `telemetry/`. Esempio:

```
http://${AWS_LAMBDA_RUNTIME_API}/2022-07-01/telemetry/
```

Per la definizione della specifica OpenAPI (OAS) della versione delle risposte alla sottoscrizione `2025-01-29`, consulta quanto segue:

- HTTP — telemetry-api-http-schema.zip
- TCP — .zip telemetry-api-tcp-schema

La tabella seguente è un riepilogo di tutti i tipi di oggetti `Event` supportati dall'API di telemetria.

Categoria	Tipo di evento	Description	Schema dei registri di eventi
Evento della piattaforma	`platform.initStart`	Inizializzazione della funzione avviata.	Schema the section called “`platform.initStart`”
Evento della piattaforma	`platform.initRuntimeDone`	Inizializzazione della funzione completata.	Schema the section called “`platform.initRuntimeDone`”
Evento della piattaforma	`platform.initReport`	Un rapporto sull'inizializzazione della funzione.	Schema the section called “`platform.initReport`”
Evento della piattaforma	`platform.start`	L'invocazione della funzione è stata avviata.	Schema the section called “`platform.start`”

Categoria	Tipo di evento	Description	Schema dei registri di eventi
Evento della piattaforma	`platform.runtimeDone`	Il runtime ha terminato l'elaborazione di un evento con esito positivo o negativo.	Schema the section called “`platform.runtimeDone`”
Evento della piattaforma	`platform.report`	Un rapporto sull'invocazione di una funzione.	Schema the section called “`platform.report`”
Evento della piattaforma	`platform.restoreStart`	Il ripristino del runtime è iniziato.	Schema the section called “`platform.restoreStart`”
Evento della piattaforma	`platform.restoreRuntimeDone`	Il ripristino del runtime è stato completato.	Schema the section called “`platform.restoreRuntimeDone`”
Evento della piattaforma	`platform.restoreReport`	Rapporto di ripristino del runtime.	Schema the section called “`platform.restoreReport`”
Evento della piattaforma	`platform.telemetrySubscription`	L'estensione che ha eseguito la sottoscrizione all'API di telemetria.	Schema the section called “`platform.telemetrySubscription`”
Evento della piattaforma	`platform.logsDropped`	Lambda ha eliminato le voci del log.	Schema the section called “`platform.logsDropped`”
Log delle funzioni	`function`	Una riga del log dal codice della funzione.	Schema the section called “`function`”

Categoria	Tipo di evento	Description	Schema dei registri di eventi
Log di estensioni	`extension`	Una riga di log dal codice dell'estensione.	Schema the section called “`extension`”

Indice

- Tipi di oggetti Event dell'API di telemetria
  - platform.initStart
  - platform.initRuntimeDone
  - platform.initReport
  - platform.start
  - platform.runtimeDone
  - platform.report
  - platform.restoreStart
  - platform.restoreRuntimeDone
  - platform.restoreReport
  - platform.extension
  - platform.telemetrySubscription
  - platform.logsDropped
  - function
  - extension
- Tipi di oggetti condivisi
  - InitPhase
  - InitReportMetrics
  - InitType
  - ReportMetrics
  - RestoreReportMetrics
  - RuntimeDoneMetrics
  - Span

- Status
- TraceContext
- TracingType

Tipi di oggetti **`Event`** dell'API di telemetria

Questa sezione descrive in dettaglio i tipi di oggetti `Event` supportati dall'API di telemetria Lambda. Nelle descrizioni degli eventi, un punto interrogativo (?) indica che l'attributo potrebbe non essere presente nell'oggetto.

`platform.initStart`

Un evento `platform.initStart` indica che la fase di inizializzazione della funzione è stata avviata. Un oggetto `platform.initStart` `Event` ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.initStart
- record: PlatformInitStart
```

L'oggetto `PlatformInitStart` ha i seguenti attributi:

- functionName: `String`
- functionVersion: `String`
- initializationType: oggetto `the section called “InitType”`
- instanceId?: `String`
- instanceMaxMemory? – `Integer`
- phase: oggetto `the section called “InitPhase”`
- runtimeVersion?: `String`
- runtimeVersionArn? – `String`

Di seguito è riportato un esempio `Event` di tipo `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.initRuntimeDone`

Un evento `platform.initRuntimeDone` indica che la fase di inizializzazione della funzione è stata completata. Un oggetto `platform.initRuntimeDone` `Event` ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.initRuntimeDone
- record: PlatformInitRuntimeDone
```

L'oggetto `PlatformInitRuntimeDone` ha i seguenti attributi:

- initializationType: oggetto the section called “InitType”
- phase: oggetto the section called “InitPhase”
- status: oggetto the section called “Status”
- spans? : elenco di oggetti the section called “Span”

Di seguito è riportato un esempio `Event` di tipo `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`

Un evento `platform.initReport` contiene un rapporto generale della fase di inizializzazione della funzione. Un oggetto `platform.initReport` `Event` ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.initReport
- record: PlatformInitReport
```

L'oggetto `PlatformInitReport` ha i seguenti attributi:

- errorType?: stringa
- initializationType: oggetto the section called “InitType”
- phase: oggetto the section called “InitPhase”
- metrics: oggetto the section called “InitReportMetrics”
- spans? : elenco di oggetti the section called “Span”
- status: oggetto the section called “Status”

Di seguito è riportato un esempio `Event` di tipo `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`

Un evento `platform.start` indica che la fase di invocazione della funzione è stata avviata. Un oggetto `platform.start` `Event` ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.start
- record: PlatformStart
```

L'oggetto `PlatformStart` ha i seguenti attributi:

- requestId: `String`
- version?: `String`
- tracing?: the section called “TraceContext”

Di seguito è riportato un esempio `Event` di tipo `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`

Un evento `platform.runtimeDone` indica che la fase di invocazione della funzione è stata completata. Un oggetto `platform.runtimeDone` `Event` ha la seguente forma:

> **Istanze gestite Lambda**
>
> L'`platform.runtimeDone`evento non è supportato per le istanze gestite Lambda. Le estensioni in esecuzione su istanze gestite non riceveranno questo evento perché le estensioni non possono sottoscrivere l'`INVOKE`evento sulle istanze gestite. A causa del modello di esecuzione simultanea in cui è possibile elaborare più chiamate contemporaneamente, le estensioni non possono eseguire l'elaborazione post-invoca per le singole chiamate come avviene tradizionalmente con le funzioni Lambda (predefinite). Per le istanze gestite, gli intervalli `responseLatency` e `responseDuration` gli intervalli normalmente inclusi sono invece disponibili nell'evento. `platform.runtimeDone` `platform.report` Per informazioni dettagliate, vedi the section called “`platform.report`”.

```
Event: Object
- time: String
- type: String = platform.runtimeDone
- record: PlatformRuntimeDone
```

L'oggetto `PlatformRuntimeDone` ha i seguenti attributi:

- errorType?: `String`
- metrics?: oggetto the section called “RuntimeDoneMetrics”
- requestId: `String`
- status: oggetto the section called “Status”
- spans? : elenco di oggetti the section called “Span”
- tracing?: oggetto the section called “TraceContext”

Di seguito è riportato un esempio `Event` di tipo `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`

Un evento `platform.report` contiene un rapporto generale della fase di invocazione della funzione. Un oggetto `platform.report` `Event` ha la seguente forma:

> **Istanze gestite Lambda**
>
> L'`platform.report`evento per Lambda Managed Instances ha metriche e intervalli diversi rispetto alle funzioni Lambda (predefinite). Per le istanze gestite:
>
> - Spans: contiene `responseLatency` e `responseDuration` invece di. `extensionOverhead` Lo `extensionOverhead` span non è disponibile perché le estensioni non possono sottoscrivere l'`INVOKE`evento su Managed Instances a causa del modello di esecuzione simultanea.
> - Metriche: include solo. `durationMs` Le seguenti metriche non sono incluse:`billedDurationMs`, `initDurationMs``maxMemoryUsedMB`, e. `memorySizeMB` Queste metriche per richiamo non sono applicabili nell'ambiente di esecuzione simultanea.

Per le metriche sull'utilizzo delle risorse, usa o Monitoraggio delle istanze gestite da Lambda Lambda Insights.

```
Event: Object
- time: String
- type: String = platform.report
- record: PlatformReport
```

L'oggetto `PlatformReport` ha i seguenti attributi:

- metrics: oggetto the section called “ReportMetrics”
- requestId: `String`
- spans? : elenco di oggetti the section called “Span”
- status: oggetto the section called “Status”
- tracing?: oggetto the section called “TraceContext”

Di seguito è riportato un esempio `Event` di tipo `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`

Un evento `platform.restoreStart` indica che un evento di ripristino dell'ambiente della funzione è stato avviato. In un evento di ripristino dell'ambiente, Lambda crea l'ambiente da uno snapshot

memorizzato nella cache anziché inizializzarlo da zero. Per ulteriori informazioni, consulta Lambda SnapStart. Un oggetto `platform.restoreStart` `Event` ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.restoreStart
- record: PlatformRestoreStart
```

L'oggetto `PlatformRestoreStart` ha i seguenti attributi:

- functionName: `String`
- functionVersion: `String`
- instanceId?: `String`
- instanceMaxMemory? – `String`
- runtimeVersion?: `String`
- runtimeVersionArn? – `String`

Di seguito è riportato un esempio `Event` di tipo `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.restoreRuntimeDone`

Un evento `platform.restoreRuntimeDone` indica che un evento di ripristino dell'ambiente della funzione è stato completato. In un evento di ripristino dell'ambiente, Lambda crea l'ambiente da uno snapshot memorizzato nella cache anziché inizializzarlo da zero. Per ulteriori informazioni, consulta Lambda SnapStart. Un oggetto `platform.restoreRuntimeDone` `Event` ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.restoreRuntimeDone
- record: PlatformRestoreRuntimeDone
```

L'oggetto `PlatformRestoreRuntimeDone` ha i seguenti attributi:

- errorType?: `String`
- spans? : elenco di oggetti the section called “Span”
- status: oggetto the section called “Status”

Di seguito è riportato un esempio `Event` di tipo `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`

Un evento `platform.restoreReport` contiene un rapporto generale di un evento di ripristino della funzione. Un oggetto `platform.restoreReport` `Event` ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.restoreReport
- record: PlatformRestoreReport
```

L'oggetto `PlatformRestoreReport` ha i seguenti attributi:

- errorType?: stringa
- metrics?: oggetto the section called “RestoreReportMetrics”
- spans? : elenco di oggetti the section called “Span”
- status: oggetto the section called “Status”

Di seguito è riportato un esempio `Event` di tipo `platform.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`

Un evento `extension` contiene i log del codice dell'estensione. Un oggetto `extension` `Event` ha la seguente forma:

```
Event: Object
- time: String
- type: String = extension
- record: {}
```

L'oggetto `PlatformExtension` ha i seguenti attributi:

- events: elenco di `String`
- name: `String`
- state: `String`

Di seguito è riportato un esempio `Event` di tipo `platform.extension`:

```
{
    "time": "2022-10-12T00:02:15.000Z",
    "type": "platform.extension",
    "record": {
        "events": [ "INVOKE", "SHUTDOWN" ],
        "name": "my-telemetry-extension",
        "state": "Ready"
    }
}
```

`platform.telemetrySubscription`

Un evento `platform.telemetrySubscription` contiene informazioni su una sottoscrizione dell'estensione. Un oggetto `platform.telemetrySubscription` `Event` ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.telemetrySubscription
- record: PlatformTelemetrySubscription
```

L'oggetto `PlatformTelemetrySubscription` ha i seguenti attributi:

- name: `String`
- state: `String`
- tipi: elenco di `String`

Di seguito è riportato un esempio `Event` di tipo `platform.telemetrySubscription`:

```
{
    "time": "2022-10-12T00:02:35.000Z",
    "type": "platform.telemetrySubscription",
    "record": {
        "name": "my-telemetry-extension",
        "state": "Subscribed",
        "types": [ "platform", "function" ]
    }
}
```

platform.logsDropped

Un evento `platform.logsDropped` contiene informazioni sugli eventi eliminati. Quando una funzione genera log a una velocità troppo elevata per consentire a Lambda di elaborarli, Lambda emette l'evento `platform.logsDropped`. Quando Lambda non è in grado di inviare log a CloudWatch o verso l'estensione sottoscritta all'API di telemetria alla velocità di produzione della funzione, elimina i log per evitare che l'esecuzione della funzione rallenti. Un oggetto `platform.logsDropped` `Event` ha la seguente forma:

```
Event: Object
- time: String
- type: String = platform.logsDropped
- record: PlatformLogsDropped
```

L'oggetto `PlatformLogsDropped` ha i seguenti attributi:

- droppedBytes: `Integer`
- droppedRecords: `Integer`
- reason: `String`

Di seguito è riportato un esempio `Event` di tipo `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"
    }
}
```

function

Un evento `function` contiene i log del codice della funzione. Un oggetto `function` `Event` ha la seguente forma:

```
Event: Object
- time: String
```

```
- type: String = function
- record: {}
```

Il formato del campo `record` dipende dal fatto che i log della funzione siano formattati in testo normale o JSON. Per ulteriori informazioni sulle opzioni di configurazione del formato dei log, consulta la pagina the section called “Formati di registro”

Di seguito è riportato un esempio di `Event` di tipo `function` in cui il formato di log è il testo normale:

```
{
    "time": "2022-10-12T00:03:50.000Z",
    "type": "function",
    "record": "[INFO] Hello world, I am a function!"
}
```

Di seguito è riportato un esempio di `Event` di tipo `function` in cui il formato di log è 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**
>
> Se la versione dello schema che stai utilizzando è precedente alla versione `2022-12-13`, il `"record"` viene sempre visualizzato come stringa anche quando il formato di registrazione della funzione è configurato come JSON. Per le istanze gestite Lambda, è necessario utilizzare la versione dello schema. `2025-01-29`

`extension`

Un evento `extension` contiene i log del codice dell'estensione. Un oggetto `extension` `Event` ha la seguente forma:

```
Event: Object
- time: String
- type: String = extension
- record: {}
```

Il formato del campo `record` dipende dal fatto che i log della funzione siano formattati in testo normale o JSON. Per ulteriori informazioni sulle opzioni di configurazione del formato dei log, consulta la pagina the section called “Formati di registro”

Di seguito è riportato un esempio di `Event` di tipo `extension` in cui il formato di log è il testo normale:

```
{
    "time": "2022-10-12T00:03:50.000Z",
    "type": "extension",
    "record": "[INFO] Hello world, I am an extension!"
}
```

Di seguito è riportato un esempio di `Event` di tipo `extension` in cui il formato di log è 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**
>
> Se la versione dello schema che stai utilizzando è precedente alla versione `2022-12-13`, il `"record"` viene sempre visualizzato come stringa anche quando il formato di registrazione della funzione è configurato come JSON. Per le istanze gestite Lambda, è necessario utilizzare la versione dello schema. `2025-01-29`

Tipi di oggetti condivisi

Questa sezione descrive in dettaglio i tipi di oggetti condivisi supportati dall'API di telemetria Lambda.

`InitPhase`

Una stringa enum che descrive la fase in cui si verifica l'operazione di inizializzazione. Nella maggior parte dei casi, Lambda esegue il codice di inizializzazione della funzione durante la fase `init`. Tuttavia, in alcuni casi di errore, Lambda può eseguire nuovamente il codice di inizializzazione della funzione durante la fase `invoke`. (Questa è chiamata init soppressa.)

- Tipo: `String`
- Valori validi: `init`|`invoke`|`snap-start`

`InitReportMetrics`

Un oggetto che contiene parametri relativi a una fase di inizializzazione.

- Tipo: `Object`

Un oggetto `InitReportMetrics` ha la seguente forma:

```
InitReportMetrics: Object
- durationMs: Double
```

Di seguito è illustrato un esempio di oggetto `InitReportMetrics` di esempio:

```
{
    "durationMs": 247.88
}
```

`InitType`

Una stringa enum che descrive come Lambda ha inizializzato l'ambiente.

- Tipo: `String`
- Valori validi: `on-demand`|`provisioned-concurrency`

`ReportMetrics`

Un oggetto che contiene i parametri di una fase completata.

- Tipo: `Object`

Un oggetto `ReportMetrics` ha la seguente forma:

```
ReportMetrics: Object
- billedDurationMs: Integer
- durationMs: Double
- initDurationMs?: Double
- maxMemoryUsedMB: Integer
- memorySizeMB: Integer
- restoreDurationMs?: Double
```

Di seguito è illustrato un esempio di oggetto `ReportMetrics` di esempio:

```
{
    "billedDurationMs": 694,
    "durationMs": 693.92,
    "initDurationMs": 397.68,
    "maxMemoryUsedMB": 84,
    "memorySizeMB": 128
}
```

`RestoreReportMetrics`

Un oggetto che contiene i parametri di una fase di ripristino completata.

- Tipo: `Object`

Un oggetto `RestoreReportMetrics` ha la seguente forma:

```
RestoreReportMetrics: Object
- durationMs: Double
```

Di seguito è illustrato un esempio di oggetto `RestoreReportMetrics` di esempio:

```
{
    "durationMs": 15.19
```

```
}
```

`RuntimeDoneMetrics`

Un oggetto che contiene parametri relativi a una fase di chiamata.

- Tipo: `Object`

Un oggetto `RuntimeDoneMetrics` ha la seguente forma:

```
RuntimeDoneMetrics: Object
- durationMs: Double
- producedBytes?: Integer
```

Di seguito è illustrato un esempio di oggetto `RuntimeDoneMetrics` di esempio:

```
{
    "durationMs": 200.0,
    "producedBytes": 15
}
```

`Span`

Un oggetto che contiene i dettagli di un intervallo. Un intervallo rappresenta un'unità di lavoro o un'operazione in una traccia. Per ulteriori informazioni sugli span, consulta Span nella pagina Tracing API del sito Web di Docs. OpenTelemetry

Lambda supporta i seguenti intervalli per l'evento `platform.RuntimeDone`:

- L'intervallo `responseLatency` descrive il tempo impiegato dalla funzione Lambda per iniziare a inviare la risposta.
- L'intervallo `responseDuration` descrive il tempo impiegato dalla funzione Lambda per finire di inviare la risposta.
- L'intervallo `runtimeOverhead` descrive il tempo impiegato dal runtime di Lambda per segnalare che è pronto per l'elaborazione del richiamo successivo della funzione. Questo è il tempo impiegato dal runtime per il richiamo dell'API dell'invocazione successiva per ottenere l'evento successivo dopo aver restituito la risposta della funzione.

Di seguito è illustrato un oggetto di intervallo `responseLatency` di esempio:

```
{
        "name": "responseLatency",
        "start": "2022-08-02T12:01:23.521Z",
        "durationMs": 23.02
      }
```

`Status`

Un oggetto che descrive lo stato di una fase di inizializzazione o invocazione. Se lo stato è `failure` o `error`, l'oggetto `Status`contiene anche un campo `errorType` che descrive l'errore.

- Tipo: `Object`
- Valori di stato validi: `success`|`failure`|`error`|`timeout`

`TraceContext`

Un oggetto che descrive le proprietà di una traccia.

- Tipo: `Object`

Un oggetto `TraceContext` ha la seguente forma:

```
TraceContext: Object
- spanId?: String
- type: TracingType enum
- value: String
```

Di seguito è illustrato un esempio di oggetto `TraceContext` di esempio:

```
{
    "spanId": "073a49012f3c312e",
    "type": "X-Amzn-Trace-Id",
    "value":
 "Root=1-62e900b2-710d76f009d6e7785905449a;Parent=0efbd19962d95b05;Sampled=1"
}
```

`TracingType`

Una stringa enum che descrive il tipo di tracciamento in un oggetto the section called “TraceContext”.

- Tipo: `String`
- Valori validi: `X-Amzn-Trace-Id`

Conversione degli oggetti **`Event`** dell'API di telemetria Lambda in intervalli OpenTelemetry

Lo schema dell'API di telemetria AWS Lambda è semanticamente compatibile con OpenTelemetry (OTel). Ciò significa che è possibile convertire gli oggetti `Event` dell'API di telemetria di AWS Lambda in intervalli OpenTelemetry (OTel). Durante la conversione, non dovresti mappare un singolo oggetto `Event` a un singolo intervallo OTel. Dovreste invece presentare tutti e tre gli eventi relativi a una fase del ciclo di vita in un singolo intervallo OTel. Ad esempio, gli eventi `start`, `runtimeDone` e `runtimeReport`rappresentano una singola chiamata di funzione. Presenta tutti e tre questi eventi come un unico intervallo OTel.

Puoi convertire i tuoi eventi usando Span Events o Child Spans (nidificati). Le tabelle di questa pagina descrivono le mappature tra le proprietà dello schema dell'API di telemetria e le proprietà dell'intervallo OTel per entrambi gli approcci. Per ulteriori informazioni sugli intervalli OTel, consulta Intervallo nella pagina API di tracciatura del sito Web della documentazione di OpenTelemetry.

Sezioni

- Mappa a intervalli OTel con eventi di intervallo
- Mappa a intervalli OTel con intervalli secondari

Mappa a intervalli OTel con eventi di intervallo

Nelle seguenti tabelle, e rappresenta l'evento proveniente dall'origine di telemetria.

Mappatura degli eventi *Start

OpenTelemetry	Schema dell'API di telemetria Lambda
`Span.Name`	La tua estensione genera questo valore in base al campo `type`.
`Span.StartTime`	Utilizza `e.time`.
`Span.EndTime`	N/D, perché l'evento non è ancora stato completato.
`Span.Kind`	Imposta su `Server`.

OpenTelemetry	Schema dell'API di telemetria Lambda
`Span.Status`	Imposta su `Unset`.
`Span.TraceId`	Analizza l'intestazione AWS X-Ray trovata in `e.tracing.value` , quindi usa il valore `TraceId`.
`Span.ParentId`	Analizza l'intestazione X-Ray trovata in `e.tracing.value` , quindi usa il valore `Parent`.
`Span.SpanId`	Usa `e.tracing.spanId` se disponibile. Altrimenti, genera un nuovo `SpanId`.
`Span.SpanContext.TraceState`	N/D per un contesto di traccia X-Ray.
`Span.SpanContext.TraceFlags`	Analizza l'intestazione X-Ray trovata in `e.tracing.value` , quindi usa il valore `Sampled`.
`Span.Attributes`	La tua estensione può aggiungere qualsiasi valore personalizzato qui.

Mappatura degli eventi *RuntimeDone

OpenTelemetry	Schema dell'API di telemetria Lambda
`Span.Name`	L'estensione genera il valore in base al campo `type`.
`Span.StartTime`	Usa `e.time` dall'evento `*Start` corrispon dente. In alternativa, utilizzare `e.time - e.metrics.durationMs` .

OpenTelemetry	Schema dell'API di telemetria Lambda
`Span.EndTime`	N/D, perché l'evento non è ancora stato completato.
`Span.Kind`	Imposta su `Server`.
`Span.Status`	Se `e.status` non è uguale a `success`, imposta su `Error`. In caso contrario, imposta il valore su `Ok`.
`Span.Events[]`	Utilizza `e.spans[]` .
`Span.Events[i].Name`	Utilizza `e.spans[i].name` .
`Span.Events[i].Time`	Utilizza `e.spans[i].start` .
`Span.TraceId`	Analizza l'intestazione AWS X-Ray trovata in `e.tracing.value` , quindi usa il valore `TraceId`.
`Span.ParentId`	Analizza l'intestazione X-Ray trovata in `e.tracing.value` , quindi usa il valore `Parent`.
`Span.SpanId`	Usa lo stesso `SpanId` dell'evento `*Start`. Se non disponibile, usa `e.tracing.spanId` o genera un nuovo `SpanId`.
`Span.SpanContext.TraceState`	N/D per un contesto di traccia X-Ray.
`Span.SpanContext.TraceFlags`	Analizza l'intestazione X-Ray trovata in `e.tracing.value` , quindi usa il valore `Sampled`.
`Span.Attributes`	La tua estensione può aggiungere qualsiasi valore personalizzato qui.

Mappatura degli eventi *Report

OpenTelemetry	Schema dell'API di telemetria Lambda
`Span.Name`	L'estensione genera il valore in base al campo `type`.
`Span.StartTime`	Usa `e.time` dall'evento `*Start` corrispondente. In alternativa, utilizzare `e.time - e.metrics.durationMs` .
`Span.EndTime`	Utilizza `e.time`.
`Span.Kind`	Imposta su `Server`.
`Span.Status`	Usa lo stesso valore dell'evento `*RuntimeDone` .
`Span.TraceId`	Analizza l'intestazione AWS X-Ray trovata in `e.tracing.value` , quindi usa il valore `TraceId`.
`Span.ParentId`	Analizza l'intestazione X-Ray trovata in `e.tracing.value` , quindi usa il valore `Parent`.
`Span.SpanId`	Usa lo stesso `SpanId` dell'evento `*Start`. Se non disponibile, usa `e.tracing.spanId` o genera un nuovo `SpanId`.
`Span.SpanContext.TraceState`	N/D per un contesto di traccia X-Ray.
`Span.SpanContext.TraceFlags`	Analizza l'intestazione X-Ray trovata in `e.tracing.value` , quindi usa il valore `Sampled`.
`Span.Attributes`	La tua estensione può aggiungere qualsiasi valore personalizzato qui.

Mappa a intervalli OTel con intervalli secondari

La tabella seguente descrive come convertire gli eventi dell'API di telemetria Lambda in intervalli OTel con intervalli secondari (annidati) per gli intervalli `*RuntimeDone`. Per le mappature `*Start` e `*Report`, fai riferimento alle tabelle in the section called “Mappa a intervalli OTel con eventi di intervallo”, poiché sono le stesse per gli intervalli secondari. In questa tabella, e rappresenta l'evento proveniente dall'origine di telemetria.

Mappatura degli eventi *RuntimeDone

OpenTelemetry	Schema dell'API di telemetria Lambda
`Span.Name`	L'estensione genera il valore in base al campo `type`.
`Span.StartTime`	Usa `e.time` dall'evento `*Start` corrispon dente. In alternativa, utilizzare `e.time - e.metrics.durationMs` .
`Span.EndTime`	N/D, perché l'evento non è ancora stato completato.
`Span.Kind`	Imposta su `Server`.
`Span.Status`	Se `e.status` non è uguale a `success`, imposta su `Error`. In caso contrario, imposta il valore su `Ok`.
`Span.TraceId`	Analizza l'intestazione AWS X-Ray trovata in `e.tracing.value` , quindi usa il valore `TraceId`.
`Span.ParentId`	Analizza l'intestazione X-Ray trovata in `e.tracing.value` , quindi usa il valore `Parent`.

OpenTelemetry	Schema dell'API di telemetria Lambda
`Span.SpanId`	Usa lo stesso `SpanId` dell'evento `*Start`. Se non disponibile, usa `e.tracing.spanId` o genera un nuovo `SpanId`.
`Span.SpanContext.TraceState`	N/D per un contesto di traccia X-Ray.
`Span.SpanContext.TraceFlags`	Analizza l'intestazione X-Ray trovata in `e.tracing.value`, quindi usa il valore `Sampled`.
`Span.Attributes`	La tua estensione può aggiungere qualsiasi valore personalizzato qui.
`ChildSpan[i].Name`	Utilizza `e.spans[i].name`.
`ChildSpan[i].StartTime`	Utilizza `e.spans[i].start`.
`ChildSpan[i].EndTime`	Utilizza `e.spans[i].start + e.spans[i].durations`.
`ChildSpan[i].Kind`	Uguale all'elemento padre `Span.Kind`.
`ChildSpan[i].Status`	Uguale all'elemento padre `Span.Status`.
`ChildSpan[i].TraceId`	Uguale all'elemento padre `Span.TraceId`.
`ChildSpan[i].ParentId`	Usa elemento padre `Span.SpanId`.
`ChildSpan[i].SpanId`	Genera un nuovo `SpanId`.
`ChildSpan[i].SpanContext.TraceState`	N/D per un contesto di traccia X-Ray.
`ChildSpan[i].SpanContext.TraceFlags`	Uguale all'elemento padre `Span.SpanContext.TraceFlags`.

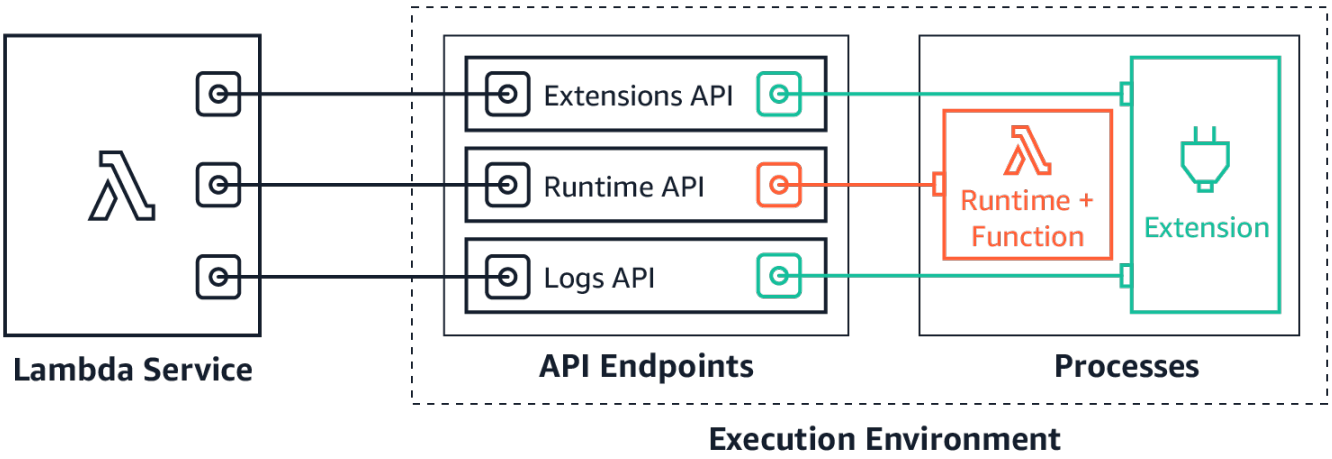
Utilizzo dell'API Logs di Lambda

> **Important**
>
> L'API di telemetria Lambda sostituisce l'API Lambda Logs. Sebbene l'API Logs rimanga completamente funzionante, in futuro consigliamo di utilizzare solo l'API di telemetria. Puoi iscrivere la tua estensione a un flusso di telemetria utilizzando l'API di telemetria o l'API Logs. Dopo la sottoscrizione utilizzando una di queste APIs, qualsiasi tentativo di sottoscrizione utilizzando l'altra API restituisce un errore.

> **Le istanze gestite Lambda non supportano l'API Logs**
>
> Le istanze gestite Lambda non supportano l'API Logs. Se utilizzi le funzioni di istanze gestite, utilizza invece l'API di telemetria. L'API di telemetria offre funzionalità avanzate per la raccolta e l'elaborazione dei dati di telemetria dalle funzioni Lambda.

Lambda acquisisce automaticamente i log di runtime e li trasmette ad Amazon. CloudWatch Questo flusso di registri contiene i log generati dal codice funzione e dalle estensioni e anche i log generati da Lambda nell'ambito del richiamo della funzione.

Le estensioni Lambda possono utilizzare l'API Logs del runtime di Lambda per eseguire la sottoscrizione ai flussi di log direttamente dall'interno dell'ambiente di esecuzione Lambda. Lambda trasmette i log all'estensione e l'estensione può quindi elaborare, filtrare e inviare i log a qualsiasi destinazione preferita.



L'API Logs consente alle estensioni di sottoscrivere tre flussi di log diversi:

- Log di funzioni che la funzione Lambda genera e scrive in `stdout` o `stderr`.
- Log di estensione generati dal codice di estensione.
- Log si piattaforma Lambda, che registrano eventi ed errori relativi a richiami ed estensioni.

> **Note**
>
> Lambda invia tutti i log a CloudWatch, anche quando un'estensione sottoscrive uno o più flussi di log.

Argomenti

- Sottoscrizione ai log di ricezione
- Utilizzo della memoria
- Protocolli destinazione
- Configurazione buffering
- Esempio di sottoscrizione
- Codice di esempio per l'API Logs
- Riferimento dell'API Logs.
- Messaggi di log

Sottoscrizione ai log di ricezione

Un'estensione Lambda può sottoscrivere i log di ricezione inviando una richiesta di sottoscrizione all'API Logs.

Per sottoscrivere i log di ricezione, è necessario l'identificatore di estensione (`Lambda-Extension-Identifier`). Innanzitutto registrare l'estensione per ricevere l'identificatore dell'estensione. Quindi sottoscrivere l'API Logs durante l'inizializzazione. Al termine della fase di inizializzazione, Lambda non elabora le richieste di sottoscrizione.

> **Note**
>
> La sottoscrizione Logs API è idempotente. Le richieste di sottoscrizione duplicate non comportano sottoscrizioni duplicate.

Utilizzo della memoria

L'utilizzo della memoria aumenta linearmente man mano che aumenta il numero di abbonati. Le sottoscrizioni consumano risorse di memoria perché ogni sottoscrizione apre un nuovo buffer di memoria per archiviare i log. Per ottimizzare l'utilizzo della memoria, è possibile regolare la configurazione di buffering. L'utilizzo della memoria buffer conta per il consumo complessivo della memoria nell'ambiente di esecuzione.

Protocolli destinazione

È possibile scegliere uno dei seguenti protocolli per ricevere i log:

1. HTTP (consigliato) – Lambda consegna i log a un endpoint HTTP locale (`http://sandbox.localdomain:${PORT}/${PATH}`) come matrice di record in formato JSON. Il parametro `$PATH` è facoltativo. Si noti che è supportato solo HTTP, non HTTPS. Puoi scegliere di ricevere i log tramite PUT o POST.
2. TCP – Lambda consegna i log a una porta TCP in formato JSON delimitato da Newline (NDJSON).

Si consiglia di utilizzare HTTP anziché TCP. Con TCP, la piattaforma Lambda non può riconoscere che i log vengono consegnati al livello dell'applicazione. Pertanto, se l'estensione si blocca si potrebbero perdere i log. HTTP non condivide questa limitazione.

Si consiglia inoltre di impostare il listener HTTP locale o la porta TCP prima di sottoscrivere i log di ricezione. Durante l'installazione, tenere presente quanto segue:

- Lambda invia i log solo alle destinazioni che si trovano all'interno dell'ambiente di esecuzione.
- Lambda ritenta il tentativo di inviare i log (con backoff) se non c'è listener o se la richiesta POST o PUT genera errori. Se il sottoscrittore del log si blocca, continuerà a ricevere i log dopo che Lambda riavvia l'ambiente di esecuzione.
- Lambda riserva la porta 9001. Non ci sono altre restrizioni o raccomandazioni sul numero di porta.

Configurazione buffering

Lambda può tamponare i log e consegnarli al sottoscrittore. È possibile configurare questo comportamento nella richiesta di sottoscrizione specificando i seguenti campi facoltativi. Si noti che Lambda utilizza il valore predefinito per qualsiasi campo non specificato.

- timeoutMs – Il tempo massimo (in millisecondi) per il buffer di un batch. Valore predefinito: 1.000 Minimo: 25. Massimo: 30.000
- maxBytes – La dimensione massima (in byte) dei registri al buffer in memoria. Valore predefinito: 262.144. Minimo: 262.144. Massimo: 1.048.576.
- maxItems – Il numero massimo di eventi da memorizzare nel buffer. Valore predefinito: 10.000 Minimo: 1.000. Massimo: 10.000.

Durante la configurazione del buffering, prendere nota dei seguenti punti:

- Lambda svuota i log se uno qualsiasi dei flussi di input è chiuso, ad esempio, se il runtime si arresta in modo anomalo.
- Ogni sottoscrittore può specificare una configurazione di buffering diversa durante la richiesta di sottoscrizione.
- Considerare la dimensione del buffer necessaria per leggere i dati. Aspettarsi di ricevere payload grandi come `2*maxBytes+metadata`, dove `maxBytes` è configurato nella richiesta di sottoscrizione. Ad esempio, Lambda aggiunge i byte di metadati seguenti a ciascun record:

```
{
"time": "2020-08-20T12:31:32.123Z",
"type": "function",
"record": "Hello World"
}
```

- Se il server di sottoscrizione non è in grado di elaborare i log in ingresso abbastanza rapidamente, Lambda potrebbe eliminare i log per mantenere limitato l'utilizzo della memoria. Per indicare il numero di record eliminati, Lambda invia un log `platform.logsDropped`. Per ulteriori informazioni, consulta the section called “Lambda: non vengono visualizzati tutti i log della mia funzione”.

Esempio di sottoscrizione

L'esempio seguente mostra una richiesta di sottoscrizione ai log della piattaforma e delle funzioni.

```
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"
  }
}
```

Se la richiesta ha esito positivo, il sottoscrittore riceve una risposta riuscita HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

Codice di esempio per l'API Logs

Per un codice di esempio che mostra come inviare i log a una destinazione personalizzata, consulta Usare AWS Lambda le estensioni per inviare log a destinazioni personalizzate su Compute Blog. AWS

Per esempi di codice in Python e Go che mostrano come sviluppare un'estensione Lambda di base e sottoscrivere l'API Logs, consulta AWS Lambda Extensions on the Samples repository. AWS GitHub Per ulteriori informazioni sulla creazione di un'estensione Lambda, consultare the section called “API estensioni”.

Riferimento dell'API Logs.

È possibile recuperare l'endpoint API Logs dalla`AWS_LAMBDA_RUNTIME_API` variabile di ambiente. Per inviare una richiesta API, utilizzare il prefisso `2020-08-15/` prima del percorso API. Esempio:

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-08-15/logs
```

La specifica OpenAPI per la versione Logs API 2020-08-15 è disponibile qui: .zip logs-api-request

Subscribe

Per sottoscrivere uno o più flussi di log disponibili nell'ambiente di esecuzione di Lambda, le estensioni inviano una richiesta API di sottoscrizione.

Percorso – `/logs`

Method – PUT

Parametri corpo

`destination` - Consultare the section called “Protocolli destinazione”. Campo obbligatorio: sì Tipo: stringhe.

`buffering` - Consultare the section called “Configurazione buffering”. Campo obbligatorio: no. Tipo: stringhe.

`types` – Un array dei tipi di log da ricevere. Campo obbligatorio: sì Tipo: array di stringhe Valori validi: «piattaforma», «funzione», «estensione».

`schemaVersion` – Obbligatorio: no. Valore di default: "2020-08-15". Impostare su "2021-03-18" per l'estensione in modo da ricevere messaggi `platform.runtimeDone`.

Parametri di risposta

Le specifiche OpenAPI per le risposte di sottoscrizione, versione 2020-08-15, sono disponibili per i protocolli HTTP e TCP:

- logs-api-http-responseHTTP: .zip
- TCP: .zip logs-api-tcp-response

Codice di risposta

- 200 – Richiesta completata con successo
- 202 – Richiesta accettata. Risposta ad una richiesta di sottoscrizione durante il test locale.
- 4XX – Richiesta non valida

- 500 – Errore servizio

Se la richiesta ha esito positivo, il sottoscrittore riceve una risposta riuscita HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

Se la richiesta fallisce, il sottoscrittore riceve una risposta di errore. Ad esempio:

```
HTTP/1.1 400 OK
{
    "errorType": "Logs.ValidationError",
    "errorMessage": URI port is not provided; types should not be empty"
}
```

Messaggi di log

L'API Logs consente alle estensioni di sottoscrivere tre flussi di log diversi:

- Funzione – Log che la funzione Lambda genera e scrive in `stdout` o `stderr`.
- Estensione – Log generati dal codice di estensione.
- Piattaforma – Log generati dalla piattaforma runtime, che registrano eventi ed errori relativi a richiami ed estensioni.

Argomenti

- Log delle funzioni
- Log di estensioni
- Log di piattaforma

Log delle funzioni

La funzione Lambda e le estensioni interne generano log di funzioni e li scrivono in `stdout` o `stderr`.

L'esempio seguente mostra il formato di un messaggio di log di funzioni. { "time": "2020-08-20T12:31:32.123Z", "type": "function", "record": "ERROR encountered. Stack trace:\n\my-function (line 10)\n" }

Log di estensioni

Le estensioni possono generare log di estensioni. Il formato del log è uguale a quello di un log di funzioni.

Log di piattaforma

Lambda genera messaggi di log per eventi della piattaforma come `platform.start`, `platform.end` e `platform.fault`.

Facoltativamente, è possibile sottoscrivere la versione 2021-03-18 dello schema dell'API Log, che include il messaggio di log `platform.runtimeDone`.

Esempio di messaggi di log di piattaforma

Nell'esempio seguente vengono illustrati i log di inizio e di fine della piattaforma. Questi log indicano l'ora di inizio e l'ora di fine chiamata per la chiamata specificata da 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"}
}
```

La piattaforma. initRuntimeDoneil messaggio di registro mostra lo stato della `Runtime init` sottofase, che fa parte della fase del ciclo di vita di Init. In caso di riuscita di `Runtime init`, il runtime invia una richiesta API `/next` di runtime (per i tipi di inizializzazione `on-demand` e `provisioned-concurrency`) o `restore/next` (per il tipo di inizializzazione `snap-start`). L'esempio seguente mostra una piattaforma di successo. initRuntimeDonemessaggio di registro per il tipo di `snap-start` inizializzazione.

```
{
  "time":"2022-07-17T18:41:57.083Z",
  "type":"platform.initRuntimeDone",
  "record":{
      "initializationType":"snap-start",
      "status":"success"
```

```
  }
}
```

Il messaggio di log platform.initReport mostra quanto è durata la fase `Init` e quanti millisecondi sono stati fatturati durante questa fase. Quando il tipo di inizializzazione è `provisioned-concurrency`, Lambda invia questo messaggio durante la chiamata. Quando il tipo di inizializzazione è `snap-start`, Lambda invia questo messaggio dopo aver ripristinato lo snapshot. L'esempio seguente mostra un messaggio di log platform.initReport riuscito per il tipo di inizializzazione `snap-start`.

```
{
  "time":"2022-07-17T18:41:57.083Z",
  "type":"platform.initReport",
  "record":{
      "initializationType":"snap-start",
      "metrics":{
          "durationMs":731.79,
          "billedDurationMs":732
          }
  }
}
```

Il log dei report della piattaforma include parametri sulla chiamata specificata da requestId. Il campo `initDurationMs` è incluso nel log solo se la chiamata include un avvio a freddo. Se la traccia AWS X-Ray è attiva, il log include i metadati X-Ray. Nell'esempio seguente viene illustrato un log di rapporto della piattaforma per una chiamata che includeva un avvio a freddo.

```
{
    "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
        }
    }
}
```

Il log della piattaforma acquisisce errori di runtime o dell'ambiente di esecuzione. L'esempio seguente mostra un messaggio di log di errore della piattaforma.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.fault",
    "record": "RequestId: d783b35e-a91d-4251-af17-035953428a2c Process exited before
 completing request"
}
```

> **Note**
>
> AWS sta attualmente implementando modifiche al servizio Lambda. A causa di queste modifiche, potresti notare piccole differenze tra la struttura e il contenuto dei messaggi di log di sistema e dei segmenti di traccia emessi da diverse funzioni Lambda nel tuo Account AWS. Uno degli output dei log interessati da questa modifica è il campo `"record"` del log degli errori della piattaforma. Gli esempi seguenti mostrano campi `"record"` nei formati vecchi e nuovi. Il nuovo stile del log degli errori contiene un messaggio più conciso
> Queste modifiche verranno implementate nelle prossime settimane e tutte le funzioni, Regioni AWS ad eccezione della Cina e delle GovCloud regioni, passeranno all'utilizzo dei messaggi di registro e dei segmenti di traccia di nuovo formato.

Example record del log degli errori della piattaforma (vecchio stile)

```
"record":"RequestId: ...\tError: Runtime exited with error: exit status
 255\nRuntime.ExitError"
```

Example record del log degli errori della piattaforma (nuovo stile)

```
"record":"RequestId: ... Status: error\tErrorType: Runtime.ExitError"
```

Lambda genera un log di estensioni della piattaforma quando un'estensione si registra con l'API delle estensioni. L'esempio seguente mostra un messaggio dell'estensione della piattaforma.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.extension",
    "record": {"name": "Foo.bar",
        "state": "Ready",
        "events": ["INVOKE", "SHUTDOWN"]
```

```
     }
}
```

Lambda genera un log di sottoscrizione al log della piattaforma quando un'estensione sottoscrive l'API Log. Nell'esempio seguente viene illustrato un messaggio di sottoscrizione log.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.logsSubscription",
    "record": {"name": "Foo.bar",
        "state": "Subscribed",
        "types": ["function", "platform"],
    }
}
```

Lambda genera un log eliminato dei log di piattaforma quando un'estensione non è in grado di elaborare il numero di log che sta ricevendo. Nell'esempio seguente viene mostrato un`platform.logsDropped` messaggio di log.

```
{
    "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
    }
}
```

Il messaggio di log platform.restoreStart mostra l'ora di inizio della fase `Restore` (solo per il tipo di inizializzazione `snap-start`). Esempio:

```
{
  "time":"2022-07-17T18:43:44.782Z",
  "type":"platform.restoreStart",
  "record":{}
}
```

Il messaggio di log platform.restoreReport mostra quanto è durata la fase `Restore` e quanti millisecondi sono stati fatturati durante questa fase (solo per il tipo di inizializzazione `snap-start`). Esempio:

```
{
  "time":"2022-07-17T18:43:45.936Z",
  "type":"platform.restoreReport",
  "record":{
      "metrics":{
          "durationMs":70.87,
          "billedDurationMs":13
      }
  }
}
```

Messaggi **`runtimeDone`** della piattaforma

Se si imposta la versione dello schema su "2021-03-18" nella richiesta di sottoscrizione, Lambda invia un messaggio `platform.runtimeDone` dopo il completamento della chiamata della funzione o con un errore. L'estensione può utilizzare questo messaggio per arrestare tutta la raccolta di dati di telemetria per questa chiamata di funzione.

La specifica OpenAPI per il tipo di evento Log nella versione dello schema 2021-03-18 è disponibile qui: schema-2021-03-18.zip

Lambda genera il messaggio di log `platform.runtimeDone` quando il runtime invia una richiesta API di runtime `Next` o `Error`. Il log `platform.runtimeDone` informa i consumatori dell'API Log che la chiamata di funzione è stata completata. Le estensioni possono utilizzare queste informazioni per decidere quando inviare tutti i dati di telemetria raccolti durante tale chiamata.

Esempi

Lambda invia il messaggio `platform.runtimeDone` dopo che il runtime invia la richiesta NEXT al completamento della chiamata della funzione. Negli esempi seguenti vengono illustrati i messaggi relativi a ciascuno dei valori di stato: riuscita, errore e timeout.

Example Esempio di messaggio di riuscita

```
{
    "time": "2021-02-04T20:00:05.123Z",
    "type": "platform.runtimeDone",
    "record": {
       "requestId":"6f7f0961f83442118a7af6fe80b88",
       "status": "success"
    }
```

```
}
```

Example Esempio di messaggio di errore

```
{
   "time": "2021-02-04T20:00:05.123Z",
   "type": "platform.runtimeDone",
   "record": {
      "requestId":"6f7f0961f83442118a7af6fe80b88",
      "status": "failure"
   }
}
```

Example Esempio di messaggio di timeout

```
{
   "time": "2021-02-04T20:00:05.123Z",
   "type": "platform.runtimeDone",
   "record": {
      "requestId":"6f7f0961f83442118a7af6fe80b88",
      "status": "timeout"
  }
}
```

Example Piattaforma di esempio. restoreRuntimeDone messaggio (solo tipo di **`snap-start`** inizializzazione)

La piattaforma. restoreRuntimeDoneil messaggio di registro mostra se la `Restore` fase ha avuto successo o meno. Lambda genera questo messaggio quando il runtime invia una richiesta API di runtime `restore/next`. Ci sono tre stati possibili: riuscito, errore e timeout. L'esempio seguente mostra una piattaforma di successo. restoreRuntimeDonemessaggio di registro.

```
{
  "time":"2022-07-17T18:43:45.936Z",
  "type":"platform.restoreRuntimeDone",
  "record":{
      "status":"success"
  }
}
```

Risoluzione dei problemi in Lambda

Negli argomenti seguenti vengono forniti suggerimenti per la risoluzione dei problemi relativi a errori e problemi che potrebbero verificarsi durante l'utilizzo della console, degli strumenti o dell'API Lambda. Se scopri un problema che non è elencato qui di seguito, puoi utilizzare il pulsante Feedback in questa pagina per segnalarlo.

Per ulteriori suggerimenti sulla risoluzione dei problemi e per risposte a domande comuni relative al supporto, visitare il Knowledge Center di AWS.

Per ulteriori informazioni sul debug e la risoluzione dei problemi delle applicazioni Lambda, consulta Debug in Serverless Land.

Argomenti

- Risoluzione dei problemi relativi alle configurazioni in Lambda
- Risoluzione dei problemi relativi alle implementazioni in Lambda
- Risoluzione dei problemi di invocazione in Lambda
- Risoluzione dei problemi di esecuzione in Lambda
- Risolvi i problemi di mappatura delle sorgenti degli eventi in Lambda
- Risoluzione dei problemi di rete in Lambda

Risoluzione dei problemi relativi alle configurazioni in Lambda

Le impostazioni di configurazione delle funzioni possono avere un impatto sulle prestazioni e sul comportamento complessivi della funzione Lambda. Queste potrebbero non causare errori di funzionamento effettivi, ma possono causare timeout e risultati imprevisti.

Negli argomenti seguenti vengono forniti suggerimenti per la risoluzione dei problemi relativi a problemi comuni che potrebbero verificarsi relativi alle impostazioni di configurazione delle funzioni Lambda.

Argomenti

- Configurazioni della memoria
- Configurazioni collegate alla CPU

- Timeout
- Perdita di memoria tra invocazioni
- Risultati asincroni restituiti a una invocazione successiva

Configurazioni della memoria

È possibile configurare una funzione Lambda per utilizzare quantità compresa tra 128 MB e 10.240 MB. Per impostazione predefinita, a qualsiasi funzione creata nella console viene assegnata la quantità di memoria minima. Molte funzioni Lambda offrono prestazioni ottimali con questa impostazione minima. Tuttavia, se si importano librerie di codici di grandi dimensioni o si completano attività che richiedono molta memoria, 128 MB non sono sufficienti.

Se le funzioni funzionano molto più lentamente del previsto, il primo passaggio consiste nell'aumentare l'impostazione della memoria. Per le funzioni collegate alla memoria, ciò risolve il collo di bottiglia e può migliorare le prestazioni della tua funzione.

Configurazioni collegate alla CPU

Per le operazioni a elaborazione intensiva, se le prestazioni della funzione Lambda sono più lente del previsto, ciò potrebbe essere dovuto al fatto che la funzione è vincolata alla CPU. In questo caso, la capacità di calcolo della funzione non può tenere il passo con il lavoro.

Sebbene Lambda non consenta di modificare direttamente la configurazione della CPU, la CPU viene controllata indirettamente tramite le impostazioni della memoria. Il servizio Lambda alloca proporzionalmente più CPU virtuale man mano che si alloca più memoria. Con 1,8 GB di memoria, una funzione Lambda ha un'intera vCPU allocata e, al di sopra di questo livello, ha accesso a più di un core vCPU. Con 10.240 MB, ha 6 vCPU disponibili. In questi casi, è possibile migliorare le prestazioni aumentando l'allocazione di memoria, anche se la funzione non utilizza tutta la memoria.

Timeout

I timeout per le funzioni Lambda possono essere impostati tra 1 e 900 secondi (15 minuti). Per impostazione predefinita, la console Lambda imposta questo valore su 3 secondi. Il valore di timeout è una valvola di sicurezza che garantisce che le funzioni non funzionino all'infinito. Una volta raggiunto il valore di timeout, Lambda interrompe la chiamata della funzione.

Se un valore di timeout viene impostato vicino alla durata media di una funzione, aumenta il rischio che la funzione scada inaspettatamente. La durata di una funzione può variare in base alla quantità di

trasferimento ed elaborazione dei dati e alla latenza dei servizi con cui interagisce la funzione. Alcune cause comuni di timeout includono:

- Quando si scaricano dati da bucket S3 o altri archivi di dati, il download è più grande o richiede più tempo della media.
- Una funzione invia una richiesta a un altro servizio, che impiega più tempo a rispondere.
- I parametri forniti a una funzione richiedono una maggiore complessità computazionale della funzione, il che fa sì che l'invocazione richieda più tempo.

Nel testare l'applicazione, assicurati che i test riflettano accuratamente la dimensione e la quantità di dati e valori realistici dei parametri. È importante sottolineare che utilizza set di dati al limite massimo di quanto ragionevolmente previsto per il carico di lavoro.

Laddove possibile, implementa limiti massimi nel tuo carico di lavoro. In questo esempio, l'applicazione potrebbe utilizzare un limite di dimensione massima per ogni tipo di file. È quindi possibile testare le prestazioni dell'applicazione per una gamma di dimensioni di file previste, fino ai limiti massimi inclusi.

Perdita di memoria tra invocazioni

Le variabili e gli oggetti globali memorizzati nella fase INIT di una invocazione Lambda mantengono il loro stato tra le invocazioni a caldo. Vengono ripristinate completamente solo quando l'ambiente di esecuzione viene eseguito per la prima volta (noto anche come "avvio a freddo"). Tutte le variabili memorizzate nell'handler vengono distrutte quando l'handler viene terminato. È consigliabile utilizzare la fase INIT per configurare connessioni al database, caricare librerie, creare cache e caricare risorse immutabili.

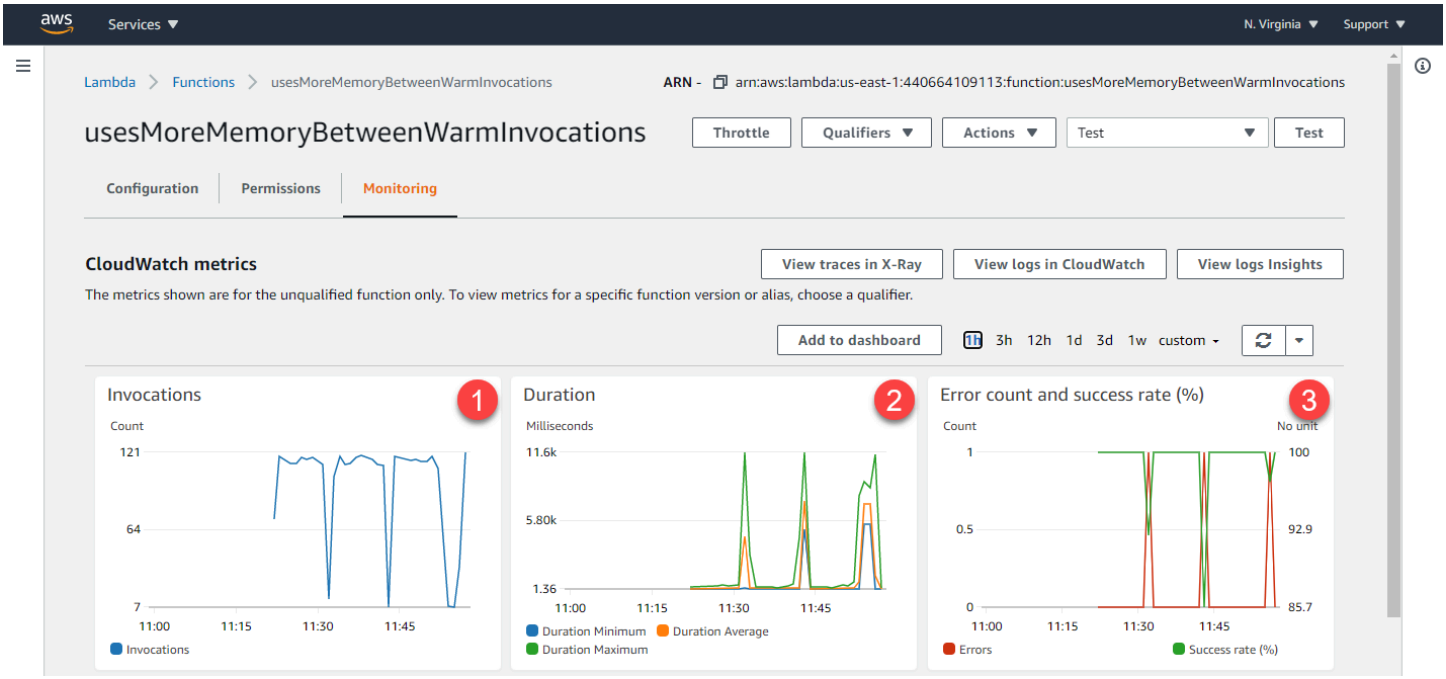
Quando utilizzi librerie di terze parti per più invocazioni nello stesso ambiente di esecuzione, assicurati di consultare la relativa documentazione per l'utilizzo in un ambiente di elaborazione serverless. Alcune librerie di connessione e registrazione al database possono salvare risultati di invocazione intermedi e altri dati. Ciò fa sì che l'utilizzo della memoria di queste librerie aumenti con le successive invocazioni a caldo. Nei casi in cui la memoria cresce rapidamente, è possibile che la funzione Lambda esaurisca la memoria, anche se il codice personalizzato elimina le variabili correttamente.

Questo problema riguarda le invocazioni che si verificano in ambienti di esecuzione a caldo. Ad esempio, il seguente codice crea una perdita di memoria tra le invocazioni. La funzione Lambda consuma memoria aggiuntiva ad ogni invocazione aumentando le dimensioni di un array globale:

```
let a = []

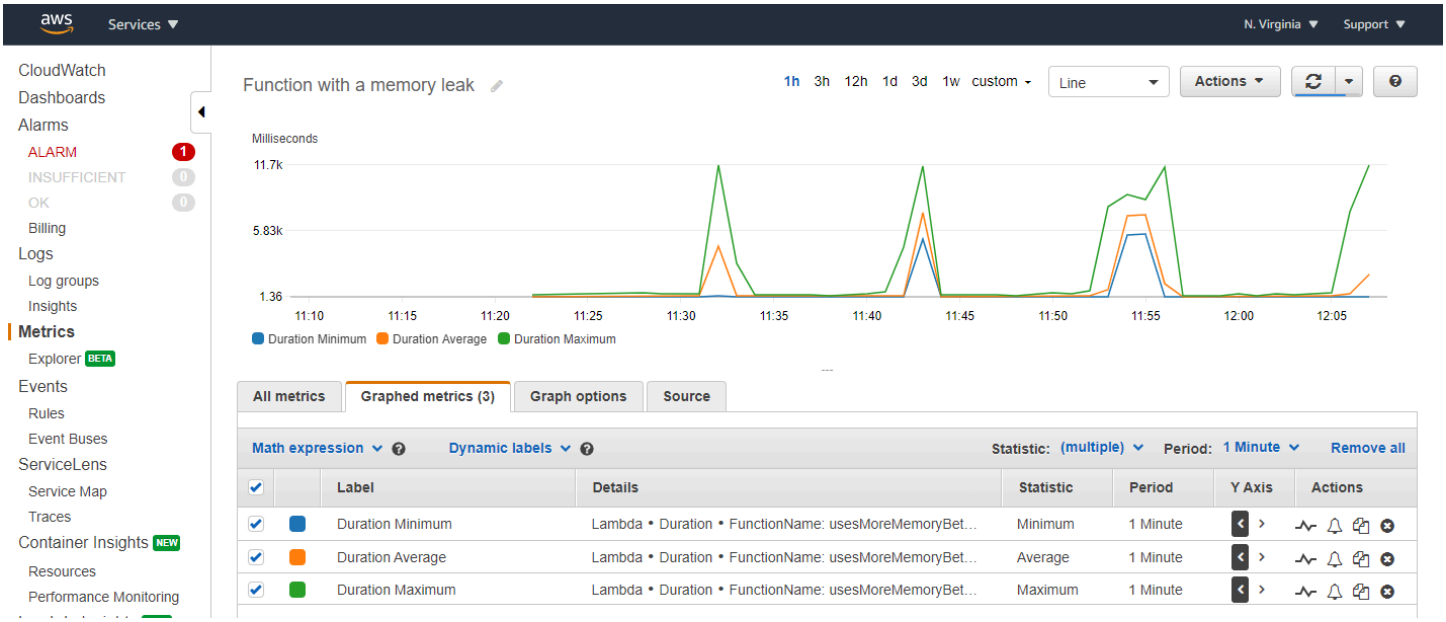
exports.handler = async (event) => {
    a.push(Array(100000).fill(1))
}
```

Configurata con 128 MB di memoria, dopo aver richiamato questa funzione 1.000 volte, la scheda Monitoraggio della funzione Lambda mostra le tipiche modifiche nelle invocazioni, nella durata e nel conteggio degli errori quando si verifica una perdita di memoria:



1. Invocazioni: un tasso di transazione costante viene interrotto periodicamente man mano che le invocazioni impiegano più tempo per essere completate. Durante lo stato stazionario, la perdita di memoria non consuma tutta la memoria allocata alla funzione. Man mano che le prestazioni peggiorano, il sistema operativo utilizza la memoria locale per adattarla alla crescente quantità di memoria richiesta dalla funzione, il che comporta un minor numero di transazioni completate.
2. Durata: prima che la funzione esaurisca la memoria, termina le invocazioni a una velocità costante di due millisecondi. Man mano che si verifica la paginazione, la durata aumenta di un ordine di grandezza.
3. Numero di errori: quando la perdita di memoria supera la memoria allocata, alla fine la funzione subisce errori dovuti al superamento del timeout di calcolo o l'ambiente di esecuzione arresta la funzione.

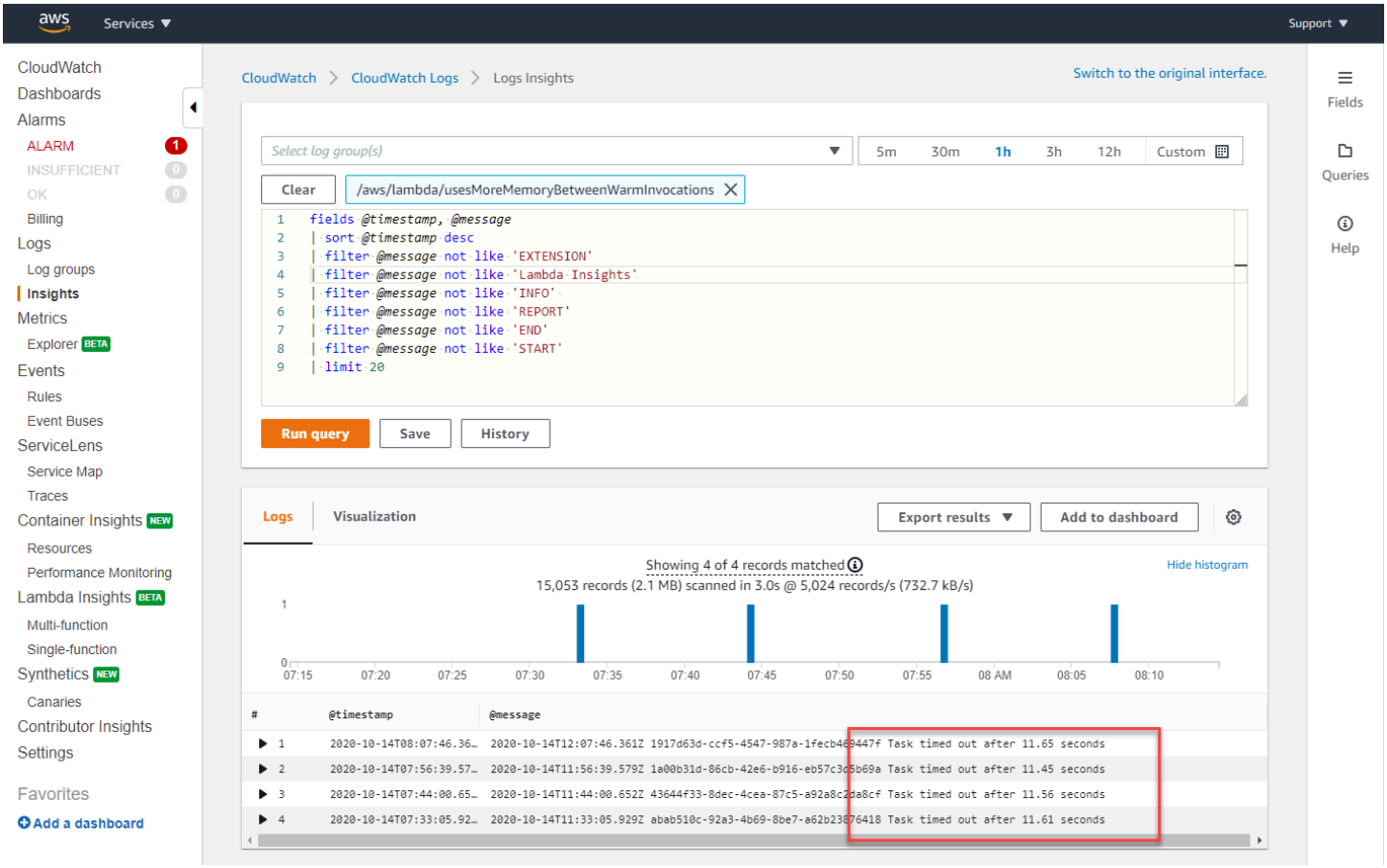
Dopo l'errore, il servizio Lambda riavvia l'ambiente di esecuzione, il che spiega perché in tutti e tre i grafici i parametri tornano allo stato originale. L'espansione dei parametri di CloudWatch per la durata fornisce maggiori dettagli per le statistiche sulla durata minima, massima e media:



Per trovare gli errori generati nelle 1.000 invocazioni, puoi utilizzare il linguaggio di query di CloudWatch Insights. La seguente query esclude i log informativi per riportare solo gli errori:

```
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'
```

Se eseguita sul gruppo di log per questa funzione, ciò dimostra che i timeout erano responsabili degli errori periodici:



Risultati asincroni restituiti a una invocazione successiva

Per il codice di funzione che utilizza modelli asincroni, è possibile che i risultati di callback di una invocazione vengano restituiti in una invocazione futura. Questo esempio utilizza Node.js ma la stessa logica può essere applicata ad altri runtime utilizzando modelli asincroni. La funzione utilizza la tradizionale sintassi di callback in JavaScript. Richiama una funzione asincrona con un contatore incrementale che tiene traccia del numero di invocazioni:

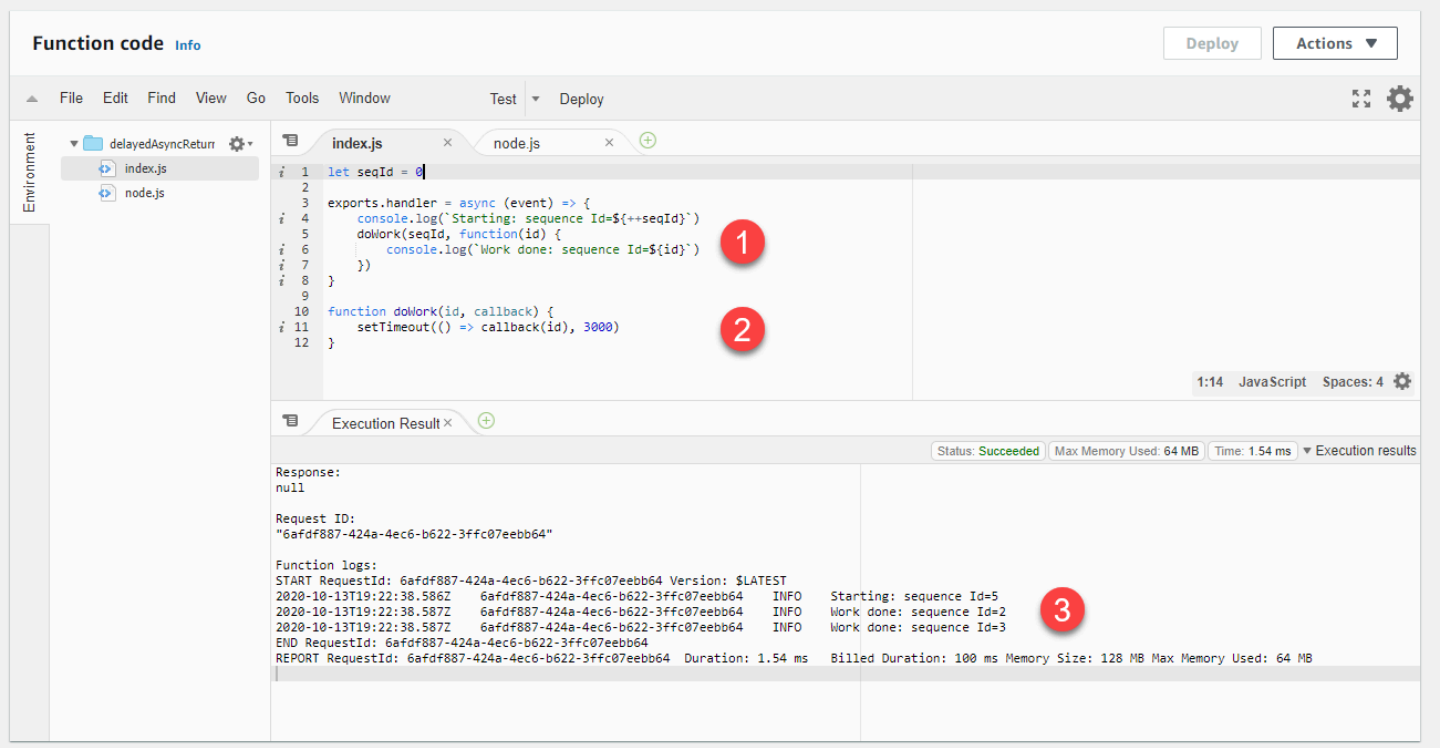
```
let seqId = 0

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)
```

```
}
```

Quando vengono richiamati più volte in successione, i risultati dei callback si verificano nelle invocazioni successive:



1. Il codice richiama la funzione doWork, fornendo una funzione di callback come ultimo parametro.
2. Il completamento della funzione doWork richiede un certo periodo di tempo prima di richiamare il callback.
3. La registrazione della funzione indica che la chiamata termina prima che la funzione DoWork termini l'esecuzione. Inoltre, dopo l'avvio di un'iterazione, vengono elaborati i callback delle iterazioni precedenti, come mostrato nei log.

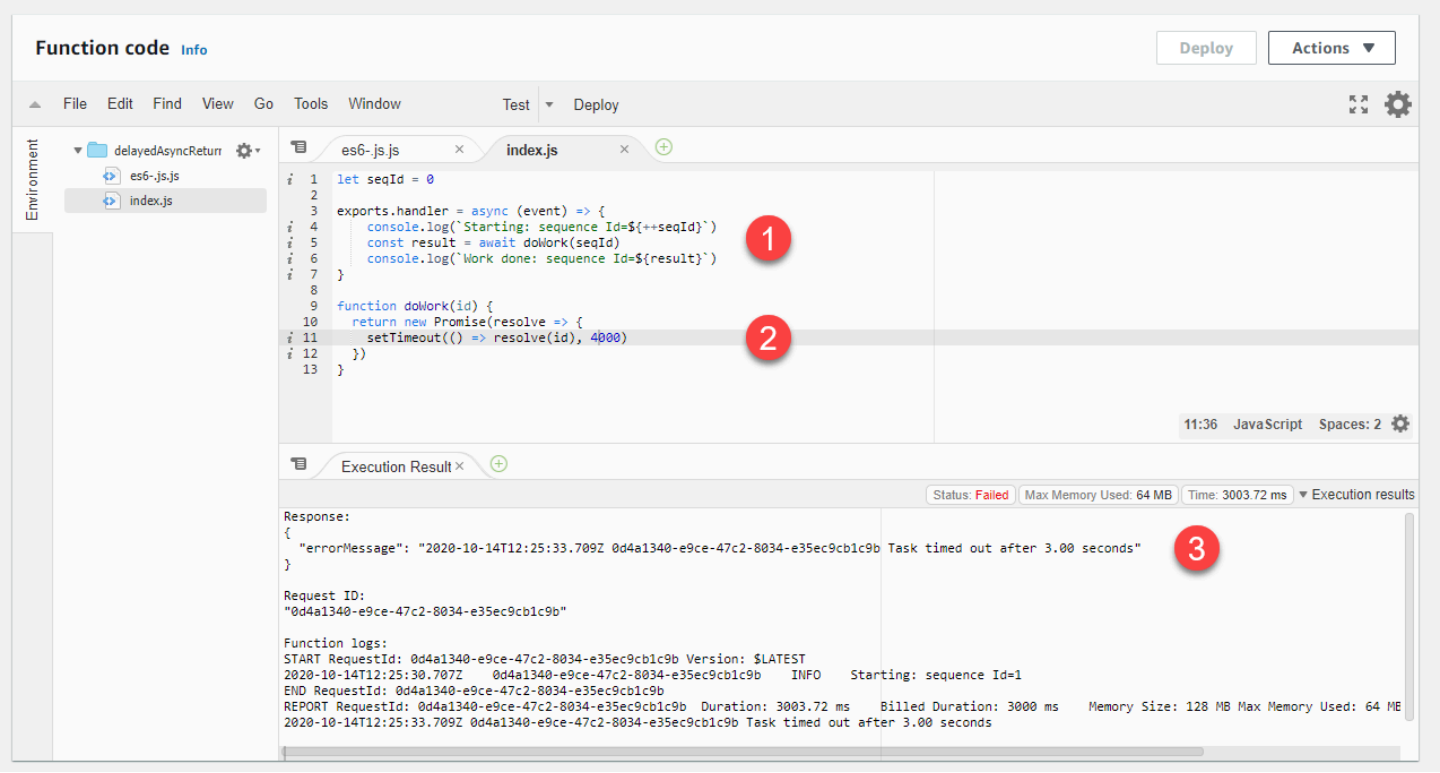
In JavaScript, i callback asincroni vengono gestiti con un ciclo di eventi. Altri runtime utilizzano meccanismi diversi per gestire la simultaneità. Quando l'ambiente di esecuzione della funzione termina, il servizio Lambda blocca l'ambiente fino all'invocazione successiva. Dopo la ripresa, JavaScript continua a elaborare il ciclo di eventi, che in questo caso include un callback asincrono da una invocazione precedente. Senza questo contesto, può sembrare che la funzione esegua codice senza motivo e restituisca dati arbitrari. In effetti, è davvero un artefatto del modo in cui la simultaneità di runtime e gli ambienti di esecuzione interagiscono.

Ciò crea la possibilità che i dati privati di una invocazione precedente vengano visualizzati in una invocazione successiva. Esistono due modi per prevenire o rilevare questo comportamento. Innanzitutto, JavaScript fornisce le parole chiave async e await per semplificare lo sviluppo asincrono e inoltre forza l'esecuzione del codice ad attendere il completamento di una chiamata asincrona. La funzione precedente può essere riscritta utilizzando questo approccio nel modo seguente:

```
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)
  })
}
```

L'utilizzo di questa sintassi impedisce all'handler di uscire prima che la funzione asincrona sia terminata. In questo caso, se il callback impiega più tempo del timeout della funzione Lambda, la funzione genererà un errore invece di restituire il risultato del callback in una invocazione successiva:



1. Il codice richiama la funzione doWork asincrona utilizzando la parola chiave await nell'handler.
2. Il completamento della funzione doWork richiede un certo periodo di tempo prima di risolvere la promessa.
3. La funzione scade perché doWork impiega più tempo del limite di timeout consentito e il risultato del callback non viene restituito in una invocazione successiva.

Di solito, è necessario accertarsi che tutti i processi in background e le callback nel codice vengano completate prima che il codice sia terminato. Se ciò non è possibile nel tuo caso d'uso, puoi utilizzare un identificatore per assicurarti che il callback appartenga all'invocazione corrente. A tale scopo, è possibile utilizzare l'awsRequestId fornito dall'oggetto del contesto. Passando questo valore al callback asincrono, è possibile confrontare il valore passato con il valore corrente per rilevare se il callback ha avuto origine da un'altra invocazione:

```
let currentContext

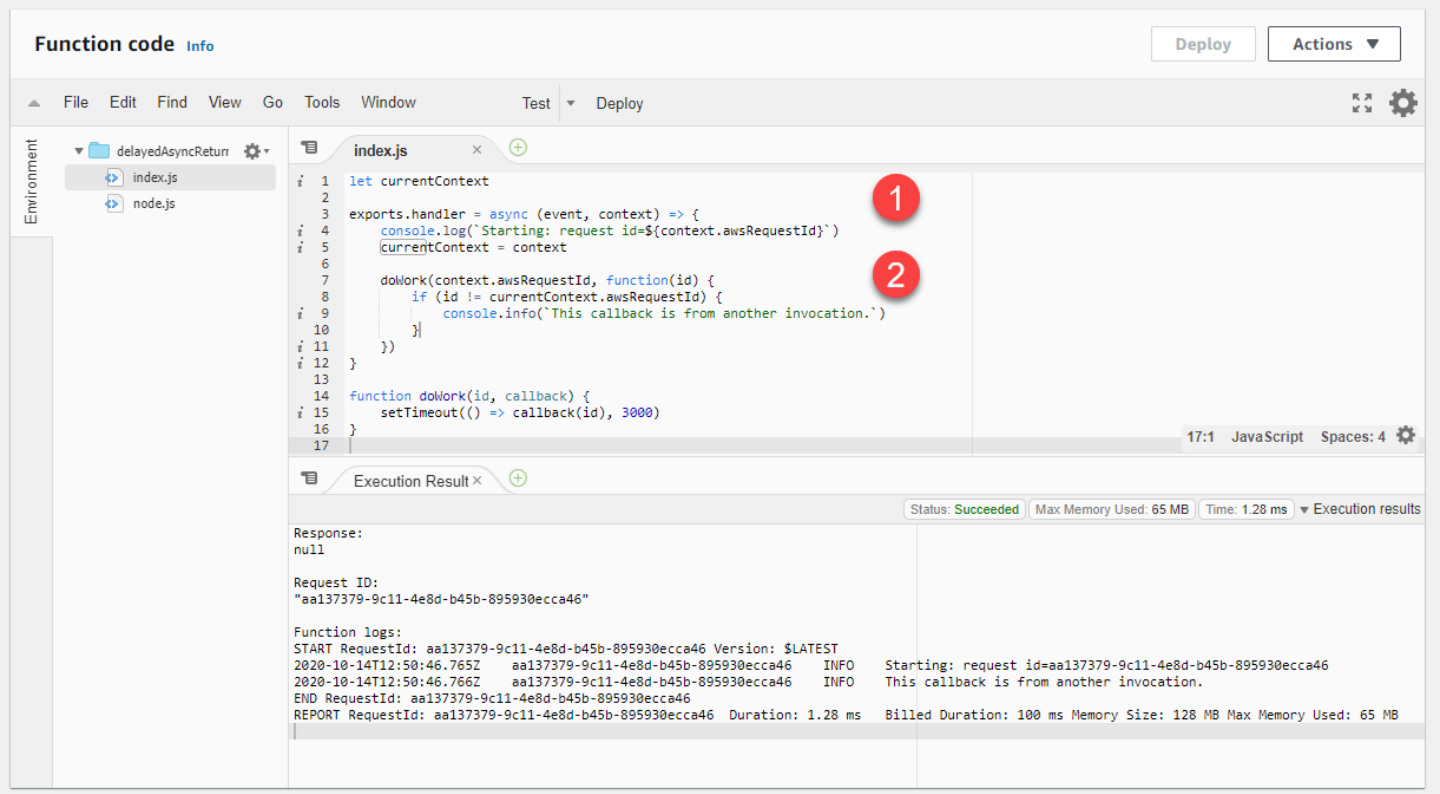
exports.handler = async (event, context) => {
    console.log(`Starting: request id=$\{context.awsRequestId}`)
    currentContext = context

    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. L'handler della funzione Lambda accetta il parametro context, che fornisce l'accesso a un ID di richiesta di invocazione univoco.
2. awsRequestId viene passato alla funzione doWork. Nel callback, l'ID viene confrontato con l'awsRequestId dell'invocazione corrente. Se questi valori sono diversi, il codice può agire di conseguenza.

Risoluzione dei problemi relativi alle implementazioni in Lambda

Quando si aggiorna la funzione, Lambda distribuisce la modifica avviando nuove istanze della funzione con il codice o le impostazioni aggiornati. Gli errori di distribuzione impediscono l'utilizzo della nuova versione e possono derivare da problemi relativi al pacchetto di distribuzione, al codice, alle autorizzazioni o agli strumenti.

Quando distribuisci gli aggiornamenti alla tua funzione direttamente con l'API Lambda o con un client come AWS CLI il, puoi vedere gli errori di Lambda direttamente nell'output. Se utilizzi servizi come AWS CloudFormation, o AWS CodeDeploy AWS CodePipeline, cerca la risposta di Lambda nei log o nel flusso di eventi per quel servizio.

Negli argomenti seguenti vengono forniti suggerimenti per la risoluzione dei problemi relativi a errori e problemi che potrebbero verificarsi durante l'utilizzo della console, degli strumenti o dell'API Lambda. Se scopri un problema che non è elencato qui di seguito, puoi utilizzare il pulsante Feedback in questa pagina per segnalarlo.

Per ulteriori suggerimenti sulla risoluzione dei problemi e per risposte a domande comuni relative al supporto, visitare il Knowledge Center di AWS.

Per ulteriori informazioni sul debug e la risoluzione dei problemi delle applicazioni Lambda, consulta Debug in Serverless Land.

Argomenti

- Generale: autorizzazione negata/Impossibile caricare tale file
- Generale: si verifica un errore quando si chiama il UpdateFunctionCode
- Amazon S3: codice di errore. PermanentRedirect
- Generale: impossibile trovare, impossibile caricare, impossibile importare, classe non trovata, file o directory non trovati
- Generale: handler di metodi non definito
- Generale: il limite di memorizzazione del codice Lambda è stato superato
- Lambda: conversione dei livelli non riuscita
- Lambda: o InvalidParameterValueException RequestEntityTooLargeException
- Lambda: InvalidParameterValueException
- Lambda: quote di simultaneità e memoria
- Lambda: configurazione alias non valida per la concorrenza fornita

Generale: autorizzazione negata/Impossibile caricare tale file

Errore: EACCES: autorizzazione negata, apri '/ .js' var/task/index

Errore: cannot load such file -- function (impossibile caricare tale file - funzione)

Errore: [Errno 13] Autorizzazione negata: '/ .py' var/task/function

Il runtime Lambda necessita dell'autorizzazione per leggere i file nel pacchetto di distribuzione. Nella notazione ottale delle autorizzazioni Linux, Lambda richiede 644 permessi per i file non eseguibili (rw-r--r--) e 755 permessi () per le directory e i file eseguibili. rwxr-xr-x

In Linux e macOS, utilizza il comando chmod per modificare le autorizzazioni file su file e directory nel pacchetto di implementazione. Ad esempio, per assegnare a un file non eseguibile le autorizzazioni corrette, utilizza il comando seguente.

```
chmod 644 <filepath>
```

Per modificare le autorizzazioni file in Windows, consulta Set, View, Change, or Remove Permissions on an Object nella documentazione di Microsoft Windows.

> **Note**
>
> Se non concedi a Lambda le autorizzazioni necessarie per accedere alle directory nel pacchetto di distribuzione, Lambda imposta le autorizzazioni per tali directory su 755 (). rwxr-xr-x

Generale: si verifica un errore quando si chiama il UpdateFunctionCode

Errore: si è verificato un errore (RequestEntityTooLargeException) durante la chiamata dell' UpdateFunctionCode operazione

Quando carichi un pacchetto di distribuzione o un archivio di livelli direttamente in Lambda, la dimensione del file ZIP è limitata a 50 MB. Per caricare un file di dimensioni maggiori, archivialo in Amazon S3 e utilizza i parametri S3Bucket e S3Key.

> **Note**
>
> Quando caricate un file direttamente con AWS SDK o in altro modo AWS CLI, il file ZIP binario viene convertito in base64, il che ne aumenta le dimensioni di circa il 30%. Per consentire questa operazione e la dimensione di altri parametri nella richiesta, il limite effettivo della dimensione della richiesta Lambda applicabile è maggiore. Per questo motivo, il limite di 50 MB è approssimativo.

Amazon S3: codice di errore. PermanentRedirect

Errore: si è verificato un errore durante GetObject. Codice di errore S3: PermanentRedirect. Messaggio di errore S3: il bucket si trova in questa regione: us-east-2. Utilizza questa regione per riprovare la richiesta

Quando carichi il pacchetto di distribuzione di una funzione da un bucket Amazon S3, il bucket deve trovarsi nella stessa regione della funzione. Questo problema può verificarsi quando specifichi un oggetto Amazon S3 in una chiamata o utilizzi il pacchetto e distribuisci i comandi nella o AWS CLI CLI. UpdateFunctionCode AWS SAM Crea un bucket artefatto di distribuzione per ogni regione in cui sviluppi applicazioni.

Generale: impossibile trovare, impossibile caricare, impossibile importare, classe non trovata, file o directory non trovati

Errore: impossibile trovare il modulo "function"

Errore: cannot load such file -- function (impossibile caricare tale file - funzione)

Errore: impossibile importare il modulo "function"

Errore: classe non trovata: Function.Handler

Errorefork/exec /var/task/function: nessun file o directory di questo tipo

Errore: impossibile caricare il tipo "Function.Handler" dal gruppo "Function".

Il nome del file o della classe nella configurazione dell'handler della funzione non corrisponde al codice. Per ulteriori informazioni, consulta la sezione seguente.

Generale: handler di metodi non definito

Errore: index.handler non è definito o non esportato

Errore: Handler "handler" mancante sul modulo "function"

Errore: metodo `handler' non definito per #<:0x000055b76ccebf98> LambdaHandler

Errore: Nessun metodo pubblico denominato HandleRequest con firma del metodo appropriata trovato sulla classe function.Handler

Errore: impossibile trovare il metodo "HandleRequest" nel tipo "Function.Handler" dal gruppo "Function"

Il nome del metodo dell'handler nella configurazione dell'handler della funzione non corrisponde al codice. Ogni runtime definisce una convenzione di denominazione per i gestori, ad esempio.

filename methodname. L'handler è il metodo nel codice della funzione che il runtime esegue quando viene invocata la funzione.

Per alcune lingue, Lambda fornisce una libreria con un'interfaccia che prevede un metodo handler per avere un nome specifico. Per informazioni dettagliate sulla denominazione dell'handler per ogni lingua, consulta i seguenti argomenti.

- Compilazione di funzioni Lambda con Node.js
- Compilazione di funzioni Lambda con Python
- Compilazione di funzioni Lambda con Ruby
- Compilazione di funzioni Lambda con Java
- Compilazione di funzioni Lambda con Go
- Compilazione di funzioni Lambda con C#
- Creazione di funzioni Lambda con PowerShell

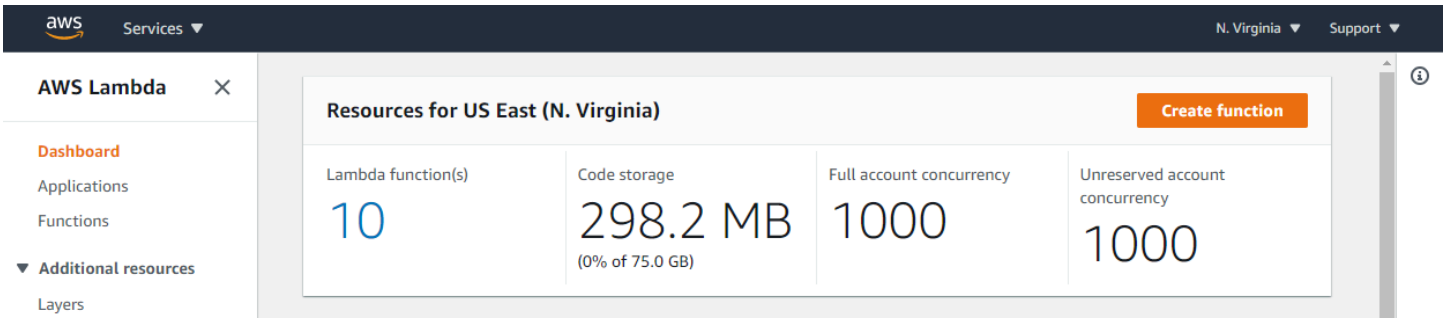
Generale: il limite di memorizzazione del codice Lambda è stato superato

Errore: limite di archiviazione del codice superato.

Lambda archivia il codice della funzione in un bucket S3 interno riservato al tuo account. A ogni AWS account vengono assegnati 75 GB di spazio di archiviazione in ogni regione. Lo spazio di archiviazione del codice include lo spazio di archiviazione totale utilizzato sia dalle funzioni che dai livelli Lambda. Se raggiungi la quota, ricevi un messaggio CodeStorageExceededExceptionquando tenti di implementare nuove funzioni.

Gestisci lo spazio di archiviazione disponibile ripulendo le vecchie versioni delle funzioni, rimuovendo il codice inutilizzato o utilizzando i livelli Lambda. Inoltre, è buona norma utilizzare AWS account separati per carichi di lavoro separati per facilitare la gestione delle quote di archiviazione.

Puoi visualizzare l'utilizzo totale dello storage nella console Lambda, nel sottomenu Dashboard:



Lambda: conversione dei livelli non riuscita

Errore: conversione dei livelli Lambda non riuscita. Per consigli sulla risoluzione di questo problema, consulta la pagina Risoluzione dei problemi di implementazione in Lambda nella Guida per l'utente di Lambda.

Quando si configura una funzione Lambda con un livello, Lambda unisce il livello con il codice della funzione. Se questo processo non viene completato, Lambda restituisce questo errore. Se lo fa, procedere come indicato di seguito:

- Eliminare tutti i file inutilizzati dal livello
- Eliminare tutti i collegamenti simbolici nel tuo livello
- Rinominare tutti i file che hanno lo stesso nome di una directory in uno qualsiasi dei livelli della funzione

Lambda: o InvalidParameterValueException RequestEntityTooLargeException

Errore:InvalidParameterValueException: Lambda non è riuscita a configurare le variabili di ambiente perché le variabili di ambiente fornite hanno superato il limite di 4 KB. Stringa misurata: {"A1":" u Y5 7ATNx5bsm... SFe cyPiPn

ErroreRequestEntityTooLargeException: la richiesta deve essere inferiore a 5120 byte per l'operazione UpdateFunctionConfiguration

La dimensione massima dell'oggetto variabili memorizzato nella configurazione della funzione non deve superare 4096 byte. Sono inclusi nomi chiave, valori, virgolette, virgole e parentesi. Anche la dimensione totale del corpo della richiesta HTTP è limitata.

```
{
    "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"
```

```
        }
    },
    ...
}
```

In questo esempio, l'oggetto è di 39 caratteri e quando archiviato occupa 39 byte (senza spazi) come stringa `{"BUCKET":"amzn-s3-demo-bucket","KEY":"file.txt"}`. I caratteri ASCII standard nei valori delle variabili di ambiente utilizzano un byte ciascuno. I caratteri ASCII e Unicode estesi possono utilizzare tra 2 e 4 byte per carattere.

Lambda: InvalidParameterValueException

Errore:InvalidParameterValueException: Lambda non è riuscita a configurare le variabili di ambiente perché le variabili di ambiente che hai fornito contengono chiavi riservate che attualmente non sono supportate per la modifica.

Lambda riserva alcune chiavi di variabili di ambiente per uso interno. Ad esempio, `AWS_REGION` viene utilizzata dal runtime per determinare la regione corrente e non può essere sovrascritta. Altre variabili, come `PATH`, sono utilizzate dal runtime ma possono essere estese nella configurazione della funzione. Per un elenco completo, consultare Variabili di ambiente di runtime definite.

Lambda: quote di simultaneità e memoria

Errore: la funzione specificata ConcurrentExecutions per la funzione riduce il numero di account al di UnreservedConcurrentExecution sotto del valore minimo

Errore: il valore MemorySize '' non è riuscito a soddisfare il vincolo: il membro deve avere un valore inferiore o uguale a 3008

Questi errori si verificano quando superi le quote di simultaneità o memoria per il tuo account. AWS I nuovi account hanno quote di concorrenza e memoria ridotte. Per risolvere gli errori relativi alla simultaneità, puoi richiedere un aumento della quota. Non è possibile richiedere un aumento della quota.

- Simultaneità: se provi a creare una funzione utilizzando la simultaneità riservata o con provisioning o se la tua richiesta di simultaneità per funzione (PutFunctionConcurrency) supera la quota di simultaneità del tuo account potresti ricevere un errore.
- Memoria: se la quantità di memoria allocata per la funzione supera la quota di memoria dell'account si verificano degli errori.

Lambda: configurazione alias non valida per la concorrenza fornita

Errore: configurazione degli alias non valida per la concorrenza fornita

Questo errore si verifica quando si tenta di aggiornare il codice o la configurazione di una funzione Lambda mentre un alias con concorrenza assegnata indica una versione con problemi. Lambda preinizializza gli ambienti di esecuzione per la concorrenza preimpostata e, se questi ambienti non possono essere inizializzati correttamente a causa di errori di codice, vincoli di risorse o stack e alias interessati, l'implementazione fallisce. Se riscontri questo problema, procedi nel seguente modo:

1. Ripristina l'alias: aggiorna temporaneamente l'alias in modo che punti alla versione funzionante in precedenza.

   ```
   aws lambda update-alias \
    --function-name <function-name> \
    --name <alias-name> \
    --function-version <known-good-version>
   ```

2. Correggi il codice di inizializzazione Lambda: assicurati che il codice di inizializzazione eseguito all'esterno del gestore non contenga eccezioni non rilevate e inizializza i client e le connessioni.
3. Reimplementa la sicurezza: implementa il codice fisso e pubblica una nuova versione. Quindi, aggiorna l'alias in modo che punti alla versione fissa. Facoltativamente, riattiva la concorrenza fornita, se necessario.

Se lo utilizzi AWS CloudFormation, aggiorna la definizione dello stack `FunctionVersion:!GetAtt version.Version` in modo che l'alias punti alla versione funzionante:

```
alias:
 Type: AWS::Lambda::Alias
 Properties:
 FunctionName: !Ref function
FunctionVersion: !GetAtt version.Version
 Name: BLUE
 ProvisionedConcurrencyConfig:
 ProvisionedConcurrentExecutions: 1
```

Risoluzione dei problemi di invocazione in Lambda

Quando si richiama una funzione Lambda, Lambda convalida la richiesta e verifica la capacità di ridimensionamento prima di inviare l'evento alla funzione o, per la invocazione asincrona, alla coda eventi. Gli errori di invocazione possono essere causati da problemi relativi ai parametri di richiesta, alla struttura degli eventi, alle impostazioni delle funzioni, alle autorizzazioni utente, alle autorizzazioni delle risorse o alle restrizioni.

Se invochi direttamente la funzione, visualizzi eventuali errori di invocazione nella risposta da Lambda. Se si richiama la funzione in modo asincrono, con un mapping di origine eventi o tramite un altro servizio, è possibile che vengano riscontrati errori nei log, nella coda DLQ o in una destinazione in caso di errore. Le opzioni di gestione degli errori e il comportamento dei tentativi variano a seconda di come si richiama la funzione e del tipo di errore.

Per un elenco di tipi di errore che l'operazione `Invoke` può restituire, consulta Invoke (Invoca).

Argomenti

- Lambda: timeout della funzione durante la fase di inizializzazione (Sandbox.Timedout)
- IAM: lambda: InvokeFunction non autorizzato
- Lambda: impossibile trovare un bootstrap (Runtime) valido. InvalidEntrypoint)
- Lambda: l'operazione non può essere eseguita ResourceConflictException
- Lambda: la funzione è bloccata in sospeso
- Lambda: una funzione sta usando tutta la simultaneità
- Generale: impossibile richiamare la funzione con altri account o servizi
- Generale: il richiamo della funzione è in loop
- Lambda: routing alias con simultaneità fornita
- Lambda: avvii a freddo con simultaneità fornita
- Lambda: avvii a freddo con nuove versioni
- Lambda: uscita imprevista di Node.js in fase di esecuzione (Runtime). NodejsExit)
- EFS: la funzione non è in grado di montare il file system EFS
- EFS: la funzione non è in grado di connettersi al file system EFS
- EFS: la funzione non è in grado di montare il file system EFS a causa del timeout
- Lambda: Lambda ha rilevato un processo IO che stava impiegando troppo tempo
- Contenitore: errori CodeArtifactUserException

- Contenitore: InvalidEntrypoint errori

Lambda: timeout della funzione durante la fase di inizializzazione (Sandbox.Timedout)

Errore: Timeout dell'attività dopo 3 secondi

Quando la fase Init scade, Lambda inizializza nuovamente l'ambiente di esecuzione rieseguendo la fase `Init` all'arrivo della successiva richiesta di invocazione. Questa operazione è chiamata inizializzazione soppressa. Tuttavia, se la funzione è configurata con una durata di timeout breve (in genere circa 3 secondi), l'init soppresso potrebbe non essere completato durante il periodo di timeout assegnato, causando un nuovo timeout della fase `Init`. In alternativa, l'init soppresso viene completato ma non lascia abbastanza tempo per il completamento della fase Invoke, causando il timeout della fase `Invoke`.

Per ridurre gli errori di timeout, utilizza una delle seguenti strategie:

- Aumenta la durata del timeout della funzione: estendi il timeout per dare più tempo per il completamento delle fasi `Init` e `Invoke`.
- Aumenta l'allocazione della memoria della funzione: una maggiore quantità di memoria significa anche una maggiore allocazione proporzionale della CPU, che può velocizzare entrambe le fasi `Init` e `Invoke`.
- Ottimizza il codice di inizializzazione della funzione: riduci il tempo necessario per l'inizializzazione per garantire che le fasi `Init` e `Invoke` possano essere completate entro il timeout configurato.

IAM: lambda: InvokeFunction non autorizzato

Errore: User: arn:aws:iam: :123456789012:user/developer non è autorizzato a eseguire: lambda: on resource: my-function InvokeFunction

L'utente, o il ruolo che assumi, deve disporre dell'autorizzazione per invocare una funzione. Questo requisito si applica anche alle funzioni Lambda e ad altre risorse di calcolo che richiamano funzioni. Aggiungi il AWSLambdaruolo della policy AWS gestita al tuo utente o aggiungi una politica personalizzata che consenta l'azione sulla funzione di destinazione. `lambda:InvokeFunction`

> **Note**
>
> Il nome dell'azione IAM (`lambda:InvokeFunction`) si riferisce all'operazione dell'API di Lambda `Invoke`.

Per ulteriori informazioni, consulta Gestione delle autorizzazioni in AWS Lambda.

Lambda: impossibile trovare un bootstrap (Runtime) valido. InvalidEntrypoint)

Errore: impossibile trovare bootstrap validi: [/var/task/bootstrap /opt/bootstrap]

Questo errore si verifica in genere quando la root del pacchetto di implementazione non contiene un file eseguibile denominato `bootstrap`. Ad esempio, se desideri implementare una funzione `provided.al2023` con un file .zip, il file `bootstrap` deve trovarsi nella root del file .zip, non in una directory.

Lambda: l'operazione non può essere eseguita ResourceConflictException

ErroreResourceConflictException: l'operazione non può essere eseguita in questo momento. La funzione è attualmente nello stato seguente: Pending (In sospeso)

Quando si connette una funzione a un VPC al momento della creazione, la funzione entra in uno stato `Pending` mentre Lambda crea le interfacce di rete elastica. Durante questo periodo, non è possibile richiamare o modificare la funzione. Se si connette la funzione a un VPC dopo la creazione, è possibile richiamarla mentre l'aggiornamento è in sospeso, ma non è possibile modificarne il codice o la configurazione.

Per ulteriori informazioni, consulta Stati funzione Lambda.

Lambda: la funzione è bloccata in sospeso

Errore: una funzione è bloccata nello stato *`Pending`* per diversi minuti.

Se una funzione è bloccata nello stato `Pending` per più di sei minuti, invoca una delle seguenti operazioni API per sbloccarla.

- UpdateFunctionCode
- UpdateFunctionConfiguration

- PublishVersion

Lambda annulla l'operazione in sospeso e imposta la funzione allo stato `Failed`. Puoi tentare, quindi, un altro aggiornamento.

Lambda: una funzione sta usando tutta la simultaneità

Problema: una funzione utilizza tutta la simultaneità disponibile, causando la limitazione di altre funzioni.

Per dividere la concorrenza disponibile del tuo AWS account in una AWS regione in pool, utilizza la concorrenza riservata. La simultaneità riservata garantisce che una funzione possa sempre scalare alla relativa simultaneità assegnata e che non scalerà oltre la simultaneità assegnata.

Generale: impossibile richiamare la funzione con altri account o servizi

Problema: è possibile invocare la funzione direttamente, ma questa non viene eseguita quando un altro servizio o account la invoca.

Puoi concedere ad altri servizi e account l'autorizzazione a invocare una funzione nella policy basata sulle risorse della funzione. Se l'invoker si trova in un altro account, tale utente deve anche disporre dell'autorizzazione per invocare le funzioni.

Generale: il richiamo della funzione è in loop

Problema: la funzione viene richiamata continuamente in un loop.

Ciò si verifica in genere quando la funzione gestisce le risorse nello stesso AWS servizio che la attiva. Ad esempio, è possibile creare una funzione che memorizza un oggetto in un bucket Amazon Simple Storage Service (Amazon S3) configurato con una notifica che richiama nuovamente la funzione. Per interrompere l'esecuzione della funzione, riduci la simultaneità a zero, il che limita tutte le invocazioni future. Quindi identificare il percorso del codice o l'errore di configurazione che ha causato la chiamata ricorsiva. Lambda rileva e interrompe automaticamente i loop ricorsivi per alcuni servizi e. AWS SDKs Per ulteriori informazioni, consulta the section called “Rilevamento di un ciclo ricorsivo”.

Lambda: routing alias con simultaneità fornita

Problema: invocazioni spillover di concorrenza con provisioning durante il routing degli alias.

Lambda utilizza un modello probabilistico semplice per distribuire il traffico tra le due versioni delle funzioni. A livelli di traffico bassi, è possibile che si verifichi una variazione elevata tra la percentuale di traffico configurata e quella effettiva in ciascuna versione. Se la tua funzione utilizza la concorrenza con provisioning, puoi evitare invocazioni spillover configurando un numero maggiore di istanze di concorrenza sottoposte a provisioning durante il periodo in cui il routing degli alias è attivo.

Lambda: avvii a freddo con simultaneità fornita

Problema: si verificano avvii a freddo dopo che è stata abilitata la simultaneità fornita.

Quando il numero di esecuzioni simultanee su una funzione è minore o uguale al livello configurato di simultaneità fornita, non dovrebbero verificarsi avvii a freddo. Per confermare se la simultaneità fornita funziona normalmente, effettuare le seguenti operazioni:

- Verificare che la simultaneità fornita sia abilitata sulla versione o sull'alias della funzione.

  > **Note**
  >
  > La simultaneità fornita non è configurabile nella versione non pubblicata della funzione ($LATEST).

- Assicurarsi che i trigger richiamino la versione o l'alias della funzione corretti. Ad esempio, se si utilizza Amazon API Gateway, verificare che API Gateway richiami la versione o l'alias della funzione con la simultaneità fornita, non $LATEST. Per confermare che viene utilizzata la concorrenza fornita, puoi controllare la metrica di ProvisionedConcurrencyInvocations Amazon CloudWatch . Un valore diverso da zero indica che la funzione sta elaborando le chiamate in ambienti di esecuzione inizializzati.
- Verifica se la concorrenza delle funzioni supera il livello configurato di concorrenza fornita controllando la metrica. ProvisionedConcurrencySpilloverInvocations CloudWatch Un valore diverso da zero indica che tutta la simultaneità fornita è in uso e che si è verificata una chiamata con un avvio a freddo.
- Verifica la frequenza di chiamata (richieste al secondo). Le funzioni con simultaneità fornita hanno un tasso massimo di 10 richieste al secondo per ogni simultaneità fornita. Ad esempio, una funzione configurata con 100 simultaneità fornita può gestire 1.000 richieste al secondo. Se la frequenza di chiamata supera le 1.000 richieste al secondo, è possibile che si verifichino alcuni avvii a freddo.

Lambda: avvii a freddo con nuove versioni

Problema: si verificano avvii a freddo durante la distribuzione di nuove versioni della funzione.

Quando si aggiorna un alias di funzione, Lambda sposta automaticamente la simultaneità fornita alla nuova versione in base ai pesi configurati sull'alias.

Errore: KMSDisabled Eccezione: Lambda non è riuscita a decrittografare le variabili di ambiente perché la chiave KMS utilizzata è disabilitata. Controlla le impostazioni delle chiavi KMS della funzione.

Questo errore può verificarsi se la chiave AWS Key Management Service (AWS KMS) è disabilitata o se la concessione che consente a Lambda di utilizzare la chiave viene revocata. Se l'autorizzazione manca, configurare la funzione per utilizzare una chiave diversa. Quindi, riassegnare la chiave personalizzata per ricreare l'autorizzazione.

Lambda: uscita imprevista di Node.js in fase di esecuzione (Runtime). NodejsExit)

Problema: il client di runtime Lambda ha rilevato un codice di uscita Node.js imprevisto.

Questo errore si verifica quando la funzione termina prima che tutte le promesse siano state risolte, ad esempio a causa di un bug del codice. Può verificarsi anche quando Node.js rileva una situazione di stallo che impedisce la risoluzione di Promises. Questo errore riguarda solo i gestori in stile asincrono, non i gestori in stile callback.

Runtime interessati: Node.js 18 e versioni successive.

Per risolvere questo problema:

1. Controlla il codice della tua funzione per verificare se ci sono promesse non mantenute nei gestori asincroni.
2. Assicurati che tutte le promesse siano correttamente risolte (risolte o rifiutate) prima del completamento della funzione.
3. Esamina il codice per verificare eventuali condizioni di gara nelle operazioni asincrone.

Per ulteriori informazioni sui codici di uscita di Node.js e sulla terminazione del processo, consultate la documentazione di Node.js.

EFS: la funzione non è in grado di montare il file system EFS

Errore: EFSMountFailureException: La funzione non è riuscita a montare il file system EFS con il punto di accesso arn:aws:elasticfilesystem:us-east- 2:123456789012:access-point/fsap-015cxmplb72b405fd.

La richiesta di montaggio al file system della funzione è stata rifiutata. Verificare le autorizzazioni della funzione e verificare che il file system e il punto di accesso esistano e siano pronti per l'uso.

EFS: la funzione non è in grado di connettersi al file system EFS

Errore: EFSMountConnectivityException: La funzione non è riuscita a connettersi al file system Amazon EFS con punto di accesso arn:aws:elasticfilesystem:us-east- 2:123456789012:access-point/fsap-015cxmplb72b405fd. Controlla la configurazione di rete e riprova.

La funzione non è in grado di stabilire una connessione al file system della funzione con il protocollo NFS (porta TCP 2049). Controllare la configurazione del gruppo di sicurezza e del routing per le sottoreti del VPC.

Se riscontri questi errori dopo aver aggiornato le impostazioni di configurazione VPC della funzione, prova a smontare e rimontare il file system.

EFS: la funzione non è in grado di montare il file system EFS a causa del timeout

Errore: EFSMountTimeoutException: La funzione non è riuscita a montare il file system EFS con il punto di accesso {arn:aws:elasticfilesystem:us-east- 2:123456789012:access-point/fsap-015cxmplb72b405fd} a causa del timeout di montaggio.

La funzione è stata in grado di connettersi al file system della funzione, ma l'operazione di montaggio è scaduta. Riprovare dopo un po' di tempo e considerare di limitare la simultaneità della funzione per ridurre il carico sul file system.

Lambda: Lambda ha rilevato un processo IO che stava impiegando troppo tempo

EFSIOException: questa istanza della funzione è stata interrotta perché Lambda ha rilevato un processo di I/O che impiegava troppo tempo.

Si è verificato il timeout di una chiamata precedente e Lambda non è stata in grado di terminare l'handler della funzione. Questo problema può verificarsi quando un file system collegato esaurisce i crediti di burst e il throughput di base è insufficiente. Per aumentare il throughput, è possibile aumentare le dimensioni del file system o utilizzare il throughput assegnato.

Contenitore: errori CodeArtifactUserException

Errore: messaggio CodeArtifactUserPendingException di errore

L'ottimizzazione CodeArtifact è in sospeso. La funzione passa allo stato Attivo quando Lambda completa l'ottimizzazione. Codice di risposta HTTP: 409.

Errore: messaggio di errore CodeArtifactUserDeletedException

CodeArtifact È programmata per essere eliminata. Codice di risposta HTTP: 409.

Errore: messaggio CodeArtifactUserFailedException di errore

Lambda non è riuscito a ottimizzare il codice. Correggere il codice e caricarlo di nuovo. Codice di risposta HTTP: 409.

Contenitore: InvalidEntrypoint errori

Errore: Runtime. ExitError o «errorType»: «Runtime. InvalidEntrypoint«

Verificare che l'elemento ENTRYPOINT dell'immagine container includa il percorso assoluto come posizione. Verificare inoltre che l'immagine non contenga un collegamento simbolico come ENTRYPOINT.

Errore: stai utilizzando un CloudFormation modello e il contenitore ENTRYPOINT viene sovrascritto con un valore nullo o vuoto.

Esamina la risorsa nel ImageConfigmodello. CloudFormation Se si dichiara una `ImageConfig` risorsa nel modello, è necessario fornire valori non vuoti per tutte e tre le proprietà.

Risoluzione dei problemi di esecuzione in Lambda

Quando il runtime Lambda esegue il codice della funzione, l'evento potrebbe essere elaborato su un'istanza della funzione che ha elaborato eventi per un determinato periodo o potrebbe richiedere

l'inizializzazione di una nuova istanza. Gli errori possono verificarsi durante l'inizializzazione della funzione, quando il codice dell'handler elabora l'evento o quando la funzione restituisce (o non riesce a restituire) una risposta.

Gli errori di esecuzione delle funzioni possono essere causati da problemi relativi al codice, alla configurazione delle funzioni, alle risorse downstream o alle autorizzazioni. Se si invoca direttamente la funzione, si ricevono errori di funzione nella risposta da Lambda. Se si richiama la funzione in modo asincrono, con un mapping di origine eventi o tramite un altro servizio, è possibile che vengano riscontrati errori nei log, nella coda DLQ o in una destinazione in caso di errore. Le opzioni di gestione degli errori e il comportamento dei tentativi variano a seconda di come si richiama la funzione e del tipo di errore.

Quando il codice della funzione o il runtime Lambda restituiscono un errore, il codice di stato nella risposta da Lambda è 200 OK. La presenza di un errore nella risposta è indicata da un'intestazione denominata `X-Amz-Function-Error`. I codici di stato serie 400 e 500 sono riservati agli errori di chiamata.

Argomenti

- Lambda: debug remoto con Visual Studio Code
- Lambda: l'esecuzione richiede troppo tempo
- Lambda: payload per eventi imprevisti
- Lambda: dimensioni del carico utile inaspettatamente grandi
- Lambda: errori di codifica e decodifica JSON
- Lambda: i log o le tracce non vengono visualizzati
- Lambda: non vengono visualizzati tutti i log della mia funzione
- Lambda: la funzione restituisce prima del termine dell'esecuzione
- Lambda: esecuzione di una versione o di un alias di funzione non intenzionali
- Lambda: rilevamento di loop infiniti
- Generale: indisponibilità del servizio downstream
- AWS SDK: versioni e aggiornamenti
- Python: caricamento librerie errato
- Java: la funzione impiega più tempo per elaborare gli eventi dopo l'aggiornamento a Java 17 da Java 11
- Kafka: gestione degli errori e nuovi tentativi di configurazione

Lambda: debug remoto con Visual Studio Code

Problema: difficoltà nella risoluzione dei problemi relativi al comportamento complesso della funzione Lambda nell'ambiente reale AWS

Lambda fornisce una funzionalità di debug remoto tramite. AWS Toolkit for Visual Studio Code Per istruzioni generali e di configurazione, vedi. Esecuzione da remoto del debug delle funzioni Lambda con Visual Studio Code

Per istruzioni dettagliate sulla risoluzione dei problemi, sui casi d'uso avanzati e sulla disponibilità delle regioni, consulta il debug remoto delle funzioni Lambda nella Guida per l'utente. AWS Toolkit for Visual Studio Code

Lambda: l'esecuzione richiede troppo tempo

Problema: l'esecuzione della funzione richiede troppo tempo.

Se l'esecuzione del codice richiede molto più tempo in Lambda rispetto al computer locale, potrebbe essere limitato dalla memoria o dalla potenza di elaborazione disponibile per la funzione. Configurare la funzione con memoria aggiuntiva per aumentare sia la memoria sia la CPU.

Lambda: payload per eventi imprevisti

Problema: errori di funzione relativi a un formato JSON non valido o a una convalida dei dati inadeguata.

Tutte le funzioni Lambda ricevono un payload di eventi nel primo parametro dell'handler. Il payload dell'evento è una struttura JSON che può contenere array ed elementi nidificati.

Il formato JSON non valido può verificarsi se fornito da servizi upstream che non utilizzano un processo affidabile per il controllo delle strutture JSON. Ciò si verifica quando i servizi concatenano stringhe di testo o incorporano input dell'utente che non sono stati eliminati. JSON viene inoltre spesso serializzato per il passaggio da un servizio all'altro. Analizza sempre le strutture JSON sia come producer che come consumer di JSON per assicurarti che la struttura sia valida.

Analogamente, il mancato controllo degli intervalli di valori nel payload dell'evento può causare errori. Nell'esempio seguente viene illustrata una funzione che calcola le ritenute d'acconto:

```
exports.handler = async (event) => {
    let pct = event.taxPct
```

```
    let salary = event.salary

    // Calculate % of paycheck for taxes
    return (salary * pct)
}
```

Questa funzione utilizza lo stipendio e l'aliquota fiscale del payload dell'evento per eseguire il calcolo. Tuttavia, il codice non riesce a verificare se gli attributi sono presenti. Inoltre, non riesce a controllare i tipi di dati o a garantire i limiti, ad esempio garantendo che la percentuale fiscale sia compresa tra 0 e 1. Di conseguenza, i valori al di fuori di questi limiti producono risultati privi di senso. Un tipo errato o un attributo mancante causa un errore di runtime.

Crea test per assicurarti che la tua funzione gestisca dimensioni di payload maggiori. La dimensione massima per il payload di un evento Lambda è 1 MB. A seconda del contenuto, payload più grandi possono significare più elementi passati alla funzione o più dati binari incorporati in un attributo JSON. In entrambi i casi, ciò può comportare una maggiore elaborazione per una funzione Lambda.

Payload più grandi possono anche causare dei timeout. Ad esempio, una funzione Lambda elabora un record ogni 100 ms e ha un timeout di 3 secondi. L'elaborazione ha esito positivo per 0-29 elementi nel payload. Tuttavia, se il payload contiene più di 30 elementi, la funzione scade e genera un errore. Per evitare ciò, assicurati che siano impostati dei timeout per gestire il tempo di elaborazione aggiuntivo per il numero massimo di elementi previsto.

Lambda: dimensioni del carico utile inaspettatamente grandi

Problema: le funzioni si stanno esaurendo o causano errori a causa di carichi utili di grandi dimensioni.

I payload più grandi possono causare timeout ed errori. Ti consigliamo di creare test per garantire che la funzione gestisca i payload più elevati previsti e per assicurarti che il timeout della funzione sia impostato correttamente.

Inoltre, alcuni payload di eventi possono contenere puntatori ad altre risorse. Ad esempio, una funzione Lambda con 128 MB di memoria può eseguire l'elaborazione delle immagini su un file JPG archiviato come oggetto in S3. La funzione funziona come previsto con file di immagine più piccoli. Tuttavia, quando viene fornito un file JPG più grande come input, la funzione Lambda genera un errore dovuto all'esaurimento della memoria. Per evitare ciò, i casi di test dovrebbero includere esempi tratti dai limiti superiori delle dimensioni dei dati previste. Il codice dovrebbe anche convalidare le dimensioni del payload.

Lambda: errori di codifica e decodifica JSON

Problema: NoSuchKey eccezione durante l'analisi degli input JSON.

Assicurati di elaborare correttamente gli attributi JSON. Ad esempio, per gli eventi generati da S3, l'`s3.object.key`attributo contiene un nome chiave dell'oggetto con codifica URL. Molte funzioni elaborano questo attributo come testo per caricare l'oggetto S3 di riferimento:

Example

```
const originalText = await s3.getObject({
  Bucket: event.Records[0].s3.bucket.name,
  Key: event.Records[0].s3.object.key
}).promise()
```

Questo codice funziona con il nome della chiave `james.jpg` ma genera un errore `NoSuchKey` quando il nome è `james beswick.jpg`. Poiché la codifica URL converte gli spazi e gli altri caratteri in un nome chiave, è necessario assicurarsi che le funzioni decodifichino le chiavi prima di utilizzare questi dati:

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: i log o le tracce non vengono visualizzati

Problema: i log non vengono visualizzati in Logs. CloudWatch

Problema: le tracce non vengono visualizzate in. AWS X-Ray

La tua funzione richiede l'autorizzazione per chiamare CloudWatch Logs e X-Ray. Aggiornare il ruolo di esecuzione per concedere a esso l'autorizzazione. Aggiungere le policy gestite seguenti per abilitare i registri e l'analisi.

- AWSLambdaBasicExecutionRole
- AWSXRayDaemonWriteAccess

Quando aggiungi le autorizzazioni alla funzione, aggiorni anche il relativo codice o la configurazione. Questa operazione forza l'arresto e la sostituzione delle istanze della funzione in esecuzione che hanno credenziali obsolete.

> **Note**
>
> Potrebbero essere necessari da 5 a 10 minuti prima che i log vengano visualizzati dopo una chiamata di funzione.

Lambda: non vengono visualizzati tutti i log della mia funzione

Problema: i registri delle funzioni non sono presenti in CloudWatch Logs, anche se le mie autorizzazioni sono corrette

Se Account AWS raggiungi i limiti di quota dei CloudWatch log, CloudWatch rallenta la registrazione delle funzioni. Quando ciò accade, alcuni dei log generati dalle tue funzioni potrebbero non apparire in Logs. CloudWatch

Se la funzione emette i log a una velocità troppo elevata per consentire a Lambda di elaborarli, ciò può anche far sì che gli output di log non vengano visualizzati in Logs. CloudWatch Quando Lambda non riesce a inviare i log CloudWatch alla velocità con cui vengono prodotti dalla funzione, li elimina per evitare che l'esecuzione della funzione rallenti. Aspettatevi di osservare costantemente i log eliminati quando la velocità effettiva dei log supera 2 per un singolo flusso di log. MB/s

Se la funzione è configurata per utilizzare log in formato JSON, Lambda tenta di inviare un `logsDropped`evento a CloudWatch Logs quando elimina i log. Tuttavia, quando CloudWatch limita la registrazione della funzione, questo evento potrebbe non raggiungere CloudWatch Logs, quindi non vedrai sempre un record quando Lambda elimina i log.

Per verificare se hai Account AWS raggiunto i limiti di quota di CloudWatch Logs, procedi come segue:

1. Apri la console Service Quotas.
2. Nel pannello di navigazione, scegli Servizi AWS (servizi AWS).
3. Dall'elenco dei AWS servizi, cerca Amazon CloudWatch Logs.
4. Nell'elenco Service Quotas, scegli le quote `CreateLogGroup throttle limit in transactions per second`, `CreateLogStream throttle limit in transactions`

`per second` e `PutLogEvents throttle limit in transactions per second` per visualizzarne l'utilizzo.

Puoi anche impostare CloudWatch allarmi per avvisarti quando l'utilizzo del tuo account supera un limite specificato per queste quote. Per ulteriori informazioni, consulta Creare un CloudWatch allarme basato su una soglia statica.

Se i limiti di quota predefiniti per CloudWatch i registri non sono sufficienti per il tuo caso d'uso, puoi richiedere un aumento della quota.

Lambda: la funzione restituisce prima del termine dell'esecuzione

Problema: (Node.js) La funzione restituisce prima che il codice finisca l'esecuzione

Molte librerie, incluso l' AWS SDK, funzionano in modo asincrono. Quando si effettua una chiamata di rete o si esegue un'altra operazione che richiede l'attesa di una risposta, le librerie restituiscono un oggetto chiamato promessa che tiene traccia dell'avanzamento dell'operazione in background.

Per attendere che la promessa si risolva in una risposta, utilizzare la parola chiave `await`. Questo blocca l'esecuzione del codice dell'handler fino a quando la promessa non viene risolta in un oggetto che contiene la risposta. Se non è necessario utilizzare i dati della risposta nel codice, è possibile restituire la promessa direttamente al runtime.

Alcune librerie non restituiscono promesse ma possono essere racchiuse in codice che lo fa. Per ulteriori informazioni, consulta Definire l'handler delle funzioni Lambda in Node.js.

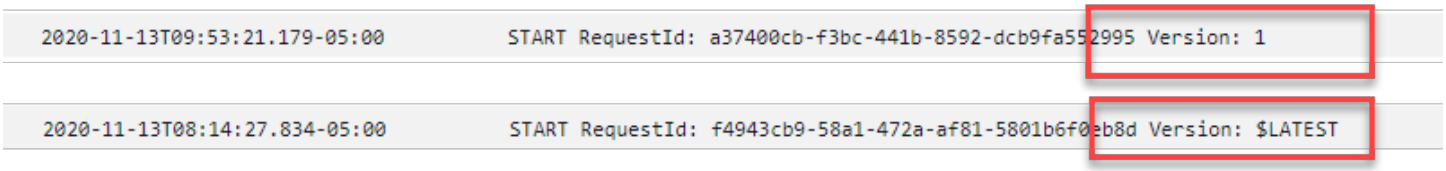
Lambda: esecuzione di una versione o di un alias di funzione non intenzionali

Problema: la versione o l'alias della funzione non sono stati richiamati

Quando si pubblicano nuove funzioni Lambda nella console o in uso AWS SAM, la versione più recente del codice è rappresentata da. `$LATEST` Per impostazione predefinita, le chiamate che non specificano una versione o un alias indirizzano automaticamente alla `$LATEST` versione del codice della funzione.

Se si utilizzano versioni o alias di funzioni specifiche, si tratta in aggiunta a versioni pubblicate immutabili di una funzione. `$LATEST` Durante la risoluzione dei problemi relativi a queste funzioni,

verificate innanzitutto che il chiamante abbia richiamato la versione o l'alias previsti. È possibile farlo controllando i registri delle funzioni. La versione della funzione che è stata richiamata viene sempre mostrata nella riga di registro START:

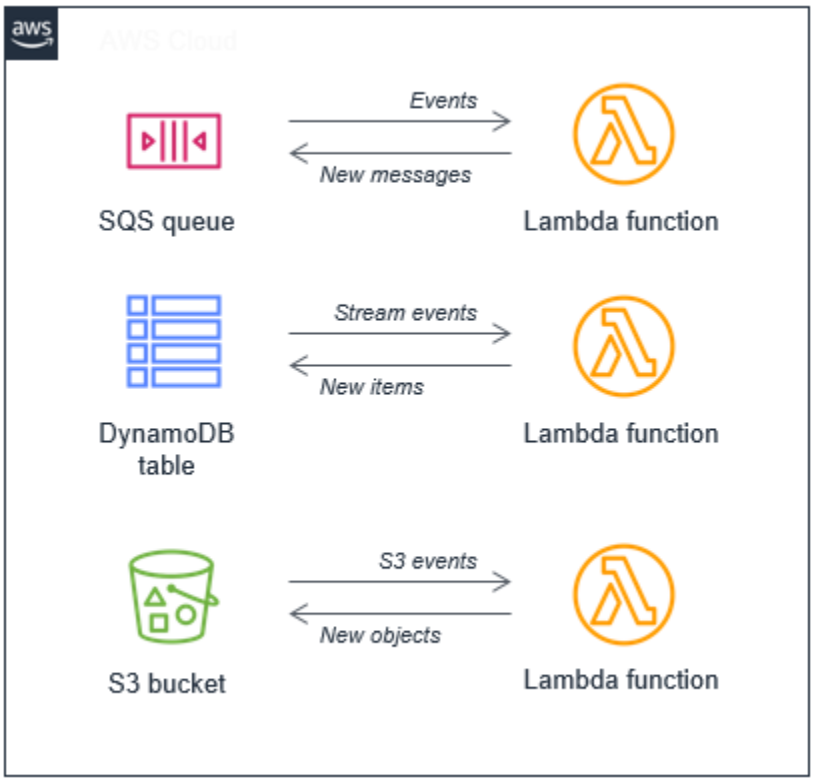


Lambda: rilevamento di loop infiniti

Problema: modelli di loop infiniti relativi alle funzioni Lambda

Esistono due tipi di cicli infiniti nelle funzioni Lambda. Il primo è all'interno della funzione stessa, causato da un ciclo che non esce mai. L'invocazione termina solo quando la funzione scade. È possibile identificarli monitorando i timeout e quindi correggendo il comportamento di loop.

Il secondo tipo di loop è tra le funzioni Lambda e altre AWS risorse. Si verificano quando un evento proveniente da una risorsa come un bucket S3 richiama una funzione Lambda, che quindi interagisce con la stessa risorsa di origine per attivare un altro evento. Ciò richiama nuovamente la funzione, che crea un'altra interazione con lo stesso bucket S3 e così via. Questi tipi di loop possono essere causati da diverse fonti di AWS eventi, tra cui code Amazon SQS e tabelle DynamoDB. Puoi utilizzare il rilevamento ricorsivo dei loop per identificare questi modelli.



Puoi evitare questi loop assicurandoti che le funzioni Lambda scrivano su risorse che non sono le stesse della risorsa che consuma. Se devi pubblicare nuovamente i dati sulla risorsa che consuma, assicurati che i nuovi dati non attivino lo stesso evento. In alternativa, utilizza il filtro degli eventi. Ad esempio, ecco due soluzioni proposte per loop infiniti con risorse S3 e DynamoDB:

- Se riscrivi nello stesso bucket S3, usa un prefisso o suffisso diverso dal trigger dell'evento.
- Se scrivi elementi nella stessa tabella DynamoDB, includi un attributo su cui può filtrare una funzione Lambda che utilizza. Se Lambda trova l'attributo, non genererà un'altra chiamata.

Generale: indisponibilità del servizio downstream

Problema: i servizi downstream su cui si basa la funzione Lambda non sono disponibili

Per le funzioni Lambda che richiamano endpoint di terze parti o altre risorse downstream, assicurati che siano in grado di gestire gli errori e i timeout del servizio. Queste risorse a valle possono avere tempi di risposta variabili o diventare non disponibili a causa di interruzioni del servizio. A seconda dell'implementazione, questi errori downstream possono apparire come timeout Lambda o eccezioni se la risposta all'errore del servizio non viene gestita all'interno del codice della funzione.

Ogni volta che una funzione dipende da un servizio a valle, ad esempio una chiamata API, implementa la gestione degli errori e la logica dei tentativi appropriati. Per i servizi critici, la funzione Lambda deve pubblicare metriche o log su. CloudWatch Ad esempio, se un'API di pagamento di terze parti non è disponibile, la funzione Lambda può registrare queste informazioni. Puoi quindi impostare CloudWatch allarmi per inviare notifiche relative a questi errori.

Poiché Lambda è in grado di scalare rapidamente, i servizi downstream non serverless potrebbero avere difficoltà a gestire i picchi di traffico. Esistono tre approcci comuni per gestire questo problema:

- Memorizzazione nella cache: valuta la possibilità di memorizzare nella cache il risultato dei valori restituiti da servizi di terze parti se non vengono modificati frequentemente. Puoi memorizzare questi valori in una variabile globale nella tua funzione o in un altro servizio. Ad esempio, i risultati di una query sull'elenco di prodotti da un'istanza Amazon RDS potrebbero essere salvati per un periodo di tempo all'interno della funzione per evitare query ridondanti.
- Accodamento: durante il salvataggio o l'aggiornamento dei dati, aggiungi una coda Amazon SQS tra la funzione Lambda e la risorsa. La coda mantiene i dati in modo duraturo mentre il servizio a valle elabora i messaggi.
- Proxy: laddove in genere vengono utilizzate connessioni di lunga durata, ad esempio per le istanze Amazon RDS, utilizza un livello proxy per raggruppare e riutilizzare tali connessioni. Per i database relazionali, Amazon RDS Proxy è un servizio progettato per aiutare a migliorare la scalabilità e la resilienza nelle applicazioni basate su Lambda.

AWS SDK: versioni e aggiornamenti

Problema: l' AWS SDK incluso nel runtime non è la versione più recente

Problema: l' AWS SDK incluso nel runtime si aggiorna automaticamente

I runtime per i linguaggi interpretati includono una versione dell' AWS SDK. Lambda aggiorna periodicamente questi runtime per utilizzare la versione SDK più recente. Per trovare la versione dell'SDK inclusa nel runtime, consulta le seguenti sezioni:

- Versioni SDK incluse nel runtime (Node.js)
- Versioni SDK incluse nel runtime (Python)
- Versioni SDK incluse nel runtime (Ruby)

Per utilizzare una versione più recente dell' AWS SDK o per bloccare le funzioni su una versione specifica, puoi raggruppare la libreria con il codice della funzione o creare un layer Lambda. Per informazioni dettagliate sulla creazione di un pacchetto di distribuzione con dipendenze, vedere i seguenti argomenti:

Node.js

- Distribuisci funzioni Lambda in Node.js con archivi di file .zip

Python

- Utilizzo di archivi di file .zip per le funzioni Lambda in Python

Ruby

- Distribuire le funzioni Ruby Lambda con gli archivi di file .zip

Java

- Distribuisci funzioni Lambda per Java con archivi di file .zip o JAR

Go

- Distribuisci funzioni Lambda per Go con gli archivi di file .zip

C#

- Crea e implementa le funzioni Lambda C# con gli archivi di file .zip

PowerShell

- Distribuzione delle funzioni Lambda di PowerShell con gli archivi di file .zip

Python: caricamento librerie errato

Problema: (Python) alcune librerie non vengono caricate correttamente dal pacchetto di distribuzione

Le librerie con moduli di estensione scritti in C o C ++ devono essere compilate in un ambiente con la stessa architettura del processore di Lambda (Amazon Linux). Per ulteriori informazioni, consulta Utilizzo di archivi di file .zip per le funzioni Lambda in Python.

Java: la funzione impiega più tempo per elaborare gli eventi dopo l'aggiornamento a Java 17 da Java 11

Problema: (Java) la funzione impiega più tempo per elaborare gli eventi dopo l'aggiornamento a Java 17 da Java 11

Ottimizza il compilatore utilizzando il parametro `JAVA_TOOL_OPTIONS`. I runtime Lambda per Java 17 e versioni successive modificano le opzioni predefinite del compilatore. La modifica migliora i tempi di avvio a freddo per le funzioni di breve durata, ma il comportamento precedente è più adatto alle funzioni ad alta intensità di calcolo e di lunga durata. Imposta `JAVA_TOOL_OPTIONS` su `-XX:-TieredCompilation` per ripristinare il comportamento di Java 11. Per ulteriori informazioni sul parametro `JAVA_TOOL_OPTIONS`, vedi the section called “Informazioni sulla variabile di ambiente `JAVA_TOOL_OPTIONS`”.

Kafka: gestione degli errori e nuovi tentativi di configurazione

Problema: la mappatura delle sorgenti degli eventi Kafka non riesce a configurare le impostazioni dei nuovi tentativi o le destinazioni in caso di errore

Le configurazioni Kafka Retry e le destinazioni in caso di errore sono disponibili solo per le mappature delle sorgenti degli eventi con la modalità provisioning abilitata. Assicurati di aver effettuato la configurazione prima di tentare di impostare configurazioni di nuovo `MinimumPollers` tentativo. `ProvisionedPollerConfig`

Errori di configurazione comuni:

- Tentativi infiniti con batch bisect: non è possibile abilitare `BisectBatchOnFunctionError` quando `MaximumRetryAttempts` è impostato su -1 (infinito). Imposta un limite di tentativi finito o disabilita il batch bisettale.
- Ricorsione dello stesso argomento: l'argomento sulla destinazione di Kafka in caso di errore non può essere lo stesso di nessuno degli argomenti di origine. Scegliete un nome diverso per l'argomento a lettere morte.
- Formato di destinazione Kafka non valido: utilizza il **`kafka://<topic-name>`** formato per specificare un argomento di Kafka come destinazione in caso di errore.
- kafka: problemi di WriteData autorizzazione: assicurati che il tuo ruolo di esecuzione disponga delle autorizzazioni per l'argomento di destinazione. `kafka-cluster:WriteData` L'argomento non esiste, eccezioni di timeout o problemi di limitazione dell'API di scrittura potrebbero richiedere un aumento dei limiti dell'account.

Risolvi i problemi di mappatura delle sorgenti degli eventi in Lambda

I problemi in Lambda relativi alla mappatura dell'origine di un evento possono essere più complessi perché implicano il debug su più servizi. Inoltre, il comportamento dell'origine degli eventi può variare in base all'origine esatta dell'evento utilizzata. Questa sezione elenca i problemi comuni relativi alla mappatura delle sorgenti degli eventi e fornisce indicazioni su come identificarli e risolverli.

> **Note**
>
> Questa sezione utilizza un'origine eventi Amazon SQS a scopo illustrativo, ma i principi si applicano ad altre mappature delle fonti degli eventi che mettono in coda i messaggi per le funzioni Lambda.

Identificazione e gestione della limitazione

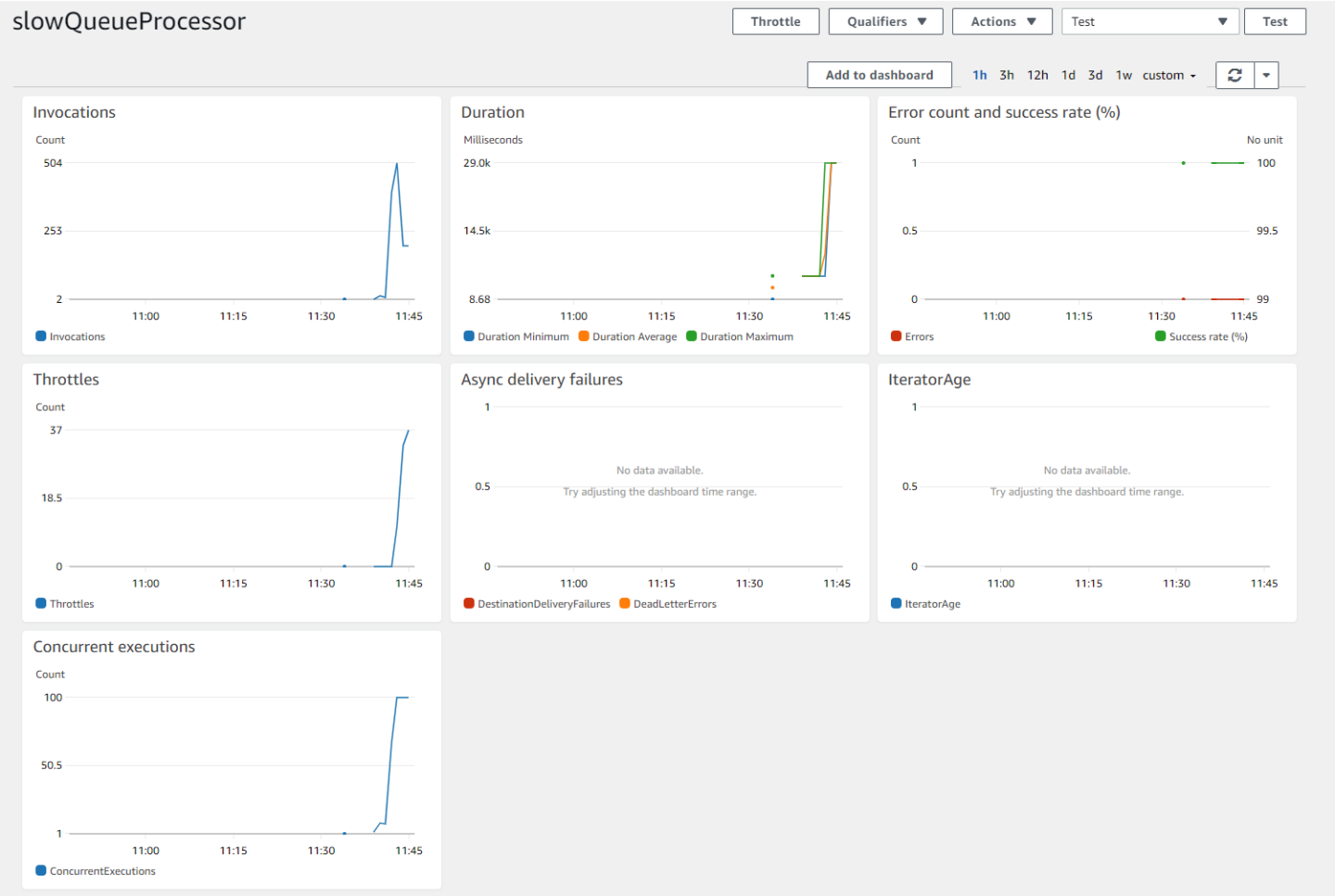
In Lambda, la limitazione si verifica quando si raggiunge il limite di concorrenza della funzione o dell'account. Considera il seguente esempio, in cui esiste una funzione Lambda che legge i messaggi da una coda Amazon SQS. Questa funzione Lambda simula invocazioni di 30 secondi e ha una dimensione batch di 1. Ciò significa che la funzione elabora solo 1 messaggio ogni 30 secondi:

```
const doWork = (ms) => new Promise(resolve => setTimeout(resolve, ms))

exports.handler = async (event) => {
    await doWork(30000)

}
```

Con un tempo di invocazione così lungo, i messaggi iniziano ad arrivare in coda più rapidamente di quanto vengano elaborati. Se la simultaneità non riservata dell'account è 100, Lambda è scalabile fino a 100 esecuzioni simultanee, quindi si verifica una limitazione. Puoi vedere questo modello nei parametri di CloudWatch per la funzione:



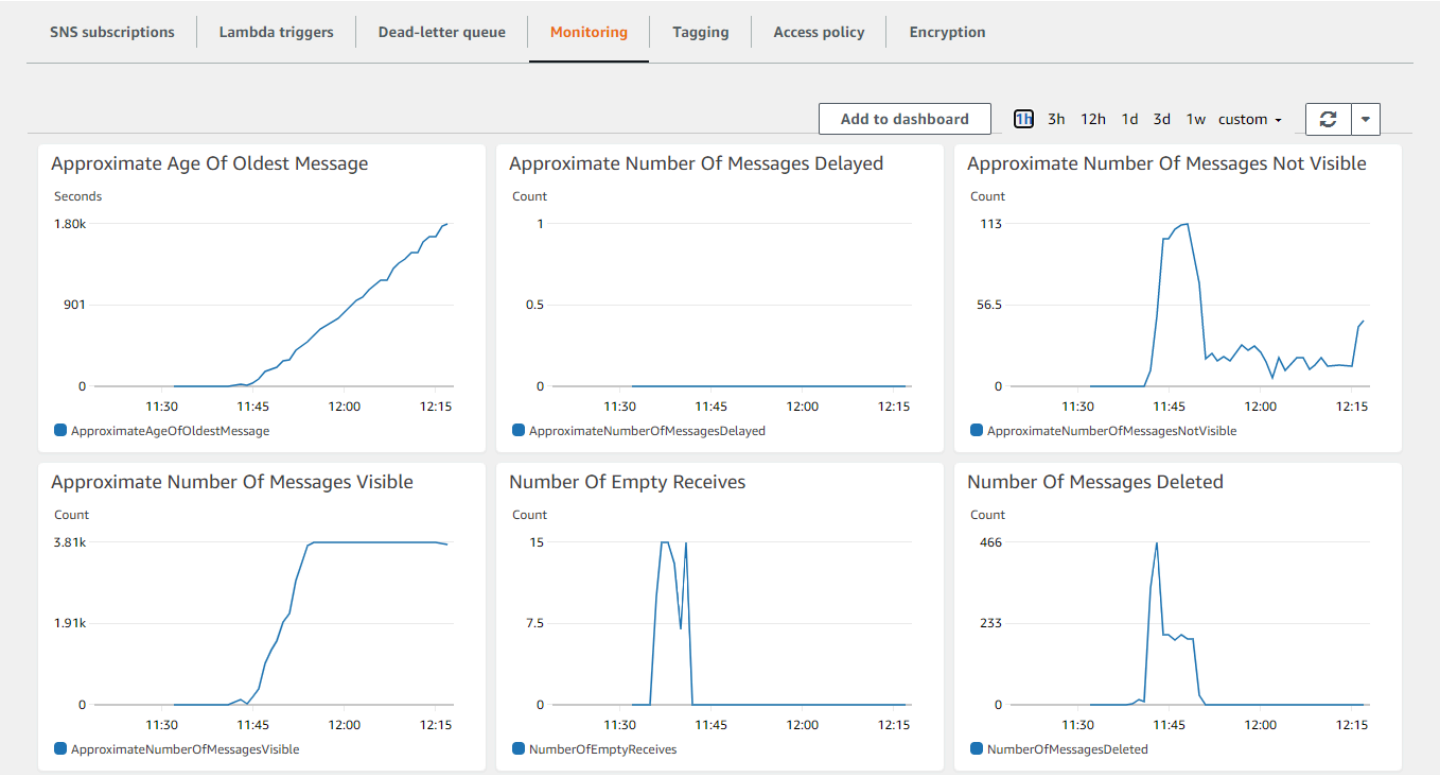
I parametri di CloudWatch per la funzione non mostrano errori, ma il grafico delle esecuzioni simultanee mostra che è stata raggiunta la simultaneità massima. Di conseguenza, il grafico Limitazioni mostra la limitazione in atto.

Puoi rilevare la limitazione utilizzando gli allarmi CloudWatch e impostare un allarme ogni volta che il parametro di limitazione per una funzione è maggiore di 0. Dopo aver identificato il problema della limitazione, hai a disposizione alcune opzioni per la risoluzione:

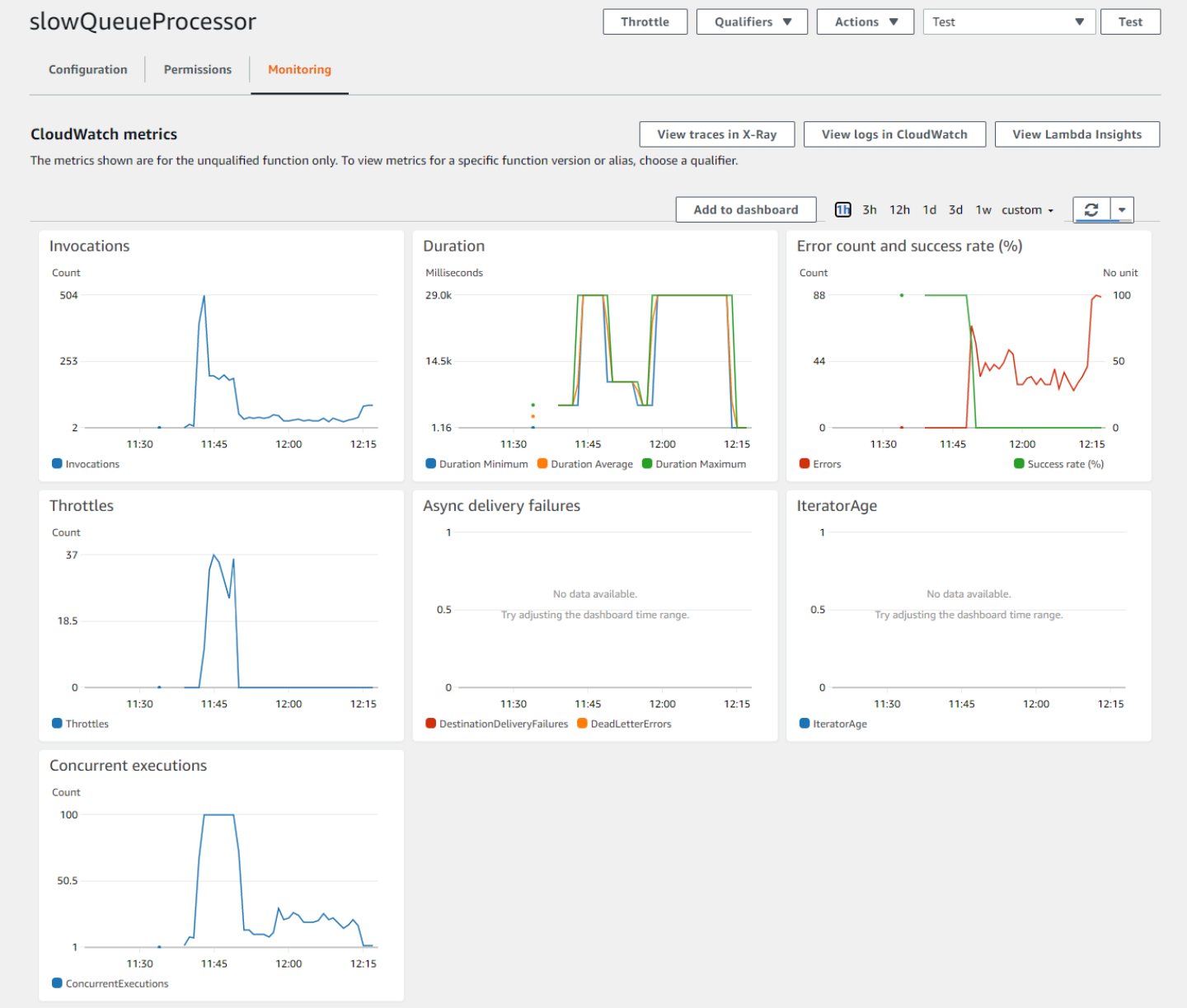
- Richiedere un aumento della simultaneità al Supporto AWS in questa regione.
- Identificare i problemi di prestazioni della funzione per migliorare la velocità di elaborazione e di conseguenza il throughput.
- Aumentare la dimensione del batch della funzione, in modo che a ogni invocazione della funzione vengano elaborati più messaggi.

Errori nella funzione di elaborazione

Se la funzione di elaborazione genera errori, i messaggi vengono restituiti alla coda SQS. Il servizio Lambda impedisce la scalabilità della funzione per evitare errori su larga scala. I seguenti parametri SQS in CloudWatch indicano un problema con l'elaborazione delle code:

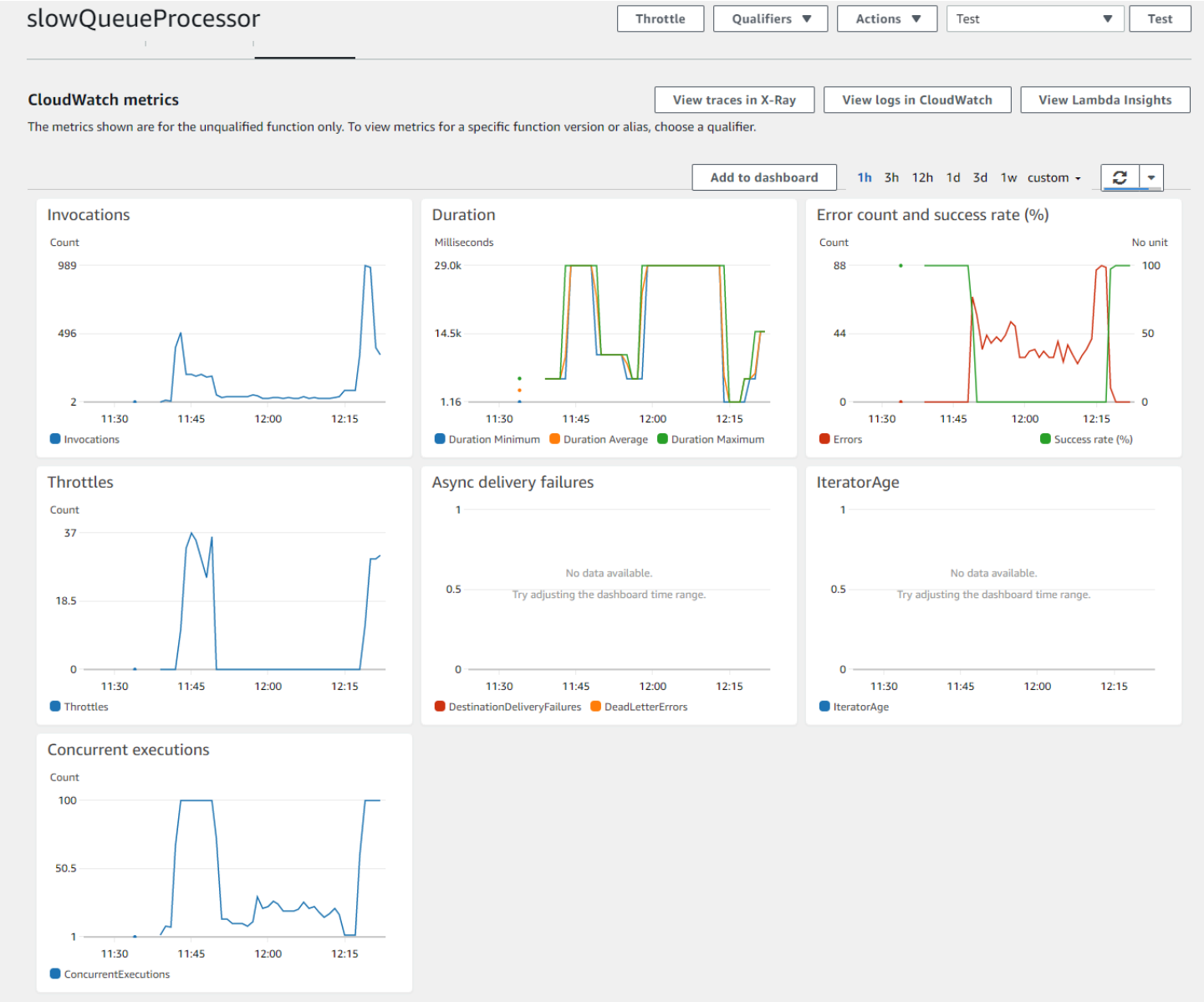


Aumentano sia l'età dei primi messaggi che il numero di messaggi visibili, mentre nessun messaggio viene eliminato. La coda continua a crescere ma i messaggi non vengono elaborati. I parametri CloudWatch per l'elaborazione della funzione Lambda indicano anche che esiste un problema:



Il parametro numero errori è diverso da zero ed è in aumento, mentre le esecuzioni simultanee si sono ridotte e la limitazione è stata interrotta. Ciò dimostra che il servizio Lambda ha interrotto la scalabilità a causa di errori. I log CloudWatch per la funzione forniscono dettagli sul tipo di errore.

È possibile risolvere questo problema identificando la funzione che causa l'errore, quindi trovando e risolvendo l'errore. Una volta corretto l'errore e implementato il nuovo codice, i parametri di CloudWatch mostrano come viene ripristinata l'elaborazione:



Qui, il numero di errori scende a zero e la percentuale di successo torna al 100%. Il servizio Lambda ricomincia a scalare la funzione, come mostrato nel grafico delle esecuzioni simultanee.

Identificazione e gestione della contropressione

Se un produttore di eventi genera costantemente messaggi per una coda SQS più velocemente di quanto una funzione Lambda possa elaborare la gestione, si verifica una contropressione. In questo caso, il monitoraggio SQS mostra l'età del primo messaggio che cresce in modo lineare, insieme al numero approssimativo di messaggi visibili. Puoi rilevare la contropressione nelle code utilizzando gli allarmi CloudWatch.

I passaggi per risolvere la contropressione dipendono dal carico di lavoro. Se l'obiettivo principale è aumentare la capacità di elaborazione e il throughput mediante la funzione Lambda:

- Richiedere un aumento della simultaneità al Supporto AWS nella regione specifica.
- Aumentare la dimensione del batch della funzione, in modo che a ogni invocazione della funzione vengano elaborati più messaggi.

Risoluzione dei problemi di rete in Lambda

Per impostazione predefinita, Lambda esegue le funzioni in un Virtual Private Cloud (VPC) interno con connettività ai servizi AWS e a Internet. Per accedere alle risorse di rete locali, è possibile configurare la funzione in modo che si connetta a un VPC nel proprio account. Quando si utilizza questa funzionalità, è possibile gestire l'accesso a Internet della funzione e la connettività di rete con risorse Amazon Virtual Private Cloud (Amazon VPC).

Gli errori di connettività di rete possono derivare da problemi relativi alla configurazione di routing del VPC, alle regole del gruppo di sicurezza, alle autorizzazioni dei ruoli AWS Identity and Access Management (IAM) o alla traduzione degli indirizzi di rete (NAT) del VPC o dalla disponibilità di risorse come indirizzi IP o interfacce di rete. A seconda del problema, potrebbe verificarsi un errore o un timeout specifico se una richiesta non riesce a raggiungere la sua destinazione.

Argomenti

- VPC: la funzione perde l'accesso a Internet o scade
- VPC: la connessione TCP o UDP fallisce a intermittenza
- VPC: è necessario accedere alla funzione Servizi AWS senza utilizzare Internet
- VPC: raggiunto il limite dell'interfaccia di rete elastica
- EC2: Interfaccia di rete elastica con tipo «lambda»
- DNS: impossibile connettersi agli host con UNKNOWNHOSTEXCEPTION

VPC: la funzione perde l'accesso a Internet o scade

Problema: la funzione Lambda perde l'accesso a Internet dopo la connessione a un VPC.

Errore: Errore: collegare ETIMEDUT 176.32.98.189:443

Errore: Errore: timeout dell'attività dopo 10,00 secondi

Errore: Tempo di lettura scaduto ReadTimeoutError. (timeout di lettura = 15)

Quando si collega una funzione a un VPC, tutte le richieste in uscita passano attraverso il VPC. Per connettersi a Internet, configurare il VPC per inviare il traffico in uscita dalla sottorete della funzione a un gateway NAT in una sottorete pubblica. Per ulteriori informazioni e le configurazioni VPC di esempio, vedere the section called “Accesso Internet per funzioni del VPC”.

Se alcune delle tue connessioni TCP stanno scadendo, verifica the section called “VPC: la connessione TCP o UDP fallisce a intermittenza” se la sottorete utilizza una lista di controllo degli accessi alla rete (NACL). Altrimenti, ciò è probabilmente dovuto alla frammentazione dei pacchetti. Le funzioni Lambda non sono in grado di gestire le richieste TCP frammentate in entrata, poiché Lambda non supporta la frammentazione IP per TCP o ICMP.

VPC: la connessione TCP o UDP fallisce a intermittenza

> **Note**
>
> Questo problema si applica solo se la sottorete utilizza una lista di controllo dell'accesso alla rete (ACL). La rete ACLs non è necessaria per consentire a Lambda di connettersi alle sottoreti.

Problema: Lambda perde in modo intermittente la connessione alle sottoreti VPC, per le quali hai configurato una lista di controllo dell'accesso alla rete (ACL).

Per le funzioni Lambda abilitate per VPC AWS , crea un ENIs iperpiano nell'account del cliente e utilizza `1024` porte temporanee `65535` per connettere Lambda al VPC del cliente. Se si utilizza la rete ACLs nella sottorete di destinazione, è necessario consentire l'intervallo di porte sia per TCP che per UDP. `1024` `65535` Non consentire questo intervallo completo di porte può causare errori di connessione intermittenti.

VPC: è necessario accedere alla funzione Servizi AWS senza utilizzare Internet

Problema: è necessario accedere alla funzione Lambda Servizi AWS senza utilizzare Internet.

Per connettere una funzione Servizi AWS da una sottorete privata senza accesso a Internet, usa gli endpoint VPC.

VPC: raggiunto il limite dell'interfaccia di rete elastica

Errore ENILimitReachedException: è stato raggiunto il limite dell'interfaccia elastica di rete per il VPC della funzione.

Quando si connette una funzione Lambda a un VPC, Lambda crea un'interfaccia di rete elastica per ogni combinazione di sottorete e gruppo di sicurezza collegato alla funzione. La quota di servizio predefinita è 250 interfacce di rete per VPC. Per richiedere un aumento di una quota, è possibile utilizzare la console Service Quotas.

EC2: Interfaccia di rete elastica con tipo «lambda»

Codice di errore: Client. OperationNotPermitted

Messaggio di errore: il gruppo di sicurezza non può essere modificato per questo tipo di interfaccia

Riceverai questo errore se cerchi di modificare un'interfaccia di rete elastica (ENI) gestita da Lambda. `ModifyNetworkInterfaceAttribute` non è incluso nell'API Lambda per le operazioni di aggiornamento su interfacce di rete elastiche create da Lambda.

DNS: impossibile connettersi agli host con UNKNOWNHOSTEXCEPTION

Messaggio di errore: UNKNOWNHOSTEXCEPTION

Le funzioni Lambda supportano un massimo di 20 connessioni TCP simultanee per la risoluzione DNS. La tua funzione potrebbe esaurire quel limite. Le richieste DNS più comuni vengono eseguite tramite UDP. Se la tua funzione consiste solo nel creare connessioni DNS UDP, è poco probabile che questo sia il tuo problema. Questo errore viene generalmente generato a causa di una configurazione errata o di un'infrastruttura degradata, quindi prima di esaminare a fondo il traffico DNS, verifica che l'infrastruttura DNS sia configurata correttamente e integra e che la funzione Lambda si riferisca a un host specificato nel DNS.

Se diagnostichi il problema come correlato al numero massimo di connessioni TCP, tieni presente che non puoi richiedere un aumento di questo limite. Se la tua funzione Lambda ritorna al DNS TCP a causa di payload DNS di grandi dimensioni, verifica che la tua soluzione utilizzi librerie che supportano EDNS. Per ulteriori informazioni su EDNS, consulta lo standard RFC 6891. Se i payload DNS superano costantemente le dimensioni massime EDNS, la soluzione potrebbe comunque esaurire il limite DNS TCP.

Applicazioni di esempio Lambda

Il repository GitHub per questa guida include applicazioni di esempio che illustrano l'uso di vari linguaggi e servizi AWS. Ogni applicazione di esempio include script per semplificare l'implementazione, la pulizia e il supporto delle risorse.

Node.js

Applicazioni Lambda di esempio in Node.js

- blank-nodejs – Una funzione Node.js che mostra l'uso del logging, delle variabili di ambiente, del tracciamento AWS X-Ray, dei livelli, dei test delle unità e dell'SDK AWS.
- nodejs-apig – Una funzione con un endpoint API pubblico che elabora un evento proveniente da API Gateway e restituisce una risposta HTTP.

Python

Applicazioni Lambda di esempio in Python

- blank-python – Una funzione Python che mostra l'uso di logging, variabili di ambiente, tracciamento AWS X-Ray, layer, unit test e SDK AWS.

Ruby

Applicazioni Lambda di esempio in Ruby

- blank-ruby – Una funzione Ruby che mostra l'uso di registrazione, variabili di ambiente, tracciamento AWS X-Ray, layer, unit test e SDK AWS.
- Esempi di Codice Ruby per Lambda AWS – Gli esempi di codice scritti in Ruby che dimostrano come interagire con AWS Lambda.

Java

Applicazioni Lambda di esempio in Java

- example-java: una funzione Java che dimostra come utilizzare Lambda per elaborare gli ordini. Questa funzione illustra come definire e deserializzare un oggetto evento di input personalizzato, utilizzare l'SDK AWS e registrare l'output.

- java-basic: una raccolta di funzioni Java minimali con unit test e configurazione della registrazione dei log delle variabili.
- java-events: una raccolta di funzioni Java che contengono codice skeleton per la gestione degli eventi di vari servizi, ad esempio Gateway Amazon API, Amazon SQS e Amazon Kinesis. Queste funzioni utilizzano la versione più recente della libreria aws-lambda-java-events (3.0.0 e versioni successive). Questi esempi non richiedono SDK AWS come dipendenza.
- s3-java – Una funzione Java che elabora gli eventi di notifica da Amazon S3 e utilizza la Java Class Library (JCL) per creare anteprime dai file di immagine caricati.
- layer-java: una funzione Java che illustra come utilizzare un livello Lambda per impacchettare dipendenze separate dal codice della funzione principale.

Esecuzione dei framework Java più diffusi su Lambda

- spring-cloud-function-samples: un esempio da Spring che mostra come utilizzare il framework Spring Cloud Function per creare funzioni AWS Lambda.
- Demo dell'applicazione Spring Boot serverless: un esempio che mostra come configurare una tipica applicazione Spring Boot in un runtime Java gestito, con e senza SnapStart, o come immagine nativa GraalVM con un runtime personalizzato.
- Demo dell'applicazione Micronaut serverless: un esempio che mostra come usare Micronaut in un runtime Java gestito, con e senza SnapStart, o come immagine nativa GraalVM con un runtime personalizzato. Scopri di più nelle guide Micronaut/Lambda.
- Demo dell'applicazione Quarkus serverless: un esempio che mostra come usare Quarkus in un runtime Java gestito, con e senza SnapStart, o come immagine nativa GraalVM con un runtime personalizzato. Scopri di più nella guida Quarkus/Lambda e nella guida Quarkus/SnapStart.

Go

Lambda fornisce le seguenti applicazioni di esempio per il runtime di Go:

Applicazioni Lambda di esempio in Go

- go-al2: una funzione hello world che restituisce l'indirizzo IP pubblico. Questa app utilizza il runtime personalizzato `provided.al2`.
- blank-go – Una funzione Go che mostra l'uso di librerie Go, registrazione, variabili d'ambiente e SDK AWS di Lambda. Questa app utilizza il runtime `go1.x`.

C#

Applicazioni Lambda di esempio in C#

- blank-csharp – Una funzione C# che mostra l'uso di librerie .NET di Lambda, logging, variabili di ambiente, tracciamento AWS X-Ray, unit test e SDK AWS.
- blank-csharp-with-layer: funzione C# che utilizza la CLI di .NET per creare un livello che raggruppa le dipendenze della funzione.
- ec2-spot – Una funzione che gestisce le richieste di istanze spot in Amazon EC2.

PowerShell

Lambda fornisce le seguenti applicazioni di esempio per PowerShell:

- blank-powershell – Una funzione PowerShell che mostra l'uso di registrazione, variabili di ambiente e SDK AWS.

Per distribuire un'applicazione di esempio, seguire le istruzioni contenute nel file README.

Usare Lambda con un SDK AWS

AWS i kit di sviluppo software (SDKs) sono disponibili per molti linguaggi di programmazione più diffusi. Ogni SDK fornisce un'API, esempi di codice e documentazione che facilitano agli sviluppatori la creazione di applicazioni nel loro linguaggio preferito.

Documentazione sugli SDK	Esempi di codice
AWS SDK per C++	AWS SDK per C++ esempi di codice
AWS CLI	AWS CLI esempi di codice
AWS SDK per Go	AWS SDK per Go esempi di codice
AWS SDK per Java	AWS SDK per Java esempi di codice
AWS SDK per JavaScript	AWS SDK per JavaScript esempi di codice
AWS SDK per Kotlin	AWS SDK per Kotlin esempi di codice
AWS SDK per .NET	AWS SDK per .NET esempi di codice
AWS SDK per PHP	AWS SDK per PHP esempi di codice
AWS Strumenti per PowerShell	AWS Strumenti per PowerShell esempi di codice
AWS SDK per Python (Boto3)	AWS SDK per Python (Boto3) esempi di codice
AWS SDK per Ruby	AWS SDK per Ruby esempi di codice
AWS SDK per Rust	AWS SDK per Rust esempi di codice
AWS SDK per SAP ABAP	AWS SDK per SAP ABAP esempi di codice
AWS SDK per Swift	AWS SDK per Swift esempi di codice

Per esempi specifici relativi a Lambda, consulta Esempi di codice per l'utilizzo di Lambda AWS SDKs.

Esempio di disponibilità

Non riesci a trovare quello che ti serve? Richiedi un esempio di codice utilizzando il link Fornisci un feedback nella parte inferiore di questa pagina.

Esempi di codice per l'utilizzo di Lambda AWS SDKs

I seguenti esempi di codice mostrano come usare Lambda con un kit di sviluppo AWS software (SDK).

Nozioni di base: esempi di codice che mostrano come eseguire le operazioni essenziali all'interno di un servizio.

Le azioni sono estratti di codice da programmi più grandi e devono essere eseguite nel contesto. Sebbene le azioni mostrino come richiamare le singole funzioni del servizio, è possibile visualizzarle contestualizzate negli scenari correlati.

Scenari: esempi di codice che mostrano come eseguire un'attività specifica chiamando più funzioni all'interno dello stesso servizio o combinate con altri Servizi AWS.

AWS i contributi della community sono esempi che sono stati creati e gestiti da più team. AWS Per fornire un feedback, utilizza il meccanismo disponibile nei repository collegati.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Esempi di codice

- Esempi di base per l'utilizzo di Lambda AWS SDKs
  - Hello Lambda
  - Scopri le basi di Lambda con un SDK AWS
  - Azioni per l'utilizzo di Lambda AWS SDKs
    - Utilizzare CreateAlias con una CLI
    - Utilizzo CreateFunction con un AWS SDK o una CLI
    - Utilizzare DeleteAlias con una CLI
    - Utilizzo DeleteFunction con un AWS SDK o una CLI
    - Utilizzare DeleteFunctionConcurrency con una CLI
    - Utilizzare DeleteProvisionedConcurrencyConfig con una CLI
    - Utilizzare GetAccountSettings con una CLI
    - Utilizzare GetAlias con una CLI
    - Utilizzo GetFunction con un AWS SDK o una CLI

- Utilizzare GetFunctionConcurrency con una CLI
- Utilizzare GetFunctionConfiguration con una CLI
- Utilizzare GetPolicy con una CLI
- Utilizzare GetProvisionedConcurrencyConfig con una CLI
- Utilizzo Invoke con un AWS SDK o una CLI
- Utilizzo ListFunctions con un AWS SDK o una CLI
- Utilizzare ListProvisionedConcurrencyConfigs con una CLI
- Utilizzare ListTags con una CLI
- Utilizzare ListVersionsByFunction con una CLI
- Utilizzare PublishVersion con una CLI
- Utilizzare PutFunctionConcurrency con una CLI
- Utilizzare PutProvisionedConcurrencyConfig con una CLI
- Utilizzare RemovePermission con una CLI
- Utilizzare TagResource con una CLI
- Utilizzare UntagResource con una CLI
- Utilizzare UpdateAlias con una CLI
- Utilizzo UpdateFunctionCode con un AWS SDK o una CLI
- Utilizzo UpdateFunctionConfiguration con un AWS SDK o una CLI

- Scenari per l'utilizzo di Lambda AWS SDKs
  - Conferma automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda utilizzando un SDK AWS
  - Esegui automaticamente la migrazione di utenti Amazon Cognito noti con una funzione Lambda utilizzando un SDK AWS
  - Creazione di un'API REST di API Gateway per monitorare i dati COVID-19
  - Creazione di una REST API per la libreria di prestiti
  - Creazione di un'applicazione di messaggistica con Step Functions
  - Creazione di un'applicazione di gestione delle risorse fotografiche che consente agli utenti di gestire le foto utilizzando etichette
  - Creazione di un'applicazione di chat websocket con API Gateway
  - Crea un'applicazione che analizza il feedback dei clienti e sintetizza l'audio
  - Richiamo a una funzione Lambda da un browser

- Trasformare i dati per l'applicazione con S3 Object Lambda
- Utilizzo di Gateway API per invocare una funzione Lambda
- Utilizzo di Step Functions per richiamare le funzioni Lambda
- Utilizzo degli eventi pianificati per invocare una funzione Lambda
- Utilizzare l'API Amazon Neptune per sviluppare una funzione Lambda che esegua query sui dati del grafico
- Scrivi dati di attività personalizzati con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito tramite un SDK AWS

- Esempi serverless per Lambda
  - Connessione a un database Amazon RDS in una funzione Lambda
  - Invocare una funzione Lambda da un trigger Kinesis
  - Richiamare una funzione Lambda da un trigger DynamoDB
  - Richiamare una funzione Lambda da un trigger Amazon DocumentDB
  - Invocare una funzione Lambda da un trigger Amazon MSK
  - Invocazione di una funzione Lambda da un trigger Amazon S3
  - Invocazione di una funzione Lambda da un trigger Amazon SNS
  - Richiamo di una funzione Lambda da un trigger Amazon SQS
  - Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis
  - Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB
  - Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS
- AWS contributi della community per Lambda
  - Creare e testare un'applicazione serverless

Esempi di base per l'utilizzo di Lambda AWS SDKs

I seguenti esempi di codice mostrano come utilizzare le nozioni di base di with. AWS Lambda AWS SDKs

Esempi

- Hello Lambda
- Scopri le basi di Lambda con un SDK AWS
- Azioni per l'utilizzo di Lambda AWS SDKs

- Utilizzare CreateAlias con una CLI
- Utilizzo CreateFunction con un AWS SDK o una CLI
- Utilizzare DeleteAlias con una CLI
- Utilizzo DeleteFunction con un AWS SDK o una CLI
- Utilizzare DeleteFunctionConcurrency con una CLI
- Utilizzare DeleteProvisionedConcurrencyConfig con una CLI
- Utilizzare GetAccountSettings con una CLI
- Utilizzare GetAlias con una CLI
- Utilizzo GetFunction con un AWS SDK o una CLI
- Utilizzare GetFunctionConcurrency con una CLI
- Utilizzare GetFunctionConfiguration con una CLI
- Utilizzare GetPolicy con una CLI
- Utilizzare GetProvisionedConcurrencyConfig con una CLI
- Utilizzo Invoke con un AWS SDK o una CLI
- Utilizzo ListFunctions con un AWS SDK o una CLI
- Utilizzare ListProvisionedConcurrencyConfigs con una CLI
- Utilizzare ListTags con una CLI
- Utilizzare ListVersionsByFunction con una CLI
- Utilizzare PublishVersion con una CLI
- Utilizzare PutFunctionConcurrency con una CLI
- Utilizzare PutProvisionedConcurrencyConfig con una CLI
- Utilizzare RemovePermission con una CLI
- Utilizzare TagResource con una CLI
- Utilizzare UntagResource con una CLI
- Utilizzare UpdateAlias con una CLI
- Utilizzo UpdateFunctionCode con un AWS SDK o una CLI
- Utilizzo UpdateFunctionConfiguration con un AWS SDK o una CLI

Hello Lambda

L'esempio di codice seguente mostra come iniziare a utilizzare Lambda.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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}");
        });
    }
}
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK per .NET API Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Codice per il CMake file CMake Lists.txt.

```
# 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})
```

Codice per il file sorgente 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;
}
```

- Per i dettagli sull'API, ListFunctionsconsulta AWS SDK per C++ API Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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)
  }
 }
}
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
        }
    }
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
};
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK per JavaScript API Reference.

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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()
```

- Per i dettagli sull'API, consulta ListFunctions AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK per Ruby API Reference.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Scopri le basi di Lambda con un SDK AWS

Gli esempi di codice seguenti mostrano come:

- Crea un ruolo IAM e una funzione Lambda, quindi carica il codice del gestore.
- Invocare la funzione con un singolo parametro e ottenere i risultati.
- Aggiorna il codice della funzione e configuralo con una variabile di ambiente.
- Invocare la funzione con nuovi parametri e ottenere i risultati. Visualizza il log di esecuzione restituito.
- Elenca le funzioni dell'account, quindi elimina le risorse.

Per ulteriori informazioni, consulta Creare una funzione Lambda con la console.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare metodi che eseguono operazioni 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;
    }


}
```

Creare una funzione che esegue lo scenario.

```
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();
    }
}
```

Definire un gestore Lambda che incrementa un numero.

```
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;
        }
    }
}
```

Definire un secondo gestore Lambda che esegue operazioni aritmetiche.

```
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;
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per .NET .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

C++

SDK per C++

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per C++ .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Go

SDK per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Creare uno scenario interattivo che ti mostri come iniziare a usare le funzioni 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
}
```

Creare una struttura che racchiude le singole operazioni Lambda.

```
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"`
}
```

Definire un gestore Lambda che incrementa un numero.

```
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
```

Definire un secondo gestore Lambda che esegue operazioni aritmetiche.

```
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
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke

- ListFunctions
- UpdateFunctionCode
- UpdateFunctionConfiguration

Java

SDK per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
        }
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK for Java 2.x .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Crea un ruolo AWS Identity and Access Management (IAM) che conceda a Lambda l'autorizzazione di scrittura nei 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);
};
```

Creare una funzione Lambda e caricare il codice del gestore.

```
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);
};
```

Invocare la funzione con un singolo parametro e ottenere i risultati.

```
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 };
};
```

Aggiornare il codice della funzione e configurare il suo ambiente Lambda con una variabile di ambiente.

```
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 = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  const result = client.send(command);
  waitForFunctionUpdated({ FunctionName: funcName });
  return result;
```

```
};
```

Elencare le funzioni per l'account.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

Eliminare il ruolo IAM e la funzione 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);
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per JavaScript .
  - CreateFunction
  - DeleteFunction

- GetFunction
- Invoke
- ListFunctions
- UpdateFunctionCode
- UpdateFunctionConfiguration

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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")
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Kotlin.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke

- ListFunctions
- UpdateFunctionCode
- UpdateFunctionConfiguration

PHP

SDK per PHP

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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";
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per PHP .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode

- UpdateFunctionConfiguration

Python

SDK per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Definire un gestore Lambda che incrementa un numero.

```
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
```

Definire un secondo gestore Lambda che esegue operazioni aritmetiche.

```
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
```

Creare funzioni che eseguono il wrap delle operazioni 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
```

Creare una funzione che esegue lo scenario.

```
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!")
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Python (Boto3).
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Ruby

SDK per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Configura le autorizzazioni IAM prerequisite per una funzione Lambda in grado di scrivere 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
```

Definisci un gestore Lambda che incrementa un numero fornito come parametro di invocazione.

```
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
```

Comprimi la funzione Lambda in un pacchetto di implementazione:

```
  # 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
```

Crea una nuova funzione Lambda.

```
  # 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
```

Invocare la tua funzione Lambda con parametri di runtime facoltativi.

```
  # 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
```

Aggiorna la configurazione della funzione Lambda per inserire una nuova variabile di ambiente.

```
 # 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
```

Aggiorna il codice della funzione Lambda con un pacchetto di implementazione diverso contenente codice diverso.

```
 # 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
```

Elenca tutte le funzioni Lambda esistenti utilizzando l'impaginatore integrato.

```
 # 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
```

Elimina una funzione Lambda specifica.

```
 # 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
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Ruby .
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Il file Cargo.toml con dipendenze utilizzato in questo scenario.

```
[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"
```

Una raccolta di utilità che semplificano le invocazioni a Lambda per questo scenario. Questo file è src/ations.rs nella cassa.

```
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(),
        );
    }
```

```
}
```

Un file binario per eseguire lo scenario dall'inizio alla fine, utilizzando i flag della linea di comando per controllare alcuni comportamenti. Questo file è in src/bin/scenario formato.rs nella cassa.

```
/*
## 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");
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API SDK AWS per Rust.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per informazioni dettagliate sull'API, consulta gli argomenti seguenti nella documentazione di riferimento dell'API AWS SDK per SAP ABAP.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke

- ListFunctions
- UpdateFunctionCode
- UpdateFunctionConfiguration

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Definisci la prima funzione Lambda, che incrementa semplicemente il valore specificato.

```
// 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()
```

Definisce la seconda funzione Lambda, che esegue un'operazione aritmetica su due numeri.

```
// 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()
```

Definisce il programma principale, che invocherà le due funzioni 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.IAMClientConfiguration(region: region)
        )

        let lambdaClient = try await LambdaClient(
            config: LambdaClient.LambdaClientConfiguration(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."
        }
    }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API SDK AWS per Swift.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Azioni per l'utilizzo di Lambda AWS SDKs

I seguenti esempi di codice mostrano come eseguire singole azioni Lambda con. AWS SDKs Ogni esempio include un collegamento a GitHub, dove è possibile trovare le istruzioni per la configurazione e l'esecuzione del codice.

Questi estratti chiamano l'API Lambda e sono estratti di codice da programmi più grandi che devono essere eseguiti in modo contestuale. È possibile visualizzare le azioni nel contesto in Scenari per l'utilizzo di Lambda AWS SDKs .

Gli esempi seguenti includono solo le azioni più comunemente utilizzate. Per un elenco completo, consulta la documentazione di riferimento dell'API AWS Lambda.

Esempi

- Utilizzare CreateAlias con una CLI
- Utilizzo CreateFunction con un AWS SDK o una CLI
- Utilizzare DeleteAlias con una CLI
- Utilizzo DeleteFunction con un AWS SDK o una CLI
- Utilizzare DeleteFunctionConcurrency con una CLI
- Utilizzare DeleteProvisionedConcurrencyConfig con una CLI
- Utilizzare GetAccountSettings con una CLI
- Utilizzare GetAlias con una CLI
- Utilizzo GetFunction con un AWS SDK o una CLI
- Utilizzare GetFunctionConcurrency con una CLI
- Utilizzare GetFunctionConfiguration con una CLI
- Utilizzare GetPolicy con una CLI
- Utilizzare GetProvisionedConcurrencyConfig con una CLI
- Utilizzo Invoke con un AWS SDK o una CLI
- Utilizzo ListFunctions con un AWS SDK o una CLI
- Utilizzare ListProvisionedConcurrencyConfigs con una CLI
- Utilizzare ListTags con una CLI
- Utilizzare ListVersionsByFunction con una CLI
- Utilizzare PublishVersion con una CLI

- Utilizzare PutFunctionConcurrency con una CLI
- Utilizzare PutProvisionedConcurrencyConfig con una CLI
- Utilizzare RemovePermission con una CLI
- Utilizzare TagResource con una CLI
- Utilizzare UntagResource con una CLI
- Utilizzare UpdateAlias con una CLI
- Utilizzo UpdateFunctionCode con un AWS SDK o una CLI
- Utilizzo UpdateFunctionConfiguration con un AWS SDK o una CLI

Utilizzare **`CreateAlias`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `CreateAlias`.

CLI

AWS CLI

Per creare un alias per una funzione Lambda

L'esempio `create-alias` seguente crea un alias denominato `LIVE` che punta alla versione 1 della funzione Lambda `my-function`.

```
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": "873282ed-4cd3-4dc8-a069-d0c647e470c6",
    "Description": "alias for live version of function"
}
```

Per ulteriori informazioni, consulta Configurazione degli alias delle funzioni AWS Lambda nella AWS Lambda Developer Guide.

- Per i dettagli sull'API, consulta CreateAliasCommand Reference.AWS CLI

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio crea un nuovo alias Lambda per la versione e la configurazione di routing specificate per specificare la percentuale di richieste di invocazione ricevute.

```
New-LMAlias -FunctionName "MylambdaFunction123" -
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6} -Description "Alias
 for version 4" -FunctionVersion 4 -Name "PowershellAlias"
```

- Per i dettagli sull'API, vedere CreateAliasin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio crea un nuovo alias Lambda per la versione e la configurazione di routing specificate per specificare la percentuale di richieste di invocazione ricevute.

```
New-LMAlias -FunctionName "MylambdaFunction123" -
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6} -Description "Alias
 for version 4" -FunctionVersion 4 -Name "PowershellAlias"
```

- Per i dettagli sull'API, vedere CreateAliasin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`CreateFunction`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `CreateFunction`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Informazioni di base

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
    }
```

- Per i dettagli sull'API, consulta la CreateFunctionsezione AWS SDK per .NET API Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
```

- Per i dettagli sull'API, consulta la CreateFunctionsezione AWS SDK per C++ API Reference.

CLI

AWS CLI

Come creare una funzione Lambda

L'esempio di `create-function` seguente crea una funzione Lambda denominata `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
```

Contenuto di `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": "873282ed-4cd3-4dc8-a069-d0c647e470c6",
    "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": ""
}
```

Per ulteriori informazioni, consulta Configurazione della memoria della funzione Lambda nella Guida per gli sviluppatori Lambda AWS .

- Per i dettagli sull'API, consulta CreateFunction AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

- Per i dettagli sull'API, consulta la CreateFunctionsezione AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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 "";
```

```
    }
```

- Per i dettagli sull'API, consulta la CreateFunctionsezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
};
```

- Per i dettagli sull'API, consulta la CreateFunctionsezione AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
    }
}
```

- Per i dettagli sull'API, CreateFunctionconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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",
            ]);
        });
    }
```

- Per i dettagli sull'API, consulta la CreateFunctionsezione AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: Questo esempio crea una nuova funzione C# (dotnetcore1.0 runtime) denominata in MyFunction AWS Lambda, che fornisce i file binari compilati per la funzione da un file

zip sul file system locale (è possibile utilizzare percorsi relativi o assoluti). Le funzioni C# Lambda specificano il gestore per la funzione utilizzando la designazione: :Namespace. AssemblyName ClassName::. MethodName È necessario sostituire in modo appropriato le parti relative al nome dell'assembly (senza il suffisso .dll), al namespace, al nome della classe e al nome del metodo delle specifiche dell'handler. La nuova funzione avrà le variabili di ambiente ’envvar1’ e ’envvar2’ impostate in base ai valori forniti.

```
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        :
```

Esempio 2: questo esempio è simile a quello precedente, tranne per il fatto che i binari delle funzioni vengono prima caricati in un bucket Amazon S3 (che deve trovarsi nella stessa regione della funzione Lambda prevista) e l’oggetto S3 risultante viene quindi referenziato durante la creazione della funzione.

```
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" }
```

- Per i dettagli sull'API, vedere CreateFunctionin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: Questo esempio crea una nuova funzione C# (dotnetcore1.0 runtime) denominata in MyFunction AWS Lambda, che fornisce i file binari compilati per la funzione da un file zip sul file system locale (è possibile utilizzare percorsi relativi o assoluti). Le funzioni C# Lambda specificano il gestore per la funzione utilizzando la designazione: :Namespace. AssemblyName ClassName::. MethodName È necessario sostituire in modo appropriato le parti relative al nome dell'assembly (senza il suffisso .dll), al namespace, al nome della classe e al nome del metodo delle specifiche dell'handler. La nuova funzione avrà le variabili di ambiente ’envvar1’ e ’envvar2’ impostate in base ai valori forniti.

```
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        :
```

Esempio 2: questo esempio è simile a quello precedente, tranne per il fatto che i binari delle funzioni vengono prima caricati in un bucket Amazon S3 (che deve trovarsi nella stessa regione della funzione Lambda prevista) e l'oggetto S3 risultante viene quindi referenziato durante la creazione della funzione.

```
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" }
```

- Per i dettagli sull'API, vedere CreateFunctionin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta CreateFunction AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta la CreateFunctionsezione AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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())
    }
```

- Per i dettagli sulle API, consulta il riferimento CreateFunctionall'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, CreateFunctionconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
    }
```

- Per i dettagli sull'API, consulta la CreateFunctionguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`DeleteAlias`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `DeleteAlias`.

CLI

AWS CLI

Per eliminare un alias di una funzione Lambda

L'esempio `delete-alias` seguente elimina l'alias denominato `LIVE` dalla funzione Lambda `my-function`.

```
aws lambda delete-alias \
```

```
    --function-name my-function \
    --name LIVE
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta Configurazione degli alias delle funzioni AWS Lambda nella AWS Lambda Developer Guide.

- Per i dettagli sull'API, consulta DeleteAliasCommand Reference.AWS CLI

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio elimina l'alias della funzione Lambda menzionato nel comando.

```
Remove-LMAlias -FunctionName "MylambdaFunction123" -Name "NewAlias"
```

- Per i dettagli sull'API, vedere DeleteAliasin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio elimina l'alias della funzione Lambda menzionato nel comando.

```
Remove-LMAlias -FunctionName "MylambdaFunction123" -Name "NewAlias"
```

- Per i dettagli sull'API, vedere DeleteAliasin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`DeleteFunction`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `DeleteFunction`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Informazioni di base

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
    }
```

- Per i dettagli sull'API, consulta la DeleteFunctionsezione AWS SDK per .NET API Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
    }
```

- Per i dettagli sull'API, consulta la DeleteFunctionsezione AWS SDK per C++ API Reference.

CLI

AWS CLI

Esempio 1: come eliminare una funzione Lambda in base al nome della funzione

L'esempio di `delete-function` seguente elimina la funzione Lambda denominata `my-function` specificandone il nome.

```
aws lambda delete-function \
    --function-name my-function
```

Questo comando non produce alcun output.

Esempio 2: eliminazione di una funzione Lambda in base all'ARN della funzione

L'esempio di `delete-function` seguente elimina la funzione Lambda denominata `my-function` specificando l'ARN della funzione.

```
aws lambda delete-function \
    --function-name arn:aws:lambda:us-west-2:123456789012:function:my-function
```

Questo comando non produce alcun output.

Esempio 3: eliminazione di una funzione Lambda in base all'ARN parziale della funzione

L'esempio di `delete-function` seguente elimina la funzione Lambda denominata `my-function` specificando l'ARN parziale della funzione.

```
aws lambda delete-function \
    --function-name 123456789012:function:my-function
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta Configurazione delle funzioni Lambda AWS nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta DeleteFunction AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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)
 }
}
```

- Per i dettagli sull'API, consulta la DeleteFunctionsezione AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
       }
   }
```

- Per i dettagli sull'API, consulta la DeleteFunctionsezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la DeleteFunctionsezione AWS SDK per JavaScript API Reference.

Kotlin

SDK per Kotlin

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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")
    }
}
```

- Per i dettagli sull'API, DeleteFunctionconsulta AWS SDK for Kotlin API reference.

PHP

SDK per PHP

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    public function deleteFunction($functionName)
    {
        return $this->lambdaClient->deleteFunction([
            'FunctionName' => $functionName,
        ]);
    }
```

- Per i dettagli sull'API, consulta la DeleteFunctionsezione AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio elimina una versione specifica di una funzione Lambda

```
Remove-LMFunction -FunctionName "MylambdaFunction123" -Qualifier '3'
```

- Per i dettagli sull'API, vedere DeleteFunctionin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio elimina una versione specifica di una funzione Lambda

```
Remove-LMFunction -FunctionName "MylambdaFunction123" -Qualifier '3'
```

- Per i dettagli sull'API, vedere DeleteFunctionin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta DeleteFunction AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta la DeleteFunctionsezione AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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)
    }
```

- Per i dettagli sulle API, consulta il riferimento DeleteFunctionall'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, DeleteFunctionconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.")
        }
```

- Per i dettagli sull'API, consulta la DeleteFunctionguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`DeleteFunctionConcurrency`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `DeleteFunctionConcurrency`.

CLI

AWS CLI

Per rimuovere un limite di esecuzione simultanea riservta da una funzione

L'esempio `delete-function-concurrency` seguente elimina il limite di esecuzione simultanea riservato dalla funzione `my-function`.

```
aws lambda delete-function-concurrency \
    --function-name  my-function
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta Prenotazione della simultaneità per una funzione Lambda nella Guida per gli sviluppatori di AWS Lambda.

- Per i dettagli sull'API, consulta DeleteFunctionConcurrency AWS CLICommand Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio rimuove la simultaneità della funzione dalla funzione Lambda.

```
Remove-LMFunctionConcurrency -FunctionName "MylambdaFunction123"
```

- Per i dettagli sull'API, vedere DeleteFunctionConcurrencyin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio rimuove la simultaneità della funzione dalla funzione Lambda.

```
Remove-LMFunctionConcurrency -FunctionName "MylambdaFunction123"
```

- Per i dettagli sull'API, vedere DeleteFunctionConcurrencyin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`DeleteProvisionedConcurrencyConfig`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `DeleteProvisionedConcurrencyConfig`.

CLI

AWS CLI

Per eliminare una configurazione della simultaneità fornita

L'esempio `delete-provisioned-concurrency-config` seguente elimina la configurazione di simultaneità fornita per l'alias `GREEN` della funzione specificata.

```
aws lambda delete-provisioned-concurrency-config \
    --function-name my-function \
    --qualifier GREEN
```

- Per i dettagli sull'API, consulta DeleteProvisionedConcurrencyConfig AWS CLICommand Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio rimuove la configurazione della simultaneità fornita per un alias specifico.

```
Remove-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
Qualifier "NewAlias1"
```

- Per i dettagli sull'API, vedere DeleteProvisionedConcurrencyConfigin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio rimuove la configurazione della simultaneità fornita per un alias specifico.

```
Remove-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
Qualifier "NewAlias1"
```

- Per i dettagli sull'API, vedere DeleteProvisionedConcurrencyConfigin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`GetAccountSettings`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `GetAccountSettings`.

CLI

AWS CLI

Per recuperare i dettagli sul tuo account in una regione AWS

L'esempio `get-account-settings` seguente mostra i limiti Lambda e le informazioni sull'utilizzo per il tuo account.

```
aws lambda get-account-settings
```

Output:

```
{
    "AccountLimit": {
       "CodeSizeUnzipped": 262144000,
       "UnreservedConcurrentExecutions": 1000,
       "ConcurrentExecutions": 1000,
       "CodeSizeZipped": 52428800,
       "TotalCodeSize": 80530636800
    },
    "AccountUsage": {
       "FunctionCount": 4,
       "TotalCodeSize": 9426
    }
}
```

Per ulteriori informazioni, consulta Limiti di AWS Lambda nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta GetAccountSettings AWS CLICommand Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio viene visualizzato per confrontare il limite dell'account e l'utilizzo dell'account

```
Get-LMAccountSetting | Select-Object
 @{Name="TotalCodeSizeLimit";Expression={$_.AccountLimit.TotalCodeSize}},
 @{Name="TotalCodeSizeUsed";Expression={$_.AccountUsage.TotalCodeSize}}
```

Output:

```
TotalCodeSizeLimit TotalCodeSizeUsed
------------------ -----------------
       80530636800          15078795
```

- Per i dettagli sull'API, vedere GetAccountSettingsin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio viene visualizzato per confrontare il limite dell'account e l'utilizzo dell'account

```
Get-LMAccountSetting | Select-Object
 @{Name="TotalCodeSizeLimit";Expression={$_.AccountLimit.TotalCodeSize}},
 @{Name="TotalCodeSizeUsed";Expression={$_.AccountUsage.TotalCodeSize}}
```

Output:

```
TotalCodeSizeLimit TotalCodeSizeUsed
------------------ -----------------
       80530636800          15078795
```

- Per i dettagli sull'API, vedere GetAccountSettingsin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`GetAlias`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `GetAlias`.

CLI

AWS CLI

Per recuperare i dettagli dell'alias di una funzione

L'esempio `get-alias` seguente visualizza i dettagli per l'alias denominato `LIVE` dalla funzione Lambda `my-function`.

```
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": "594f41fb-b85f-4c20-95c7-6ca5f2a92c93",
    "Description": "alias for live version of function"
}
```

Per ulteriori informazioni, consulta Configurazione degli alias delle funzioni AWS Lambda nella AWS Lambda Developer Guide.

- Per i dettagli sull'API, consulta GetAliasCommand Reference.AWS CLI

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio recupera i pesi di configurazione del routing per uno specifico alias della funzione Lambda.

```
Get-LMAlias -FunctionName "MylambdaFunction123" -Name "newlabel1" -Select
 RoutingConfig
```

Output:

```
AdditionalVersionWeights
------------------------
{[1, 0.6]}
```

- Per i dettagli sull'API, vedere GetAliasin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio recupera i pesi di configurazione del routing per uno specifico alias della funzione Lambda.

```
Get-LMAlias -FunctionName "MylambdaFunction123" -Name "newlabel1" -Select
 RoutingConfig
```

Output:

```
AdditionalVersionWeights
------------------------
{[1, 0.6]}
```

- Per i dettagli sull'API, vedere GetAliasin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`GetFunction`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `GetFunction`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Informazioni di base

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
    }
```

- Per i dettagli sull'API, consulta la GetFunctionsezione AWS SDK per .NET API Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
       }
```

- Per i dettagli sull'API, consulta la GetFunctionsezione AWS SDK per C++ API Reference.

CLI

AWS CLI

Come recuperare le informazioni relative a una funzione

Nell'esempio di `get-function` seguente vengono visualizzate informazioni sulla funzione `my-function`.

```
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": "28f0fb31-5c5c-43d3-8955-03e76c5c1075",
        "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": ""
    }
}
```

Per ulteriori informazioni, consulta Configurazione della memoria della funzione Lambda nella Guida per gli sviluppatori Lambda AWS .

- Per i dettagli sull'API, consulta GetFunction AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

- Per i dettagli sull'API, consulta la GetFunctionsezione AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
        }
    }
```

- Per i dettagli sull'API, consulta la GetFunctionsezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la GetFunctionsezione AWS SDK per JavaScript API Reference.

PHP

SDK per PHP

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    public function getFunction($functionName)
    {
        return $this->lambdaClient->getFunction([
            'FunctionName' => $functionName,
        ]);
    }
```

- Per i dettagli sull'API, consulta la GetFunctionsezione AWS SDK per PHP API Reference.

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta GetFunction AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta la GetFunctionsezione AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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)
    }
```

- Per i dettagli sulle API, consulta il riferimento GetFunctionall'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, GetFunctionconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
    }
```

- Per i dettagli sull'API, consulta la GetFunctionguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`GetFunctionConcurrency`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `GetFunctionConcurrency`.

CLI

AWS CLI

Per visualizzare l'impostazione di simultaneità riservata per una funzione

L'esempio `get-function-concurrency` seguente recupera l'impostazione della simultaneità riservata per la funzione specificata.

```
aws lambda get-function-concurrency \
    --function-name my-function
```

Output:

```
{
    "ReservedConcurrentExecutions": 250
}
```

- Per i dettagli sull'API, consulta GetFunctionConcurrency AWS CLICommand Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio ottiene la simultaneità riservata per la funzione Lambda

```
Get-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -Select *
```

Output:

```
ReservedConcurrentExecutions
----------------------------
100
```

- Per i dettagli sull'API, vedere GetFunctionConcurrencyin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio ottiene la simultaneità riservata per la funzione Lambda

```
Get-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -Select *
```

Output:

```
ReservedConcurrentExecutions
----------------------------
```

```
100
```

- Per i dettagli sull'API, vedere GetFunctionConcurrencyin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`GetFunctionConfiguration`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `GetFunctionConfiguration`.

CLI

AWS CLI

Per recuperare le impostazioni specifiche della versione di una funzione Lambda

L'esempio `get-function-configuration` seguente visualizza le impostazioni per la versione 2 della funzione `my-function`.

```
aws lambda get-function-configuration \
    --function-name  my-function:2
```

Output:

```
{
    "FunctionName": "my-function",
    "LastModified": "2019-09-26T20:28:40.438+0000",
    "RevisionId": "e52502d4-9320-4688-9cd6-152a6ab7490d",
    "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"
}
```

Per ulteriori informazioni, consulta Configurazione delle funzioni Lambda AWS nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta GetFunctionConfiguration AWS CLICommand Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio recupera la configurazione specifica della versione di una funzione 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
```

- Per i dettagli sull'API, vedere GetFunctionConfigurationin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio recupera la configurazione specifica della versione di una funzione 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
```

- Per i dettagli sull'API, vedere GetFunctionConfigurationin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`GetPolicy`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `GetPolicy`.

CLI

AWS CLI

Per recuperare la policy IAM basata sulle risorse per una funzione, una versione o un alias

L'esempio `get-policy` seguente mostra le informazioni sulla policy sulla funzione Lambda `my-function`.

```
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": "93017fc9-59cb-41dc-901b-4845ce4bf668"
}
```

Per ulteriori informazioni, consulta Using Resource-based Policies for Lambda nella AWS Lambda Developer Guide.AWS

- Per i dettagli sull'API, consulta Command Reference. GetPolicyAWS CLI

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio mostra la policy delle funzioni della funzione 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"}]}
```

- Per i dettagli sull'API, vedere GetPolicyin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio mostra la policy delle funzioni della funzione 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"}]}
```

- Per i dettagli sull'API, vedere GetPolicyin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`GetProvisionedConcurrencyConfig`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `GetProvisionedConcurrencyConfig`.

CLI

AWS CLI

Per visualizzare una configurazione della simultaneità fornita

L'esempio `get-provisioned-concurrency-config` seguente visualizza i dettagli della configurazione di simultaneità fornita per l'alias `BLUE` della funzione specificata.

```
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"
}
```

- Per i dettagli sull'API, consulta GetProvisionedConcurrencyConfig AWS CLICommand Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio ottiene la configurazione di simultaneità fornita per l'alias specificato della funzione 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                             :
```

- Per i dettagli sull'API, vedere GetProvisionedConcurrencyConfigin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio ottiene la configurazione di simultaneità fornita per l'alias specificato della funzione 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                             :
```

- Per i dettagli sull'API, vedere GetProvisionedConcurrencyConfigin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`Invoke`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `Invoke`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Informazioni di base

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
    }
```

- Per informazioni dettagliate sull'API, consulta Invoke nella documentazione di riferimento dell'API AWS SDK per .NET .

C++

SDK per C++

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
```

- Per informazioni dettagliate sull'API, consulta Invoke nella documentazione di riferimento dell'API AWS SDK per C++ .

CLI

AWS CLI

Esempio 1: come invocare una funzione Lambda in modo sincrono

L'esempio di `invoke` seguente invoca la funzione `my-function` in modo sincrono. L'`cli-binary-format`opzione è obbligatoria se utilizzi la versione 2 della AWS CLI. Per ulteriori informazioni, consulta Opzioni della riga di comando globali supportate da AWS CLI nella Guida per l'utente dell'Interfaccia della linea di comando AWS (AWS CLI).

```
aws lambda invoke \
    --function-name my-function \
    --cli-binary-format raw-in-base64-out \
```

```
    --payload '{ "name": "Bob" }' \
    response.json
```

Output:

```
{
    "ExecutedVersion": "$LATEST",
    "StatusCode": 200
}
```

Per ulteriori informazioni, consulta Invocare una funzione Lambda in modo sincrono nella Guida per gli sviluppatori di AWS Lambda.

Esempio 2: come invocare una funzione Lambda in modo asincrono

L'esempio di `invoke` seguente invoca la funzione `my-function` in modo asincrono. L'`cli-binary-format`opzione è obbligatoria se utilizzi la versione 2 della AWS CLI. Per ulteriori informazioni, consulta Opzioni della riga di comando globali supportate da AWS CLI nella Guida per l'utente dell'Interfaccia della linea di comando AWS (AWS CLI).

```
aws lambda invoke \
    --function-name my-function \
    --invocation-type Event \
    --cli-binary-format raw-in-base64-out \
    --payload '{ "name": "Bob" }' \
    response.json
```

Output:

```
{
    "StatusCode": 202
}
```

Per ulteriori informazioni, consulta Invocare una funzione Lambda in modo asincrono nella Guida per gli sviluppatori di AWS Lambda.

- Per informazioni dettagliate sull'API, consulta Invoke nella documentazione di riferimento dei comandi della AWS CLI .

Go

SDK per Go V2

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

- Per informazioni dettagliate sull'API, consulta Invoke nella documentazione di riferimento dell'API AWS SDK per Go .

Java

SDK per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
        }
    }
```

- Per informazioni dettagliate sull'API, consulta Invoke nella documentazione di riferimento dell'API AWS SDK for Java 2.x .

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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 };
};
```

- Per informazioni dettagliate sull'API, consulta Invoke nella documentazione di riferimento dell'API AWS SDK per JavaScript .

Kotlin

SDK per Kotlin

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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}")
    }
}
```

- Per informazioni dettagliate sull'API, consulta Invoke nella documentazione di riferimento dell'API AWS SDK per Kotlin.

PHP

SDK per PHP

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    public function invoke($functionName, $params, $logType = 'None')
    {
        return $this->lambdaClient->invoke([
            'FunctionName' => $functionName,
            'Payload' => json_encode($params),
            'LogType' => $logType,
        ]);
    }
```

- Per informazioni dettagliate sull'API, consulta Invoke nella documentazione di riferimento dell'API AWS SDK per PHP .

Python

SDK per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per informazioni dettagliate sull'API, consulta Invoke nella documentazione di riferimento dell'API AWS SDK per Python (Boto3).

Ruby

SDK per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per informazioni dettagliate sull'API, consulta Invoke nella documentazione di riferimento dell'API AWS SDK per Ruby .

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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")
    }
}
```

- Per informazioni dettagliate sull'API, consulta la pagina Invoke nella documentazione di riferimento dell'API SDK AWS per Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per informazioni dettagliate sull'API, consulta Invoke nella documentazione di riferimento dell'API AWS SDK per SAP ABAP.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
    }
```

- Per informazioni dettagliate sull'API, consulta Invoke nella documentazione di riferimento dell'API AWS SDK per Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`ListFunctions`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `ListFunctions`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Informazioni di base

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
}
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK per .NET API Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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());
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK per C++ API Reference.

CLI

AWS CLI

Come recuperare un elenco di funzioni Lambda

L'esempio di `list-functions` seguente visualizza un elenco di tutte le funzioni per l'utente attuale.

```
aws lambda list-functions
```

Output:

```
{
    "Functions": [
        {
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "$LATEST",
            "CodeSha256": "dBG9m8SGdmlEjw/JYXlhhvCrAv5TxvXsbL/RMr0fT/I=",
            "FunctionName": "helloworld",
            "MemorySize": 128,
            "RevisionId": "1718e831-badf-4253-9518-d0644210af7b",
            "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": "$LATEST",
            "CodeSha256": "sU0cJ2/hOZevwV/lTxCuQqK3gDZP3i8gUoqUUVRmY6E=",
            "FunctionName": "my-function",
            "VpcConfig": {
                "SubnetIds": [],
                "VpcId": "",
                "SecurityGroupIds": []
            },
            "MemorySize": 256,
            "RevisionId": "93017fc9-59cb-41dc-901b-4845ce4bf668",
            "CodeSize": 266,
            "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-
function",
            "Handler": "index.handler",
            "Role": "arn:aws:iam::123456789012:role/service-role/
helloWorldPython-role-uy3l9qyq",
            "Timeout": 3,
            "LastModified": "2025-10-01T16:47:28.490+0000",
            "Runtime": "nodejs22.x",
            "Description": ""
        },
        {
            "Layers": [
                {
                    "CodeSize": 41784542,
                    "Arn": "arn:aws:lambda:us-
west-2:420165488524:layer:AWSLambda-Python37-SciPy1x:2"
                },
                {
                    "CodeSize": 4121,
                    "Arn": "arn:aws:lambda:us-
west-2:123456789012:layer:pythonLayer:1"
                }
            ],
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "Version": "$LATEST",
            "CodeSha256": "ZQukCqxtkqFgyF2cU41Avj99TKQ/hNihPtDtRcc08mI=",
            "FunctionName": "my-python-function",
            "VpcConfig": {
```

```
                "SubnetIds": [],
                "VpcId": "",
                "SecurityGroupIds": []
            },
            "MemorySize": 128,
            "RevisionId": "80b4eabc-acf7-4ea8-919a-e874c213707d",
            "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": ""
        }
    ]
}
```

Per ulteriori informazioni, consulta Configurazione della memoria della funzione Lambda nella Guida per gli sviluppatori Lambda AWS .

- Per i dettagli sull'API, consulta ListFunctions AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK per Go API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK per JavaScript API Reference.

PHP

SDK per PHP

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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,
        ]);
    }
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio mostra tutte le funzioni Lambda con dimensioni di codice ordinate

```
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
```

- Per i dettagli sull'API, vedere ListFunctionsin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio mostra tutte le funzioni Lambda con dimensioni di codice ordinate

```
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
```

- Per i dettagli sull'API, vedere ListFunctionsin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta ListFunctions AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta la ListFunctionssezione AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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)
   }
```

- Per i dettagli sulle API, consulta il riferimento ListFunctionsall'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, ListFunctionsconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
    }
```

- Per i dettagli sull'API, consulta la ListFunctionsguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`ListProvisionedConcurrencyConfigs`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `ListProvisionedConcurrencyConfigs`.

CLI

AWS CLI

Per ottenere un elenco di configurazioni di simultaneità fornita

L'esempio `list-provisioned-concurrency-configs` seguente elenca le configurazioni di simultaneità fornita per la funzione specificata.

```
aws lambda list-provisioned-concurrency-configs \
    --function-name my-function
```

Output:

```
{
    "ProvisionedConcurrencyConfigs": [
        {
            "FunctionArn": "arn:aws:lambda:us-east-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-east-2:123456789012:function:my-
function:BLUE",
            "RequestedProvisionedConcurrentExecutions": 100,
            "AvailableProvisionedConcurrentExecutions": 100,
            "AllocatedProvisionedConcurrentExecutions": 100,
            "Status": "READY",
            "LastModified": "2019-12-31T20:28:49+0000"
        }
    ]
}
```

- Per i dettagli sull'API, consulta ListProvisionedConcurrencyConfigs AWS CLICommand Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio recupera l'elenco di configurazioni di simultaneità fornita per una funzione Lambda.

```
Get-LMProvisionedConcurrencyConfigList -FunctionName "MylambdaFunction123"
```

- Per i dettagli sull'API, vedere ListProvisionedConcurrencyConfigsin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio recupera l'elenco di configurazioni di simultaneità fornita per una funzione Lambda.

```
Get-LMProvisionedConcurrencyConfigList -FunctionName "MylambdaFunction123"
```

- Per i dettagli sull'API, vedere ListProvisionedConcurrencyConfigsin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`ListTags`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `ListTags`.

CLI

AWS CLI

Per recuperare l'elenco di tag per una funzione Lambda

L'esempio `list-tags` seguente visualizza i tag collegati alla funzione Lambda `my-function`.

```
aws lambda list-tags \
    --resource arn:aws:lambda:us-west-2:123456789012:function:my-function
```

Output:

```
{
    "Tags": {
        "Category": "Web Tools",
        "Department": "Sales"
    }
}
```

Per ulteriori informazioni, consulta Tagging delle funzioni Lambda nella Guida per gli sviluppatori di AWS Lambda.

- Per i dettagli sull'API, consulta ListTags AWS CLICommand Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: recupera i tag e i relativi valori attualmente impostati sulla funzione specificata.

```
Get-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction"
```

Output:

```
Key        Value
---        -----
California Sacramento
Oregon     Salem
Washington Olympia
```

- Per i dettagli sull'API, vedere ListTagsin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: recupera i tag e i relativi valori attualmente impostati sulla funzione specificata.

```
Get-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction"
```

Output:

```
Key        Value
---        -----
California Sacramento
Oregon     Salem
Washington Olympia
```

- Per i dettagli sull'API, vedere ListTagsin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`ListVersionsByFunction`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `ListVersionsByFunction`.

CLI

AWS CLI

Per recuperare un elenco di versioni di una funzione

L'esempio `list-versions-by-function` seguente visualizza l'elenco di versioni per la funzione Lambda `my-function`.

```
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": "93017fc9-59cb-41dc-901b-4845ce4bf668",
            "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": "949c8914-012e-4795-998c-e467121951b1",
            "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": "cd669f21-0f3d-4e1c-9566-948837f2e2ea",
            "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"
        }
    ]
}
```

Per ulteriori informazioni, consulta Configurazione degli alias delle funzioni AWS Lambda nella AWS Lambda Developer Guide.

- Per i dettagli sull'API, consulta ListVersionsByFunctionCommand Reference.AWS CLI

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio restituisce l’elenco di configurazioni specifiche della versione per ogni versione della funzione 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
```

- Per i dettagli sull'API, vedere ListVersionsByFunctionin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio restituisce l'elenco di configurazioni specifiche della versione per ogni versione della funzione 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
```

- Per i dettagli sull'API, vedere ListVersionsByFunctionin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`PublishVersion`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `PublishVersion`.

CLI

AWS CLI

Per pubblicare una nuova versione di una funzione

L'esempio `publish-version` seguente illustra la pubblicazione di una nuova versione della funzione Lambda `my-function`.

```
aws lambda publish-version \
    --function-name my-function
```

Output:

```
{
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "CodeSha256": "dBG9m8SGdmlEjw/JYXlhhvCrAv5TxvXsbL/RMr0fT/I=",
    "FunctionName": "my-function",
    "CodeSize": 294,
    "RevisionId": "f31d3d39-cc63-4520-97d4-43cd44c94c20",
    "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": ""
}
```

Per ulteriori informazioni, consulta Configurazione degli alias delle funzioni AWS Lambda nella AWS Lambda Developer Guide.

- Per i dettagli sull'API, consulta PublishVersionCommand Reference.AWS CLI

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio crea una versione per lo snapshot esistente del codice funzione Lambda

```
Publish-LMVersion -FunctionName "MylambdaFunction123" -Description "Publishing
 Existing Snapshot of function code as a  new version through Powershell"
```

- Per i dettagli sull'API, vedere PublishVersionin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio crea una versione per lo snapshot esistente del codice funzione Lambda

```
Publish-LMVersion -FunctionName "MylambdaFunction123" -Description "Publishing
 Existing Snapshot of function code as a  new version through Powershell"
```

- Per i dettagli sull'API, vedere PublishVersionin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`PutFunctionConcurrency`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `PutFunctionConcurrency`.

CLI

AWS CLI

Per configurare un limite di simultaneità riservata per una funzione

L'esempio `put-function-concurrency` seguente configura 100 esecuzioni simultanee riservate per la funzione `my-function`.

```
aws lambda put-function-concurrency \
    --function-name  my-function  \
    --reserved-concurrent-executions 100
```

Output:

```
{
    "ReservedConcurrentExecutions": 100
}
```

Per ulteriori informazioni, consulta Reserving Concurrency for a Lambda Function nella Guida per gli sviluppatori di AWS Lambda.

- Per i dettagli sull'API, consulta PutFunctionConcurrency AWS CLICommand Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio applica le impostazioni di simultaneità per l'intera funzione.

```
Write-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -
ReservedConcurrentExecution 100
```

- Per i dettagli sull'API, vedere PutFunctionConcurrencyin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio applica le impostazioni di simultaneità per l'intera funzione.

```
Write-LMFunctionConcurrency -FunctionName "MylambdaFunction123" -
ReservedConcurrentExecution 100
```

- Per i dettagli sull'API, vedere PutFunctionConcurrencyin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`PutProvisionedConcurrencyConfig`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `PutProvisionedConcurrencyConfig`.

CLI

AWS CLI

Per allocare la simultaneità fornita

L'esempio `put-provisioned-concurrency-config` seguente alloca 100 simultaneità fornita per l'alias BLUE della funzione specificata.

```
aws lambda put-provisioned-concurrency-config \
    --function-name my-function \
    --qualifier BLUE \
    --provisioned-concurrent-executions 100
```

Output:

```
{
    "Requested ProvisionedConcurrentExecutions": 100,
    "Allocated ProvisionedConcurrentExecutions": 0,
    "Status": "IN_PROGRESS",
    "LastModified": "2019-11-21T19:32:12+0000"
}
```

- Per i dettagli sull'API, consulta PutProvisionedConcurrencyConfig AWS CLICommand Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio aggiunge una configurazione di simultaneità fornita all'alias di una funzione

```
Write-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
ProvisionedConcurrentExecution 20 -Qualifier "NewAlias1"
```

- Per i dettagli sull'API, vedere PutProvisionedConcurrencyConfigin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio aggiunge una configurazione di simultaneità fornita all'alias di una funzione

```
Write-LMProvisionedConcurrencyConfig -FunctionName "MylambdaFunction123" -
ProvisionedConcurrentExecution 20 -Qualifier "NewAlias1"
```

- Per i dettagli sull'API, vedere PutProvisionedConcurrencyConfigin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`RemovePermission`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `RemovePermission`.

CLI

AWS CLI

Per rimuovere le autorizzazioni da una funzione Lambda esistente

L'esempio `remove-permission` seguente concede rimuove l'autorizzazione per invocare una funzione denominata `my-function`.

```
aws lambda remove-permission \
    --function-name my-function \
    --statement-id sns
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta Using Resource-based Policies for Lambda nella AWS Lambda Developer Guide.AWS

- Per i dettagli sull'API, consulta Command Reference. RemovePermissionAWS CLI

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio rimuove la politica StatementId della funzione per la funzione Lambda specificata.

```
$policy =  Get-LMPolicy -FunctionName "MylambdaFunction123" -Select Policy |
 ConvertFrom-Json| Select-Object -ExpandProperty Statement
Remove-LMPermission -FunctionName "MylambdaFunction123" -StatementId
 $policy[0].Sid
```

- Per i dettagli sull'API, vedere RemovePermissionin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio rimuove la politica StatementId della funzione per la funzione Lambda specificata.

```
$policy =  Get-LMPolicy -FunctionName "MylambdaFunction123" -Select Policy |
 ConvertFrom-Json| Select-Object -ExpandProperty Statement
Remove-LMPermission -FunctionName "MylambdaFunction123" -StatementId
 $policy[0].Sid
```

- Per i dettagli sull'API, vedere RemovePermissionin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`TagResource`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `TagResource`.

CLI

AWS CLI

Per aggiungere tag a una funzione Lambda esistente

L'esempio `tag-resource` seguente aggiunge un tag con il nome della chiave `DEPARTMENT` e un valore di `Department A` alla funzione Lambda specificata.

```
aws lambda tag-resource \
    --resource arn:aws:lambda:us-west-2:123456789012:function:my-function \
    --tags "DEPARTMENT=Department A"
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta Tagging Lambda Functions nella Guida per gli sviluppatori di AWS Lambda.

- Per i dettagli sull'API, consulta TagResource AWS CLICommand Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: aggiunge i tre tag (Washington, Oregon e California) e i relativi valori associati alla funzione specificata identificata dal relativo ARN.

```
Add-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction" -Tag @{ "Washington" = "Olympia";
 "Oregon" = "Salem"; "California" = "Sacramento" }
```

- Per i dettagli sull'API, vedere TagResourcein AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: aggiunge i tre tag (Washington, Oregon e California) e i relativi valori associati alla funzione specificata identificata dal relativo ARN.

```
Add-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction" -Tag @{ "Washington" = "Olympia";
 "Oregon" = "Salem"; "California" = "Sacramento" }
```

- Per i dettagli sull'API, vedere TagResourcein AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`UntagResource`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `UntagResource`.

CLI

AWS CLI

Per rimuovere i tag da una funzione Lambda esistente

L'esempio `untag-resource` seguente rimuove il tag con il tag `DEPARTMENT` per il nome della chiave dalla funzione Lambda `my-function`.

```
aws lambda untag-resource \
    --resource arn:aws:lambda:us-west-2:123456789012:function:my-function \
    --tag-keys DEPARTMENT
```

Questo comando non produce alcun output.

Per ulteriori informazioni, consulta Tagging Lambda Functions nella Guida per gli sviluppatori di AWS Lambda.

- Per i dettagli sull'API, consulta UntagResource AWS CLICommand Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: rimuove i tag forniti da una funzione. Il cmdlet richiederà una conferma prima di procedere, a meno che non venga specificato lo switch -Force. Viene effettuata una sola chiamata al servizio per rimuovere i tag.

```
Remove-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction" -TagKey
 "Washington","Oregon","California"
```

Esempio 2: rimuove i tag forniti da una funzione. Il cmdlet richiederà una conferma prima di procedere, a meno che non venga specificato lo switch -Force. Una volta effettuata la chiamata al servizio per tag fornito.

```
"Washington","Oregon","California" | Remove-LMResourceTag -Resource
 "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
```

- Per i dettagli sull'API, vedere UntagResourcein AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: rimuove i tag forniti da una funzione. Il cmdlet richiederà una conferma prima di procedere, a meno che non venga specificato lo switch -Force. Viene effettuata una sola chiamata al servizio per rimuovere i tag.

```
Remove-LMResourceTag -Resource "arn:aws:lambda:us-
west-2:123456789012:function:MyFunction" -TagKey
 "Washington","Oregon","California"
```

Esempio 2: rimuove i tag forniti da una funzione. Il cmdlet richiederà una conferma prima di procedere, a meno che non venga specificato lo switch -Force. Una volta effettuata la chiamata al servizio per tag fornito.

```
"Washington","Oregon","California" | Remove-LMResourceTag -Resource
 "arn:aws:lambda:us-west-2:123456789012:function:MyFunction"
```

- Per i dettagli sull'API, vedere UntagResourcein AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare **`UpdateAlias`** con una CLI

Gli esempi di codice seguenti mostrano come utilizzare `UpdateAlias`.

CLI

AWS CLI

Per aggiornare l'alias di una funzione

L'esempio `update-alias` seguente aggiorna l'alias denominato `LIVE` che punti alla versione 3 della funzione Lambda `my-function`.

```
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": "594f41fb-b85f-4c20-95c7-6ca5f2a92c93",
    "Description": "alias for live version of function"
}
```

Per ulteriori informazioni, consulta Configurazione degli alias delle funzioni AWS Lambda nella AWS Lambda Developer Guide.

- Per i dettagli sull'API, consulta UpdateAliasCommand Reference.AWS CLI

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio aggiorna la configurazione di un alias di funzione Lambda esistente. Aggiorna il RoutingConfiguration valore per spostare il 60% (0,6) del traffico alla versione 1

```
Update-LMAlias -FunctionName "MylambdaFunction123" -Description
 " Alias for version 2" -FunctionVersion 2 -Name "newlabel1" -
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6}
```

- Per i dettagli sull'API, vedere UpdateAliasin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio aggiorna la configurazione di un alias di funzione Lambda esistente. Aggiorna il RoutingConfiguration valore per spostare il 60% (0,6) del traffico alla versione 1

```
Update-LMAlias -FunctionName "MylambdaFunction123" -Description
 " Alias for version 2" -FunctionVersion 2 -Name "newlabel1" -
RoutingConfig_AdditionalVersionWeight @{Name="1";Value="0.6}
```

- Per i dettagli sull'API, vedere UpdateAliasin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, vedere. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`UpdateFunctionCode`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `UpdateFunctionCode`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Informazioni di base

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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}.");
    }
```

- Per i dettagli sull'API, consulta la UpdateFunctionCodesezione AWS SDK per .NET API Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
```

- Per i dettagli sull'API, consulta la UpdateFunctionCodesezione AWS SDK per C++ API Reference.

CLI

AWS CLI

Come aggiornare il codice di una funzione Lambda

L'esempio di `update-function-code` seguente sostituisce il codice della versione ($LATEST) non pubblicata della funzione `my-function` con i contenuti del file zip specificato.

```
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": "e52502d4-9320-4688-9cd6-152a6ab7490d",
    "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"
}
```

Per ulteriori informazioni, consulta Configurazione delle funzioni Lambda AWS nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta UpdateFunctionCode AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
}
```

- Per i dettagli sull'API, consulta la UpdateFunctionCodesezione AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
        }
    }
```

- Per i dettagli sull'API, consulta la UpdateFunctionCodesezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
};
```

- Per i dettagli sull'API, consulta la UpdateFunctionCodesezione AWS SDK per JavaScript API Reference.

PHP

SDK per PHP

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    public function updateFunctionCode($functionName, $s3Bucket, $s3Key)
    {
        return $this->lambdaClient->updateFunctionCode([
            'FunctionName' => $functionName,
            'S3Bucket' => $s3Bucket,
            'S3Key' => $s3Key,
        ]);
    }
```

- Per i dettagli sull'API, consulta la UpdateFunctionCodesezione AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: aggiorna la funzione denominata MyFunction '' con il nuovo contenuto contenuto nel file zip specificato. Per una funzione Lambda C# .NET Core, il file zip deve contenere l'assembly compilato.

```
Update-LMFunctionCode -FunctionName MyFunction -ZipFilename .\UpdatedCode.zip
```

Esempio 2: questo esempio è simile a quello precedente ma utilizza un oggetto Amazon S3 contenente il codice aggiornato per aggiornare la funzione.

```
Update-LMFunctionCode -FunctionName MyFunction -BucketName amzn-s3-demo-bucket -
Key UpdatedCode.zip
```

- Per i dettagli sull'API, vedere UpdateFunctionCodein AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: aggiorna la funzione denominata MyFunction '' con il nuovo contenuto contenuto nel file zip specificato. Per una funzione Lambda C# .NET Core, il file zip deve contenere l'assembly compilato.

```
Update-LMFunctionCode -FunctionName MyFunction -ZipFilename .\UpdatedCode.zip
```

Esempio 2: questo esempio è simile a quello precedente ma utilizza un oggetto Amazon S3 contenente il codice aggiornato per aggiornare la funzione.

```
Update-LMFunctionCode -FunctionName MyFunction -BucketName amzn-s3-demo-bucket -
Key UpdatedCode.zip
```

- Per i dettagli sull'API, vedere UpdateFunctionCodein AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta UpdateFunctionCode AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta la UpdateFunctionCodesezione AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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())
    }
```

- Per i dettagli sulle API, consulta il riferimento UpdateFunctionCodeall'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, UpdateFunctionCodeconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
```

- Per i dettagli sull'API, consulta la UpdateFunctionCodeguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo **`UpdateFunctionConfiguration`** con un AWS SDK o una CLI

Gli esempi di codice seguenti mostrano come utilizzare `UpdateFunctionConfiguration`.

Gli esempi di operazioni sono estratti di codice da programmi più grandi e devono essere eseguiti nel contesto. È possibile visualizzare questa operazione nel contesto nel seguente esempio di codice:

- Informazioni di base

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
    }
```

- Per i dettagli sull'API, consulta la UpdateFunctionConfigurationsezione AWS SDK per .NET API Reference.

C++

SDK per C++

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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;
        }
```

- Per i dettagli sull'API, consulta la UpdateFunctionConfigurationsezione AWS SDK per C++ API Reference.

CLI

AWS CLI

Come modificare la configurazione di una funzione

L'esempio di `update-function-configuration` seguente modifica la dimensione della memoria in 256 MB per la versione non pubblicata ($LATEST) della funzione `my-function`.

```
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": "e52502d4-9320-4688-9cd6-152a6ab7490d",
    "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"
}
```

Per ulteriori informazioni, consulta Configurazione delle funzioni Lambda AWS nella Guida per gli sviluppatori di AWS .

- Per i dettagli sull'API, consulta UpdateFunctionConfiguration AWS CLICommand Reference.

Go

SDK per Go V2

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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)
 }
}
```

- Per i dettagli sull'API, consulta la UpdateFunctionConfigurationsezione AWS SDK per Go API Reference.

Java

SDK per Java 2.x

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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);
        }
    }
```

- Per i dettagli sull'API, consulta la UpdateFunctionConfigurationsezione AWS SDK for Java 2.x API Reference.

JavaScript

SDK per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  const result = client.send(command);
  waitForFunctionUpdated({ FunctionName: funcName });
  return result;
};
```

- Per i dettagli sull'API, consulta la UpdateFunctionConfigurationsezione AWS SDK per JavaScript API Reference.

PHP

SDK per PHP

> **Note**
>
> C'è altro su GitHub. Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

```
    public function updateFunctionConfiguration($functionName, $handler,
 $environment = '')
    {
```

```
        return $this->lambdaClient->updateFunctionConfiguration([
            'FunctionName' => $functionName,
            'Handler' => "$handler.lambda_handler",
            'Environment' => $environment,
        ]);
    }
```

- Per i dettagli sull'API, consulta la UpdateFunctionConfigurationsezione AWS SDK per PHP API Reference.

PowerShell

Strumenti per PowerShell V4

Esempio 1: questo esempio aggiorna la configurazione della funzione Lambda esistente

```
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
```

- Per i dettagli sull'API, vedere UpdateFunctionConfigurationin AWS Strumenti per PowerShell Cmdlet Reference (V4).

Strumenti per V5 PowerShell

Esempio 1: questo esempio aggiorna la configurazione della funzione Lambda esistente

```
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
```

- Per i dettagli sull'API, vedere UpdateFunctionConfigurationin AWS Strumenti per PowerShell Cmdlet Reference (V5).

Python

SDK per Python (Boto3)

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta UpdateFunctionConfiguration AWSSDK for Python (Boto3) API Reference.

Ruby

SDK per Ruby

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
```

- Per i dettagli sull'API, consulta la UpdateFunctionConfigurationsezione AWS SDK per Ruby API Reference.

Rust

SDK per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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)
    }
```

- Per i dettagli sulle API, consulta il riferimento UpdateFunctionConfigurationall'API AWS SDK for Rust.

SAP ABAP

SDK per SAP ABAP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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.
```

- Per i dettagli sulle API, UpdateFunctionConfigurationconsulta AWS SDK for SAP ABAP API reference.

Swift

SDK per Swift

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice 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
        }
    }
```

- Per i dettagli sull'API, consulta la UpdateFunctionConfigurationguida di riferimento all'API AWS SDK for Swift.

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Scenari per l'utilizzo di Lambda AWS SDKs

I seguenti esempi di codice mostrano come implementare scenari comuni in Lambda con. AWS SDKs Questi scenari illustrano come eseguire attività specifiche richiamando più funzioni all'interno di Lambda o combinate con altri Servizi AWS. Ogni scenario include un collegamento al codice sorgente completo, dove è possibile trovare le istruzioni su come configurare ed eseguire il codice.

Gli scenari sono relativi a un livello intermedio di esperienza per aiutarti a comprendere le azioni di servizio nel contesto.

Esempi

- Conferma automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda utilizzando un SDK AWS
- Esegui automaticamente la migrazione di utenti Amazon Cognito noti con una funzione Lambda utilizzando un SDK AWS
- Creazione di un'API REST di API Gateway per monitorare i dati COVID-19
- Creazione di una REST API per la libreria di prestiti
- Creazione di un'applicazione di messaggistica con Step Functions
- Creazione di un'applicazione di gestione delle risorse fotografiche che consente agli utenti di gestire le foto utilizzando etichette
- Creazione di un'applicazione di chat websocket con API Gateway
- Crea un'applicazione che analizza il feedback dei clienti e sintetizza l'audio
- Richiamo a una funzione Lambda da un browser
- Trasformare i dati per l'applicazione con S3 Object Lambda
- Utilizzo di Gateway API per invocare una funzione Lambda
- Utilizzo di Step Functions per richiamare le funzioni Lambda
- Utilizzo degli eventi pianificati per invocare una funzione Lambda
- Utilizzare l'API Amazon Neptune per sviluppare una funzione Lambda che esegua query sui dati del grafico

- Scrivi dati di attività personalizzati con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito tramite un SDK AWS

Conferma automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda utilizzando un SDK AWS

L'esempio di codice seguente mostra come confermare automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda.

- Configura un pool di utenti per chiamare una funzione Lambda per il trigger `PreSignUp`.
- Registra un utente con Amazon Cognito.
- La funzione Lambda esegue la scansione di una tabella DynamoDB e conferma automaticamente gli utenti noti.
- Accedi come nuovo utente, quindi elimina le risorse.

Go

SDK per Go V2

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
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))
}
```

Gestisci il trigger `PreSignUp` con una funzione 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)
}
```

Crea una struttura che esegue operazioni comuni.

```
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))
}
```

Crea una struttura che racchiude le azioni di 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
}
```

Crea una struttura che racchiude le azioni di 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
}
```

Crea una struttura che racchiuda le azioni di CloudWatch Logs.

```
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
}
```

Crea una struttura che racchiuda le azioni. CloudFormation

```
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
}
```

Eliminare le risorse.

```
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!")
 }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

JavaScript

SDK per (v3 JavaScript)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Configura un'esecuzione interattiva dello "Scenario". Gli esempi JavaScript (v3) condividono uno Scenario runner per semplificare esempi complessi. Il codice sorgente completo è attivo. 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.",
  });
}
```

Questo scenario dimostra la conferma automatica di un utente noto. Orchestra i passaggi di esempio.

```
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,
  );
```

Questi sono passaggi condivisi con altri scenari.

```
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 },
);
```

Un handler per il trigger `PreSignUp` con una funzione 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);
};
```

Modulo di azioni CloudWatch Logs.

```
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];
  }
};
```

Modulo di azioni 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];
  }
};
```

Modulo di azioni 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];
  }
};
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per JavaScript .
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Esegui automaticamente la migrazione di utenti Amazon Cognito noti con una funzione Lambda utilizzando un SDK AWS

L'esempio di codice seguente mostra come migrare automaticamente gli utenti noti di Amazon Cognito con una funzione Lambda.

- Configura un pool di utenti per chiamare una funzione Lambda per il trigger `MigrateUser`.

- Accedi ad Amazon Cognito con un nome utente e un indirizzo e-mail non incluso nel pool di utenti.
- La funzione Lambda esegue la scansione di una tabella DynamoDB e migra automaticamente gli utenti noti al pool di utenti.
- Esegui il flusso della password dimenticata per reimpostare la password per l'utente migrato.
- Accedi come nuovo utente, quindi elimina le risorse.

Go

SDK per Go V2

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
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))
}
```

Gestisci il trigger `MigrateUser` con una funzione 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)
}
```

Crea una struttura che esegue operazioni comuni.

```
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))
}
```

Crea una struttura che racchiude le azioni di 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
```

```
}
```

Crea una struttura che racchiude le azioni di 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
}
```

Crea una struttura che racchiuda le azioni di CloudWatch Logs.

```
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
}
```

Crea una struttura che racchiuda le azioni. CloudFormation

```
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
}
```

Eliminare le risorse.

```
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!")
 }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - ConfirmForgotPassword
  - DeleteUser
  - ForgotPassword
  - InitiateAuth
  - SignUp
  - UpdateUserPool

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Creazione di un'API REST di API Gateway per monitorare i dati COVID-19

Il seguente esempio di codice mostra come creare una REST API che simula un sistema per monitorare i casi quotidiani di COVID-19 negli Stati Uniti, utilizzando dati fittizi.

Python

SDK per Python (Boto3)

Mostra come usare AWS Chalice con per AWS SDK per Python (Boto3) creare un'API REST serverless che utilizzi Amazon API Gateway e Amazon DynamoDB. AWS Lambda La REST API simula un sistema che monitora i casi giornalieri di COVID-19 negli Stati Uniti, utilizzando dati fittizi. Scopri come:

- Usa AWS Chalice per definire i percorsi nelle funzioni Lambda che vengono chiamate per gestire le richieste REST che arrivano tramite API Gateway.
- Utilizza le funzioni Lambda per recuperare e archiviare i dati in una tabella DynamoDB per soddisfare le richieste REST.
- Definisci la struttura della tabella e le risorse dei ruoli di sicurezza in un AWS CloudFormation modello.
- Usa AWS Chalice e CloudFormation per impacchettare e distribuire tutte le risorse necessarie.
- Usa CloudFormation per ripulire tutte le risorse create.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Gateway API
- CloudFormation
- DynamoDB
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Creazione di una REST API per la libreria di prestiti

L'esempio di codice seguente mostra come creare una libreria di prestiti in cui gli utenti possono prendere in prestito e restituire libri tramite una REST API supportata da un database Amazon Aurora.

Python

SDK per Python (Boto3)

Mostra come utilizzare l' AWS SDK per Python (Boto3) API Amazon Relational Database Service (Amazon RDS) e AWS Chalice per creare un'API REST supportata da un database Amazon Aurora. Il servizio Web è completamente serverless e rappresenta una semplice libreria di prestiti in cui gli utenti possono prendere in prestito e restituire libri. Scopri come:

- Creare e gestire un cluster di database Aurora serverless.
- Utilizzalo per gestire le credenziali Gestione dei segreti AWS del database.
- Implementare un livello di archiviazione di dati che utilizza Amazon RDS per spostare i dati dentro e fuori dal database.
- Usa AWS Chalice per distribuire un'API REST serverless su Amazon API Gateway e. AWS Lambda
- Utilizza il pacchetto Richieste per inviare le richieste al servizio Web.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Gateway API
- Aurora
- Lambda
- Secrets Manager

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Creazione di un'applicazione di messaggistica con Step Functions

Il seguente esempio di codice mostra come creare un'applicazione di AWS Step Functions messaggistica che recupera i record dei messaggi da una tabella di database.

Python

SDK per Python (Boto3)

Mostra come usare AWS SDK per Python (Boto3) with per creare un'applicazione di messaggistica che AWS Step Functions recupera i record dei messaggi da una tabella Amazon DynamoDB e li invia con Amazon Simple Queue Service (Amazon SQS). La macchina a stati si integra con una AWS Lambda funzione per scansionare il database alla ricerca di messaggi non inviati.

- Crea una macchina a stati che recuperi e aggiorni i record di messaggi da una tabella Amazon DynamoDB.
- Aggiorna la definizione della macchina a stati per inviare messaggi anche ad Amazon Simple Queue Service (Amazon SQS).
- Avvia e arresta l'esecuzione della macchina a stati.
- Connettiti a Lambda, DynamoDB e Amazon SQS da una macchina a stati utilizzando le integrazioni di servizi.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda
- Amazon SQS
- Step Functions

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Creazione di un'applicazione di gestione delle risorse fotografiche che consente agli utenti di gestire le foto utilizzando etichette

L'esempio di codice seguente mostra come creare un'applicazione serverless che consente agli utenti di gestire le foto mediante etichette.

.NET

SDK per .NET

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

C++

SDK per C++

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

Java

SDK per Java 2.x

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

JavaScript

SDK per JavaScript (v3)

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

Kotlin

SDK per Kotlin

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB

- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

PHP

SDK per PHP

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

Rust

SDK per Rust

Mostra come sviluppare un'applicazione per la gestione delle risorse fotografiche che rileva le etichette nelle immagini utilizzando Amazon Rekognition e le archivia per recuperarle in seguito.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Per approfondire l'origine di questo esempio, consulta il post su AWS Community.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon Rekognition
- Simple Storage Service (Amazon S3)
- Amazon SNS

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Creazione di un'applicazione di chat websocket con API Gateway

L'esempio di codice seguente mostra come creare un'applicazione chat servita da un'API websocket creata su Gateway Amazon API.

Python

SDK per Python (Boto3)

Mostra come utilizzarlo AWS SDK per Python (Boto3) con Amazon API Gateway V2 per creare un'API websocket che si integri con Amazon AWS Lambda DynamoDB.

- Crea un'API WebSocket servita da Gateway API.
- Definisci un gestore Lambda che memorizzi le connessioni in DynamoDB e invii messaggi ad altri partecipanti alla chat.
- Connettiti all'applicazione di chat websocket e invia messaggi con il pacchetto Websockets.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Gateway API

- DynamoDB
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Crea un'applicazione che analizza il feedback dei clienti e sintetizza l'audio

Il seguente esempio di codice spiega come creare un'applicazione che analizza schede dei commenti dei clienti, le traduce dalla loro lingua originale, ne determina la valutazione e genera un file audio dal testo tradotto.

.NET

SDK per .NET

Questa applicazione di esempio analizza e archivia le schede di feedback dei clienti. In particolare, soddisfa l'esigenza di un hotel fittizio a New York City. L'hotel riceve feedback dagli ospiti in varie lingue sotto forma di schede di commento fisiche. Tale feedback viene caricato nell'app tramite un client Web. Dopo aver caricato l'immagine di una scheda di commento, vengono eseguiti i seguenti passaggi:

- Il testo viene estratto dall'immagine utilizzando Amazon Textract.
- Amazon Comprehend determina il sentiment del testo estratto e la sua lingua.
- Il testo estratto viene tradotto in inglese utilizzando Amazon Translate.
- Amazon Polly sintetizza un file audio dal testo estratto.

L'app completa può essere implementata con AWS CDK. Per il codice sorgente e le istruzioni di distribuzione, consulta il progetto in GitHub.

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Java

SDK per Java 2.x

Questa applicazione di esempio analizza e archivia le schede di feedback dei clienti. In particolare, soddisfa l'esigenza di un hotel fittizio a New York City. L'hotel riceve feedback dagli ospiti in varie lingue sotto forma di schede di commento fisiche. Tale feedback viene caricato nell'app tramite un client Web. Dopo aver caricato l'immagine di una scheda di commento, vengono eseguiti i seguenti passaggi:

- Il testo viene estratto dall'immagine utilizzando Amazon Textract.
- Amazon Comprehend determina il sentiment del testo estratto e la sua lingua.
- Il testo estratto viene tradotto in inglese utilizzando Amazon Translate.
- Amazon Polly sintetizza un file audio dal testo estratto.

L'app completa può essere implementata con AWS CDK. Per il codice sorgente e le istruzioni di distribuzione, consulta il progetto in GitHub.

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

JavaScript

SDK per JavaScript (v3)

Questa applicazione di esempio analizza e archivia le schede di feedback dei clienti. In particolare, soddisfa l'esigenza di un hotel fittizio a New York City. L'hotel riceve feedback dagli ospiti in varie lingue sotto forma di schede di commento fisiche. Tale feedback viene caricato nell'app tramite un client Web. Dopo aver caricato l'immagine di una scheda di commento, vengono eseguiti i seguenti passaggi:

- Il testo viene estratto dall'immagine utilizzando Amazon Textract.
- Amazon Comprehend determina il sentiment del testo estratto e la sua lingua.

- Il testo estratto viene tradotto in inglese utilizzando Amazon Translate.
- Amazon Polly sintetizza un file audio dal testo estratto.

L'app completa può essere implementata con AWS CDK. Per il codice sorgente e le istruzioni di distribuzione, consulta il progetto in. GitHub I seguenti estratti mostrano come AWS SDK per JavaScript viene utilizzato all'interno delle funzioni 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 };
};
```

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Ruby

SDK per Ruby

Questa applicazione di esempio analizza e archivia le schede di feedback dei clienti. In particolare, soddisfa l'esigenza di un hotel fittizio a New York City. L'hotel riceve feedback dagli ospiti in varie lingue sotto forma di schede di commento fisiche. Tale feedback viene caricato nell'app tramite un client Web. Dopo aver caricato l'immagine di una scheda di commento, vengono eseguiti i seguenti passaggi:

- Il testo viene estratto dall'immagine utilizzando Amazon Textract.
- Amazon Comprehend determina il sentiment del testo estratto e la sua lingua.
- Il testo estratto viene tradotto in inglese utilizzando Amazon Translate.

- Amazon Polly sintetizza un file audio dal testo estratto.

L'app completa può essere implementata con AWS CDK. Per il codice sorgente e le istruzioni di distribuzione, consulta il progetto in. GitHub

Servizi utilizzati in questo esempio

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Richiamo a una funzione Lambda da un browser

Il seguente esempio di codice mostra come richiamare una AWS Lambda funzione da un browser.

JavaScript

SDK per JavaScript (v2)

Puoi creare un'applicazione basata su browser che utilizza una AWS Lambda funzione per aggiornare una tabella Amazon DynamoDB con selezioni utente.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda

SDK per JavaScript (v3)

Puoi creare un'applicazione basata su browser che utilizza una AWS Lambda funzione per aggiornare una tabella Amazon DynamoDB con selezioni utente. Questa app utilizza la versione 3. AWS SDK per JavaScript

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Trasformare i dati per l'applicazione con S3 Object Lambda

L'esempio di codice seguente mostra come trasformare i dati per l'applicazione con S3 Object Lambda.

.NET

SDK per .NET

Mostra come aggiungere codice personalizzato alle richieste S3 GET standard per modificare l'oggetto richiesto recuperato da S3 in modo che questo soddisfi le esigenze del client o dell'applicazione richiedente.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Lambda
- Simple Storage Service (Amazon S3)

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo di Gateway API per invocare una funzione Lambda

I seguenti esempi di codice mostrano come creare una AWS Lambda funzione richiamata da Amazon API Gateway.

Java

SDK per Java 2.x

Mostra come creare una AWS Lambda funzione utilizzando l'API runtime Lambda Java. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. In questo esempio viene illustrato come creare una funzione Lambda invocata da Gateway Amazon API che analizza una tabella Amazon DynamoDB per le ricorrenze di lavoro e utilizza Amazon Simple Notification Service (Amazon SNS) per inviare un messaggio di testo ai dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon SNS

JavaScript

SDK per JavaScript (v3)

Mostra come creare una AWS Lambda funzione utilizzando l'API di JavaScript runtime Lambda. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. In questo esempio viene illustrato come creare una funzione Lambda invocata da Gateway Amazon API che analizza una tabella Amazon DynamoDB per le ricorrenze di lavoro e utilizza Amazon Simple Notification Service (Amazon SNS) per inviare un messaggio di testo ai dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK per JavaScript v3.

Servizi utilizzati in questo esempio

- Gateway API

- DynamoDB
- Lambda
- Amazon SNS

Python

SDK per Python (Boto3)

L'esempio mostra come creare e utilizzare una REST API di Gateway Amazon API destinata a una funzione AWS Lambda . Il gestore Lambda dimostra come definire route in base ai metodi HTTP, come ottenere dati dalla stringa, dall'intestazione e dal corpo della query e come restituire una risposta JSON.

- Distribuire una funzione Lambda.
- Creare una REST API di Gateway API.
- Creare una risorsa REST destinata alla funzione Lambda.
- Concedere l'autorizzazione affinché Gateway API invochi la funzione Lambda.
- Utilizza il pacchetto Richieste per inviare le richieste alla REST API.
- Eliminare tutte le risorse create durante la demo.

Questo esempio è visualizzato al meglio su GitHub. Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda
- Amazon SNS

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo di Step Functions per richiamare le funzioni Lambda

Il seguente esempio di codice mostra come creare una macchina a AWS Step Functions stati che richiama AWS Lambda funzioni in sequenza.

Java

SDK per Java 2.x

Mostra come creare un flusso di lavoro AWS serverless utilizzando AWS Step Functions and. AWS SDK for Java 2.x Ogni fase del flusso di lavoro viene implementata utilizzando una AWS Lambda funzione.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- DynamoDB
- Lambda
- Amazon SES
- Step Functions

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzo degli eventi pianificati per invocare una funzione Lambda

I seguenti esempi di codice mostrano come creare una AWS Lambda funzione richiamata da un evento EventBridge pianificato di Amazon.

Java

SDK per Java 2.x

Mostra come creare un evento EventBridge pianificato da Amazon che richiami una AWS Lambda funzione. Configura EventBridge per utilizzare un'espressione cron per pianificare quando viene richiamata la funzione Lambda. In questo esempio, viene creata una funzione Lambda utilizzando l'API di runtime Lambda Java. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Questo esempio dimostra come creare un'app che invia un messaggio di testo via mobile ai tuoi dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Servizi utilizzati in questo esempio

- CloudWatch Registri
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

JavaScript

SDK per JavaScript (v3)

Mostra come creare un evento EventBridge pianificato da Amazon che richiami una AWS Lambda funzione. Configura EventBridge per utilizzare un'espressione cron per pianificare quando viene richiamata la funzione Lambda. In questo esempio, crei una funzione Lambda utilizzando l'API JavaScript Lambda runtime. Questo esempio richiama diversi AWS servizi per eseguire un caso d'uso specifico. Questo esempio dimostra come creare un'app che invia un messaggio di testo via mobile ai tuoi dipendenti che si congratula con loro alla data dell'anniversario di un anno.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su. GitHub

Questo esempio è anche disponibile nella Guida per lo sviluppatore di AWS SDK per JavaScript v3.

Servizi utilizzati in questo esempio

- CloudWatch Registri
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Python

SDK per Python (Boto3)

Questo esempio mostra come registrare una AWS Lambda funzione come destinazione di un EventBridge evento Amazon pianificato. Il gestore Lambda scrive un messaggio intuitivo e i dati completi dell'evento su Amazon CloudWatch Logs per recuperarli in un secondo momento.

- Distribuzione di una funzione Lambda.
- Crea un evento EventBridge pianificato e rende la funzione Lambda la destinazione.
- Concede il permesso di EventBridge invocare la funzione Lambda.
- Stampa i dati più recenti dai CloudWatch registri per mostrare il risultato delle chiamate pianificate.
- Elimina tutte le risorse create durante la demo.

Questo esempio è visualizzato al meglio su. GitHub Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, vedi l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- CloudWatch Registri
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Utilizzare l'API Amazon Neptune per sviluppare una funzione Lambda che esegua query sui dati del grafico

L'esempio di codice seguente mostra come utilizzare l'API Neptune per eseguire query sui dati del grafico.

Java

SDK per Java 2.x

Mostra come utilizzare l'API Java di Amazon Neptune per creare una funzione Lambda che eseguano query sui dati del grafico all'interno del VPC.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Lambda
- Neptune

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Scrivi dati di attività personalizzati con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito tramite un SDK AWS

L'esempio di codice seguente mostra come scrivere dati di attività personalizzate con una funzione Lambda dopo l'autenticazione utente di Amazon Cognito.

- Usa le funzioni di amministratore per aggiungere un utente a un pool di utenti.
- Configura un pool di utenti per chiamare una funzione Lambda per il trigger `PostAuthentication`.
- Accedere con il nuovo utente ad Amazon Cognito.
- La funzione Lambda scrive informazioni personalizzate nei CloudWatch log e in una tabella DynamoDB.
- Acquisisci e visualizza dati personalizzati dalla tabella DynamoDB, quindi elimina le risorse.

Go

SDK per Go V2

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri di più sulla configurazione e l'esecuzione nel Repository di esempi di codice AWS.

Esegui uno scenario interattivo al prompt dei comandi.

```
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))
}
```

Gestisci il trigger `PostAuthentication` con una funzione 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)
}
```

Crea una struttura che esegue operazioni comuni.

```
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))
}
```

Crea una struttura che racchiude le azioni di 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
}
```

Crea una struttura che racchiude le azioni di 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
}
```

Crea una struttura che racchiuda le azioni di CloudWatch Logs.

```
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
}
```

Crea una struttura che racchiuda le azioni. CloudFormation

```
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
}
```

Eliminare le risorse.

```
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!")
  }
}
```

- Per informazioni dettagliate sull'API, consulta i seguenti argomenti nella documentazione di riferimento dell'API AWS SDK per Go .
  - AdminCreateUser

- AdminSetUserPassword
- DeleteUser
- InitiateAuth
- UpdateUserPool

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consulta. Usare Lambda con un SDK AWS Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Esempi serverless per Lambda

I seguenti esempi di codice mostrano come usare Lambda con. AWS SDKs

Esempi

- Connessione a un database Amazon RDS in una funzione Lambda
- Invocare una funzione Lambda da un trigger Kinesis
- Richiamare una funzione Lambda da un trigger DynamoDB
- Richiamare una funzione Lambda da un trigger Amazon DocumentDB
- Invocare una funzione Lambda da un trigger Amazon MSK
- Invocazione di una funzione Lambda da un trigger Amazon S3
- Invocazione di una funzione Lambda da un trigger Amazon SNS
- Richiamo di una funzione Lambda da un trigger Amazon SQS
- Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis
- Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB
- Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS

Connessione a un database Amazon RDS in una funzione Lambda

Gli esempi di codice seguenti mostrano come implementare una funzione Lambda che si connette a un database RDS. La funzione effettua una semplice richiesta al database e restituisce il risultato.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda con .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda utilizzando. 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)
  }
};
```

Connessione a un database Amazon RDS in una funzione Lambda utilizzando. 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 per PHP

> **Note**
>
> C'è di più su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Connessione a un database Amazon RDS in una funzione Lambda tramite 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}")
    }))
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Invocare una funzione Lambda da un trigger Kinesis

I seguenti esempi di codice spiegano come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso Kinesis. La funzione recupera il payload Kinesis, lo decodifica da Base64 e registra il contenuto del record.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento Kinesis con Lambda utilizzando. 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;
}
```

Consumo di un evento Kinesis con Lambda utilizzando. 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Kinesis con Lambda tramite 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
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Richiamare una funzione Lambda da un trigger DynamoDB

I seguenti esempi di codice spiegano come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso DynamoDB. La funzione recupera il payload DynamoDB e registra il contenuto del record.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento DynamoDB con Lambda utilizzando. 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)}`);
};
```

Consumo di un evento DynamoDB con Lambda utilizzando. 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento DynamoDB con Lambda tramite 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(())

}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Richiamare una funzione Lambda da un trigger Amazon DocumentDB

I seguenti esempi di codice spiegano come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un flusso di modifiche di DocumentDB. La funzione recupera il payload DocumentDB e registra il contenuto del record.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento Amazon DocumentDB con Lambda utilizzando. 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));
};
```

Consumo di un evento Amazon DocumentDB con Lambda utilizzando 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon DocumentDB con Lambda tramite Ruby.

```
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(())

}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Invocare una funzione Lambda da un trigger Amazon MSK

I seguenti esempi di codice spiegano come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di record da un cluster Amazon MSK. La funzione recupera il payload MSK e registra il contenuto del record.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento Amazon MSK con JavaScript Lambda utilizzando.

```
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)
```

```
      })
    }
}
```

Consumo di un evento Amazon MSK con TypeScript Lambda utilizzando.

```
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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento Amazon MSK con Lambda tramite 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
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Invocazione di una funzione Lambda da un trigger Amazon S3

I seguenti esempi di codice mostrano come implementare una funzione Lambda che riceve un evento attivato dal caricamento di un oggetto in un bucket S3. La funzione recupera il nome del bucket S3 e la chiave dell'oggetto dal parametro evento e chiama l'API Amazon S3 per recuperare e registrare il tipo di contenuto dell'oggetto.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento S3 con JavaScript Lambda utilizzando.

```
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);
    }
};
```

Consumo di un evento S3 con TypeScript Lambda utilizzando.

```
// 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento S3 con Lambda tramite 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(())
```

```
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Invocazione di una funzione Lambda da un trigger Amazon SNS

I seguenti esempi di codice mostrano come implementare una funzione Lambda che riceve un evento attivato dal ricevimento di messaggi da un argomento SNS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento SNS con JavaScript Lambda utilizzando.

```
// 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;
  }
}
```

Consumo di un evento SNS con TypeScript Lambda utilizzando.

```
// 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SNS con Lambda tramite 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
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Richiamo di una funzione Lambda da un trigger Amazon SQS

I seguenti esempi di codice spiegano come implementare una funzione Lambda che riceve un evento attivato dalla ricezione di messaggi da una coda SQS. La funzione recupera i messaggi dal parametro dell'evento e registra il contenuto di ogni messaggio.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Consumo di un evento SQS con JavaScript Lambda utilizzando.

```
// 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;
  }
}
```

Consumo di un evento SQS con TypeScript Lambda utilizzando.

```
// 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Utilizzo di un evento SQS con Lambda tramite 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
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Kinesis

I seguenti esempi di codice spiegano come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da un flusso Kinesis. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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;
}
```

Segnalazione degli errori degli elementi batch di Kinesis utilizzando 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di Kinesis con Lambda tramite 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
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger DynamoDB

I seguenti esempi di codice spiegano come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da un flusso DynamoDB. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione degli errori degli elementi batch di DynamoDB utilizzando 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: [] };
};
```

Segnalazione degli errori degli elementi batch di DynamoDB utilizzando 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di DynamoDB con Lambda tramite 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
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

Segnalazione di errori di elementi batch per funzioni Lambda con un trigger Amazon SQS

I seguenti esempi di codice spiegano come implementare una risposta batch parziale per funzioni Lambda che ricevono eventi da una coda SQS. La funzione riporta gli errori degli elementi batch nella risposta, segnalando a Lambda di riprovare tali messaggi in un secondo momento.

.NET

SDK per .NET

> **Note**
>
> C'è altro su. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite .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 per Go V2

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 per Java 2.x

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 per JavaScript (v3)

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione degli errori degli elementi batch SQS utilizzando 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}`);
}
```

Segnalazione degli errori degli elementi batch SQS utilizzando 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 per PHP

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 per Python (Boto3)

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 per Ruby

> **Note**
>
> C'è dell'altro GitHub. Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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 per Rust

> **Note**
>
> C'è altro da fare. GitHub Trova l'esempio completo e scopri come eseguire la configurazione e l'esecuzione nel repository di Esempi serverless.

Segnalazione di errori di elementi batch di SQS con Lambda tramite 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
}
```

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

AWS contributi della community per Lambda

AWS i contributi della community sono esempi che sono stati creati e gestiti da più team all'interno AWS. Per fornire un feedback, utilizza il meccanismo disponibile nei repository collegati.

Esempi

- Creare e testare un'applicazione serverless

Creare e testare un'applicazione serverless

Gli esempi di codice seguenti mostrano come creare e testare un'applicazione serverless utilizzando Gateway API con Lambda e DynamoDB

.NET

SDK per .NET

Mostra come creare e testare un'applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l'SDK per .NET.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, consulta l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

Go

SDK per Go V2

Mostra come creare e testare un'applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l'SDK per Go.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

Java

SDK per Java 2.x

Mostra come creare e testare un'applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l'SDK per Java.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

Rust

SDK per Rust

Mostra come creare e testare un'applicazione serverless composta da Gateway API con Lambda e DynamoDB utilizzando l'SDK per Rust.

Per il codice sorgente completo e le istruzioni su come configurarlo ed eseguirlo, guarda l'esempio completo su GitHub.

Servizi utilizzati in questo esempio

- Gateway API
- DynamoDB
- Lambda

Per un elenco completo delle guide per sviluppatori AWS SDK e degli esempi di codice, consultaUsare Lambda con un SDK AWS. Questo argomento include anche informazioni su come iniziare e dettagli sulle versioni precedenti dell'SDK.

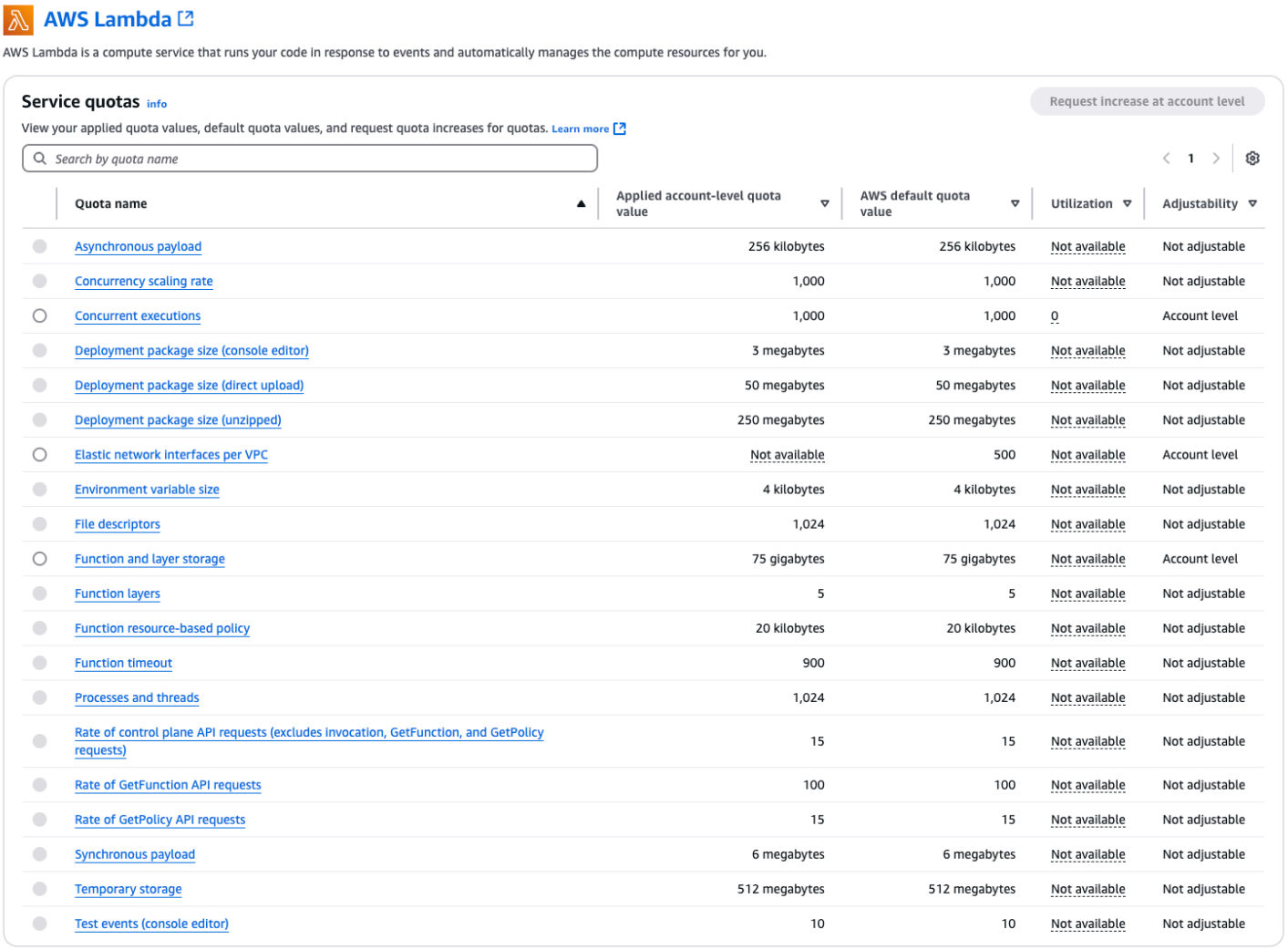
Quote di Lambda

> **Important**
>
> Account AWS I nuovi hanno quote di concorrenza e memoria ridotte. AWS aumenta automaticamente queste quote in base all'utilizzo.

AWS Lambda è progettato per scalare rapidamente per soddisfare la domanda, permettendo alle funzioni di scalare per soddisfare il traffico dell'applicazione. Lambda è progettata per attività di elaborazione di breve durata che non mantengono o si basano sullo stato tra le chiamate. Il codice può essere eseguito per un massimo di 15 minuti in una singola chiamata e una singola funzione può utilizzare fino a 10.240 MB di memoria.

È importante comprendere le barriere messe in atto per proteggere il proprio account e i carichi di lavoro degli altri clienti. Le quote di servizio esistono in tutti i AWS servizi e consistono in limiti rigidi, che non è possibile modificare, e limiti flessibili, per i quali è possibile richiedere aumenti. Per impostazione predefinita, a tutti i nuovi account viene assegnato un profilo di quota che consente l'esplorazione dei AWS servizi.

Per visualizzare le quote applicabili al tuo account, vai al pannello di controllo di Service Quotas. Qui è possibile visualizzare le quote di servizio, richiedere un aumento delle quote e visualizzare l'utilizzo corrente. Da qui, puoi approfondire un AWS servizio specifico, come 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

Le seguenti sezioni elencano le quote e i limiti predefiniti in Lambda per categoria.

Argomenti

- Elaborazione e archiviazione
- Configurazione, implementazione ed esecuzione della funzione
- Richieste API Lambda
- Altri servizi

Elaborazione e archiviazione

Lambda imposta le quote per la quantità di risorse di calcolo e storage che è possibile utilizzare per eseguire e archiviare le funzioni. Le quote per le esecuzioni e l'archiviazione simultanee sono

applicate in base alla Regione AWS. Le quote dell'interfaccia di rete elastica (ENI) sono applicate in base al cloud privato virtuale (VPC), indipendentemente dalla regione. Le seguenti quote possono essere aumentate rispetto ai relativi valori predefiniti. Per ulteriori informazioni, consulta Richiesta di un aumento di quota nella Guida per l'utente delle Service Quotas.

Risorsa	Quota predefinita	Può essere aumentato fino a
Esecuzioni simultanee	1.000	Decine di migliaia
Storage per funzioni caricate (archivi di file .zip) e livelli. Ogni versione di funzione e di livello consuma spazio di storage. Per le best practice da seguire per la gestione dell'archiviazione del codice, consulta Monitoraggio dell'archiviazione del codice Lambda in Serverless Land.	75 GB	Terabyte
Storage per le funzioni definite come immagini di container. Queste immagini sono memorizzate in Amazon ECR.	Consulta Service Quotas di Amazon ECR.	
Interfacce di rete elastiche per Virtual Private Cloud (VPC) **Note** Questa quota è condivisa con altri servizi, ad esempio, Amazon Elastic File System (Amazon EFS). Consulta Quote Amazon VPC.	500	Migliaia

Per ulteriori informazioni sulla simultaneità e su come Lambda ridimensiona la simultaneità della funzione in risposta al traffico, consulta Informazioni sulla scalabilità della funzione Lambda.

Configurazione, implementazione ed esecuzione della funzione

Le seguenti quote si applicano alla configurazione, all'implementazione e all'esecuzione della funzione. Fatto salvo per quanto indicato, non possono essere modificate.

> **Note**
>
> La documentazione Lambda, i messaggi di log e la console utilizzano l'abbreviazione MB (anziché MiB) per fare riferimento a 1.024 KB.

Risorsa	Quota
Allocazione di memoria della funzione	Da 128 MB a 10.240 MB, in incrementi di 1 MB. Nota: Lambda alloca la potenza della CPU in proporzione alla quantità di memoria configurata. È possibile aumentare o diminuire la memoria e la potenza della CPU assegnate alla funzione utilizzando l'imposta zione Memory (MB). A 1.769 MB, una funzione ha l'equivalente di una vCPU.
Timeout della funzione.	900 secondi (15 minuti)
Variabili di ambiente della funzione	4 KB, per tutte le variabili di ambiente associate alla funzione, in forma aggregata
Policy basata sulle risorse della funzione	20 KB
Livelli della funzione	5 strati
Limite di dimensionamento della simultaneità delle funzioni	Per ogni funzione, 1.000 ambienti di esecuzione ogni 10 secondi

Risorsa	Quota
Payload dell'invocazione (richiesta e risposta)	6 MB ciascuno per richiesta e risposta (sincrono) 200 MB per ogni risposta in streaming (sincrona) 1 MB (asincrono) 1 MB per la dimensione totale combinata dei valori della riga di richiesta e dell'intestazione
Larghezza di banda per le risposte in streaming	Senza limite per i primi 6 MB di risposta della funzione Per risposte superiori a 6 MB, 2 MBps per il resto della risposta
Dimensioni del pacchetto di implementazione (archivio di file .zip)	50 MB (zippato, se caricato tramite l'API SDKs Lambda o). Carica file di dimensioni maggiori con Amazon S3. 50 MB (se caricati tramite la console Lambda) 250 MB La dimensione massima del contenuto di un pacchetto di implementazione, inclusi livelli e runtime personalizzati (decompresso).
Impostazioni dell'immagine di container	16 KB
Dimensione del pacchetto del codice dell'immagine di container	10 GB (dimensione massima dell'immagine non compressa, inclusi tutti i livelli)

Risorsa	Quota
Eventi di test (editor della console)	10
Storage della directory `/tmp`	Compreso tra 512 MB e 10.240 MB in incrementi di 1 MB
Descrittori di file	1,024 **Note** Le istanze gestite Lambda utilizzano i limiti predefiniti dei descrittori di file di Bottleroc ket. Per ulteriori informazi oni, consulta Comprendere l'ambiente di esecuzione di Lambda Managed Instances.
Processi/thread dell'esecuzione	1,024 **Note** Le istanze gestite Lambda utilizzano i limiti di processo e thread predefiniti di Bottleroc ket. Per ulteriori informazi oni, consulta Comprendere l'ambiente di esecuzione di Lambda Managed Instances.

Richieste API Lambda

Le seguenti quote sono associate alle richieste API Lambda.

Risorsa	Quota
Richieste di chiamata per funzione per regione (sincrona)	Ogni istanza del tuo ambiente di esecuzione può gestire fino a 10 richieste al secondo. In altre parole, il limite totale di chiamate è 10 volte il limite di simultaneità. Per informazioni, consulta Informazioni sulla scalabilità della funzione Lambda.
Richieste di chiamata per funzione per regione (asincrona)	Ogni istanza del tuo ambiente di esecuzione può soddisfare un numero illimitato di richieste. In altre parole, il limite totale di chiamate si basa solo sulla simultaneità disponibile per la funzione. Per informazioni, consulta Informazioni sulla scalabilità della funzione Lambda.
Richieste di invocazione per versione di funzione o alias (richieste al secondo)	10 x simultaneità fornita allocata **Note** Questa quota si applica solo alle funzioni che utilizzano la simultaneità fornita.
Richieste API GetFunction	100 richieste al secondo. Non può essere aumentato.
Richieste API GetPolicy	15 richieste al secondo. Non può essere aumentato.
Il resto delle richieste API del piano di controllo (escluse le chiamate e le richieste) GetFunction GetPolicy	15 richieste al secondo in tutto APIs (non 15 richieste al secondo per API). Non può essere aumentato.

Altri servizi

Le quote per altri servizi, come AWS Identity and Access Management (IAM), Amazon CloudFront (Lambda @Edge) e Amazon Virtual Private Cloud (Amazon VPC), possono influire sulle funzioni Lambda. Per ulteriori informazioni, consulta la pagina Servizio AWS quotas nella Riferimenti generali di Amazon Web Services e la pagina Richiamare Lambda con eventi di altri servizi AWS.

Molte applicazioni che coinvolgono Lambda utilizzano più AWS servizi. Poiché servizi diversi hanno quote diverse per diverse funzionalità, può essere difficile gestire queste quote nell'intera applicazione. Ad esempio, API Gateway ha un limite di accelerazione predefinito di 10.000 richieste al secondo, mentre Lambda ha un limite di concorrenza predefinito di 1.000. A causa di questa mancata corrispondenza, è possibile che Lambda sia in grado di gestire un numero maggiore di richieste in arrivo da API Gateway. Puoi risolvere questo problema richiedendo un aumento del limite di concorrenza Lambda in modo che corrisponda al livello di traffico previsto.

Il test di carico dell'applicazione consente di monitorare le prestazioni dell'applicazione end-to-end prima della distribuzione in produzione. Durante un test di carico, è possibile identificare eventuali quote che possono fungere da fattore limitante per i livelli di traffico previsti e agire di conseguenza.

Cronologia dei documenti

Nella tabella seguente sono descritte le modifiche importanti apportate alla Guida per sviluppatori di AWS Lambda a partire da maggio 2018. Per ricevere notifiche sugli aggiornamenti della documentazione, è possibile iscriversi al feed RSS.

Modifica	Descrizione	Data
AWS aggiornamenti delle politiche gestiti	Lambda ha rilasciato una nuova policy gestita `AWSLambdaBasicDurableExecutionRolePolicy` che fornisce autorizzazioni di scrittura per CloudWatch i log e read/write autorizzazioni per l'esecuzione durevole utilizzate APIs dalle funzioni durevoli Lambda.	1 dicembre 2025
Ruolo collegato ai servizi per Lambda	Lambda ora supporta un ruolo collegato al servizio () `AWSServiceRoleForLambda` che consente a Lambda di terminare le istanze gestite come parte dei provider di capacità Lambda.	30 novembre 2025
Istanze gestite Lambda	Lambda ora supporta le istanze gestite, che forniscono capacità di elaborazione dedicata per le tue funzioni con prestazioni, sicurezza e controllo migliorati. Per i dettagli, consulta Lambda Managed Instances.	30 novembre 2025

AWS aggiornamenti delle politiche gestite	Lambda ha aggiunto una nuova policy AWS gestita ()`AWSLambdaBasicDurableExecutionRolePolicy` .	30 novembre 2025
AWS aggiornamenti delle politiche gestite	Lambda ha aggiornato la politica AWS gestita esistente `AWSLambda_FullAccess`) consenti le azioni `kms:DescribeKey` `eiam:CreateServiceLinkedRole` .	30 novembre 2025
AWS aggiornamento gestito della politica	Lambda ha aggiunto una nuova policy AWS gestita (`AWSLambdaManagedEC2ResourceOperator`) per abilitare la gestione automatizzata delle EC2 istanze Amazon per i provider di capacità Lambda. Per i dettagli, consulta Aggiornamenti Lambda alle policy AWS gestite.	30 novembre 2025
AWS aggiornamento gestito della politica	Lambda ha aggiunto una nuova policy AWS gestita ()`AWSLambdaServiceRolePolicy` . Per i dettagli, consulta Aggiornamenti Lambda alle policy AWS gestite.	30 novembre 2025

AWS aggiornamenti delle politiche gestite	Lambda ha aggiornato due policy AWS gestite esistenti (`AWSLambda_ReadOnly Access` e `AWSLambda _FullAccess`).	20 febbraio 2025
Runtime di Node.js 22.x	Lambda ora supporta Node.js 22 come runtime gestito e immagine di base del container (`nodejs22.x`).	22 novembre 2024
SnapStart supporto per Python e.NET	Lambda SnapStart è ora disponibile per i runtime gestiti in Python e.NET, a partire da e. `python3.12` `dotnet8`	18 novembre 2024
Runtime di Python 3.13	Ora Lambda supporta Python 3.13 come runtime gestito e immagine di base del container.	13 novembre 2024
Crittografia gestita dal cliente per pacchetti di implement azione .zip	Lambda ora supporta la crittografia a chiave gestita AWS KMS dal cliente per i pacchetti di distribuzione.zip.	8 novembre 2024
Support SnapStart per nuove regioni	Lambda SnapStartè ora disponibile nelle seguenti regioni: Europa (Spagna), Europa (Zurigo), Asia Pacifico (Melbourne), Asia Pacifico (Hyderabad) e Medio Oriente (Emirati Arabi Uniti).	12 gennaio 2024
AWS aggiornamenti delle politiche gestite	Lambda ha aggiornato una policy AWS gestita esistente ()`AWSLambdaVPCAccess ExecutionRole` .	5 gennaio 2024

Runtime di Python 3.12	Ora Lambda supporta Python 3.12 come runtime gestito e immagine di base del container. Per ulteriori informazioni, consulta Python 3.12 runtime ora disponibile nel blog di AWS Lambda Compute AWS .	14 dicembre 2023
Runtime Java 21	Lambda ora supporta Java 21 come runtime gestito e immagine di base del container (`java21`).	16 novembre 2023
Runtime di Node.js 20.x	Lambda ora supporta Node.js 20 come runtime gestito e immagine di base del container (`nodejs20.x`). Per ulteriori informazioni, consulta il runtime di Node.js 20.x ora disponibile nel AWS Lambda Compute Blog. AWS	14 novembre 2023
Runtime provided.al2023	Lambda ora supporta Amazon Linux 2023 come runtime gestito e immagine di base del container. Per ulteriori informazioni, consulta la sezione Introduzione al runtime di Amazon Linux 2023 AWS Lambda sul blog di AWS Compute.	9 novembre 2023

IPv6 supporto per sottoreti dual-stack	Lambda ora supporta il IPv6 traffico in uscita verso sottoreti dual-stack. Per ulteriori informazioni, consulta IPv6 support.	12 ottobre 2023
Test di funzioni e applicazioni serverless	Scopri le tecniche per eseguire il debug e automatizzare i test delle funzioni serverles s nel cloud. Ora è presente un capitolo sui test e nelle sezioni dei linguaggi Python e Typescript sono state incluse delle risorse in merito. Per ulteriori informazioni, consulta la sezione Test di funzioni e applicazioni serverless.	16 giugno 2023
Runtime Ruby 3.2	Lambda ora supporta un nuovo runtime per Ruby 3.2. Per ulteriori informazioni, consulta la sezione Compilazione di funzioni Lambda con Ruby.	7 giugno 2023
Streaming delle risposte	Lambda ora supporta lo streaming delle risposte dalle funzioni. Per ulteriori informazioni, consulta la pagina Configurazione di una funzione Lambda per lo streaming delle risposte.	6 aprile 2023

Parametri di chiamata asincrona	Lambda rilascia parametri di chiamata asincrona. Per ulteriori informazioni, consulta Parametri di chiamata asincrona.	9 febbraio 2023
Controlli della versione di runtime	Lambda rilascia nuove versioni di runtime che includono aggiornamenti di sicurezza , correzioni di bug e nuove funzionalità. Ora puoi controlla re quando le tue funzioni vengono aggiornate alle nuove versioni di runtime. Per ulteriori informazioni, consulta Aggiornamenti di runtime Lambda.	23 gennaio 2023
Lambda SnapStart	Usa Lambda SnapStart per ridurre i tempi di avvio delle funzioni Java senza fornire risorse aggiuntive o implement are complesse ottimizza zioni delle prestazioni. Per ulteriori informazioni, consulta Migliorare le prestazioni di avvio con SnapStart Lambda.	28 novembre 2022
Runtime di Node.js 18	Lambda ora supporta un nuovo runtime per Node.js 18. Node.js 18 utilizza Amazon Linux 2. Per ulteriori dettagli, consultare Compilazione di funzioni Lambda con Node.js.	18 novembre 2022

lambda: SourceFunctionArn chiave di condizione	Per una AWS risorsa, la chiave `lambda:SourceFunctionArn` condition filtra l'accesso alla risorsa tramite l'ARN di una funzione Lambda. Per maggiori dettagli, consulta Utilizzo delle credenziali dell'ambiente di esecuzione Lambda.	1 luglio 2022
Runtime di Node.js 16	Lambda ora supporta un nuovo runtime per Node.js 16. Node.js 16 utilizza Amazon Linux 2. Per ulteriori dettagli, consultare Compilazione di funzioni Lambda con Node.js.	11 maggio 2022
Funzione Lambda URLs	Lambda ora supporta le funzioni URLs, che sono endpoint HTTP (S) dedicati per le funzioni Lambda. Per i dettagli, consulta Funzione Lambda. URLs	6 aprile 2022
Eventi di test condivisi nella console AWS Lambda	Lambda ora supporta la condivisione di eventi di test con altri utenti nello stesso Account AWS. Per informazioni dettagliate, consulta Test delle funzioni Lambda nella console.	16 marzo 2022

PrincipalOrgId nelle politiche basate sulle risorse	Lambda ora supporta la concessione delle autorizzazioni a un'organizzazione in AWS Organizations. Per informazioni dettagliate, consulta Utilizzo delle policy basate su risorse per AWS Lambda.	11 marzo 2022
Runtime .NET 6	Lambda ora supporta un nuovo runtime per .NET 6. Per informazioni dettagliate, consultare Runtime di Lambda.	23 febbraio 2022
Filtraggio di eventi per le origini di eventi Kinesis, DynamoDB e Amazon SQS	Lambda ora supporta il filtraggio di eventi per le origini di eventi Kinesis, DynamoDB e Amazon SQS. Per informazioni dettagliate, consulta Filtro eventi Lambda.	24 novembre 2021
Autenticazione mTLS per Amazon MSK e origine eventi Apache Kafka autogestito	Lambda ora supporta l'autenticazione mTLS per Amazon MSK e le origini eventi Apache Kafka autogestito. Per informazioni dettagliate, consultare Uso di Lambda con Amazon MSK.	19 novembre 2021
Lambda su Graviton2	Lambda ora supporta Graviton2 per le funzioni che utilizzano l'architettura arm64. Per informazioni dettagliate, consulta Architetture del set di istruzioni Lambda.	29 settembre 2021

Runtime di Python 3.9	Lambda ora supporta un nuovo runtime per Python 3.9. Per informazioni dettagliate, consultare Runtime di Lambda.	16 agosto 2021
Nuove versioni di runtime per Node.js, Python e Java	Sono disponibili nuove versioni del runtime per Node.js, Python e Java. Per informazioni dettagliate, consultare Runtime di Lambda.	21 luglio 2021
Supporto per RabbitMQ come origine evento su Lambda	Lambda ora supporta Amazon MQ per RabbitMQ come origine evento. Amazon MQ è un servizio gestito di broker di messaggistica per Apache ActiveMQ e RabbitMQ che facilita la configurazione e la gestione di broker di messaggistica nel cloud. Per informazioni dettagliate, consultare Uso di Lambda con Amazon MQ.	7 luglio 2021

Autenticazione SASL/PLAIN per Kafka autogestito su Lambda	SASL/PLAIN is now a supported authentication mechanism for self-managed Kafka event sources on Lambda Customers already using SASL/PLAINsul loro cluster Kafka autogestito ora possono usare facilmente Lambda per creare applicazioni consumer senza dover modificare il modo in cui si autenticano. Per informazioni dettagliate, consultare Uso di Lambda con Apache Kafka autogestito.	29 giugno 2021
API estensioni Lambda	Disponibilità generale per le estensioni Lambda. Utilizza le estensioni per potenziare le funzioni Lambda. È possibile utilizzare le estensioni fornite dai Lambda Partners oppure creare le tue estensioni Lambda. Per informazioni dettagliate, consultare API Extensions di Lambda.	24 maggio 2021
Nuova esperienza della console Lambda	La console Lambda è stata riprogettata per migliorare le prestazioni e la coerenza.	2 marzo 2021
Runtime di Node.js 14	Lambda ora supporta un nuovo runtime per Node.js 14. Node.js 14 utilizza Amazon Linux 2. Per ulteriori dettagli, consultare Compilazione di funzioni Lambda con Node.js.	27 gennaio 2021

Immagini di container Lambda	Lambda ora supporta le funzioni definite come immagini di container. È possibile combinare la flessibilità degli strumenti per container con l'agilità e la semplicità operativa di Lambda per costruire applicazioni. Per ulteriori dettagli, consultare Uso di immagini di container con Lambda.	1 dicembre 2020
Firma del codice per le funzioni Lambda	Lambda ora supporta la firma del codice. Gli amministratori possono configurare le funzioni Lambda in modo da accettare al momento della distribuzione solo codice firmato. Lambda controlla le firme per assicurarsi che il codice non venga alterato o manomesso. Inoltre, Lambda assicura che il codice sia firmato da sviluppatori affidabili prima di accettarne la distribuzione. Per informazioni dettagliate, consultare l'argomento Configurazione della firma del codice per Lambda.	23 novembre 2020

Anteprima: API Runtime Logs di Lambda	Lambda ora supporta l'API Runtime Logs. Le estensioni Lambda possono utilizzare l'API Logs per eseguire la sottoscrizione ai flussi di log nell'ambiente di esecuzione. Per informazioni dettagliate, consultare API Runtime Logs di Lambda.	12 novembre 2020
Nuova origine evento per Amazon MQ	Lambda ora supporta Amazon MQ come origine evento. Utilizza una funzione Lambda per elaborare i record dal broker di messaggi Amazon MQ. Per informazioni dettagliate, consultare Uso di Lambda con Amazon MQ.	5 novembre 2020
Anteprima: API estensioni Lambda	Utilizza le estensioni Lambda per potenziare le funzioni Lambda. È possibile utilizzare le estensioni fornite dai Lambda Partners oppure creare le tue estensioni Lambda. Per informazioni dettagliate, consultare API Extensions di Lambda.	8 ottobre 2020
Support per Java 8 e runtime personalizzati su AL2	Lambda supporta ora Java 8 e runtime personalizzati su Amazon Linux 2. Per informazioni dettagliate, consultare Runtime di Lambda.	12 agosto 2020

Nuova origine evento per Amazon Managed Streaming for Apache Kafka	Lambda ora supporta Amazon MSK come origine evento. Utilizza una funzione Lambda con Amazon MSK per elaborare i record in un argomento Kafka. Per informazioni dettagliate, consultare Uso di Lambda con Amazon MSK.	11 agosto 2020
Chiavi di condizione IAM per le impostazioni di Amazon VPC	Ora è possibile utilizzare le chiavi di condizione specifiche di Lambda per le impostazioni VPC. Ad esempio, è possibile richiedere che tutte le funzioni dell'organizzazione siano connesse a un VPC. È inoltre possibile specificare le sottoreti e i gruppi di sicurezza che gli utenti della funzione possono e non possono utilizzare. Per informazioni dettagliate, consultare Configurazione di VPC per le funzioni IAM.	10 agosto 2020

Impostazioni di concorrenza per i consumer del flusso Kinesis HTTP/2	Ora puoi utilizzare le seguenti impostazioni di concorrenza per gli utenti Kinesis con fan-out avanzato (flussi HTTP/2):, ,, e. ParallelizationFactor MaximumRetryAttempts MaximumRecordAgeIn Seconds DestinationConfig BisectBatchOnFunctionError Per i dettagli, consulta Utilizzo AWS Lambda con Amazon Kinesis.	7 luglio 2020
Periodo di batch per i consumer dei flussi Kinesis HTTP/2	Ora puoi configurare una finestra batch (MaximumBatchingWindowInSeconds) per i flussi HTTP/2. Lambda legge i record dal flusso fino a quando non ha raccolto un batch completo o fino alla scadenza del periodo di batch. Per i dettagli, consulta Utilizzo AWS Lambda con Amazon Kinesis.	18 giugno 2020
Supporto per i file system Amazon EFS	È ora possibile connettere un file system Amazon EFS alle proprie funzioni Lambda per l'accesso condiviso ai file di rete. Per informazioni dettagliate, consultare Configurazione dell'accesso al file system per le funzioni Lambda.	16 giugno 2020

AWS CDK applicazioni di esempio nella console Lambda	La console Lambda ora include applicazioni di esempio che utilizzano for. AWS Cloud Development Kit (AWS CDK) TypeScript AWS CDK È un framework che consente di definire le risorse dell'applicazione in Python TypeScript, Java o.NET.	1 giugno 2020
Support per il runtime di.NET Core 3.1.0 in AWS Lambda	AWS Lambda ora supporta il runtime di.NET Core 3.1.0. Per i dettagli, consultare Interfaccia a riga di comando (CLI) di .NET Core.	31 marzo 2020
Support per API Gateway HTTP APIs	Documentazione aggiornata e ampliata per l'utilizzo di Lambda con API Gateway, incluso il supporto per HTTP. APIs È stata aggiunta un'applicazione di esempio che crea un'API e una funzione con CloudFormation. Per informazioni dettagliate, consultare Uso di Lambda con Gateway Amazon API.	23 marzo 2020
Ruby 2.7	Un nuovo runtime è disponibile per Ruby 2.7, ruby2.7, che è il primo runtime Ruby a utilizzare Amazon Linux 2. Per informazioni dettagliate, consultare Compilazione di funzioni Lambda con Ruby.	19 febbraio 2020

Parametri di concorrenza	Lambda ora riporta i parametri `ConcurrentExecutions` per tutte le funzioni, gli alias e le versioni. È possibile visualizzare un grafico per questo parametro nella pagina di monitoraggio della funzione. In precedenza, `ConcurrentExecutions` è stato segnalato solo a livello di account e per le funzioni che utilizzano la concorrenza riservata. Per ulteriori dettagli, consultare Parametri delle funzioni di AWS Lambda.	18 febbraio 2020

Aggiornamento agli stati delle funzioni	Gli stati delle funzioni vengono ora applicati per impostazione predefinita a tutte le funzioni. Quando si connette una funzione a un VPC, Lambda crea interfacce di rete elastiche condivise. Ciò consente alla funzione di dimensionarsi senza creare interfacce di rete aggiuntive. Durante questo periodo, non è possibile eseguire ulteriori operazioni sulla funzione, incluso l'aggiornamento della configurazione e la pubblicazione delle versioni. In alcuni casi, viene influenzata anche la chiamata. I dettagli sullo stato corrente di una funzione sono disponibili dall'API Lambda. Questo aggiornamento viene rilasciato in fasi. Per i dettagli, consulta il ciclo di vita degli stati Lambda aggiornato per le reti VPC sul blog di Compute. AWS Per ulteriori informazioni sugli stati delle funzioni, consultare Stati delle funzioni AWS Lambda.	24 gennaio 2020

Aggiornamenti all'output dell'API di configurazione delle funzioni	Sono stati aggiunti codici motivo a StateReasonCode(InvalidSubnet, InvalidSecurityGroup) e LastUpdateStatusReasonCode (SubnetOutOfIPAddresses, InvalidSubnet, InvalidSecurityGroup) per le funzioni che si connettono a un VPC. Per ulteriori informazioni sugli stati delle funzioni, consultare Stati delle funzioni AWS Lambda.	20 gennaio 2020
Concorrenza fornita	È ora possibile allocare la concorrenza fornita per una versione o un alias di funzione. La concorrenza fornita consente a una funzione di dimensionarsi senza fluttuazioni nella latenza. Per ulteriori dettagli, consultare l'argomento relativo alla gestione della concorrenza di una funzione Lambda.	3 dicembre 2019

Creare un proxy di database	È ora possibile utilizzare la console Lambda per creare un proxy di database per una funzione Lambda. Un proxy di database consente a una funzione di raggiunge re livelli di concorrenza elevati senza esaurire le connessio ni al database. Per informazi oni dettagliate, consultare Configurazione dell'accesso al database per una funzione Lambda.	3 dicembre 2019
Supporto dei percentili per il parametro della durata	È ora possibile filtrare il parametro della durata in base ai percentili. Per ulteriori dettagli, consultare Parametri di AWS Lambda.	26 novembre 2019
Maggiore concorrenza per le origini eventi di flusso	Una nuova opzione per le mappature dell'origine evento del flusso DynamoDB e del flusso Kinesis consente di elaborare più batch alla volta da ogni shard. Quando si aumenta il numero di batch simultanei per shard, la concorrenza della funzione può essere fino a 10 volte il numero degli shard nel flusso. Per informazioni dettagliate, consultare Mappatura dell'orig ine evento di Lambda.	25 novembre 2019

Stati delle funzioni	Quando crei o aggiorni una funzione, questa entra in uno stato di attesa mentre Lambda fornisce le risorse per supportarla. Se connetti la funzione a un VPC, Lambda può creare immediatamente un'interfaccia di rete elastica condivisa, invece di creare interfacce di rete quando viene richiamata la funzione. Ciò si traduce in prestazioni migliori per le funzioni connesse con VPC, ma potrebbe richieder e un aggiornamento dell'auto mazione. Per ulteriori dettagli, consultare Stati delle funzioni di AWS Lambda.	25 novembre 2019
Opzioni di gestione degli errori per la chiamata asincrona	Sono disponibili nuove opzioni di configurazione per la chiamata asincrona. È possibile configurare Lambda in modo da limitare i tentativi e impostare un'età massima dell'evento. Per ulteriori dettagli, consultare l'argomen to relativo alla configurazione della gestione degli errori per l'invocazione asincrona.	25 novembre 2019

Gestione degli errori per le origini eventi di flusso	Sono disponibili nuove opzioni di configurazione per le mappature delle origini eventi che leggono dai flussi. È possibile configurare le mappature dell'origine evento del flusso DynamoDB e del flusso Kinesis per limitare i tentativi e impostare un'età massima dei record. Quando si verificano errori, è possibile configurare la mappatura dell'origine evento per dividere i batch prima di ripetere il tentativo e inviare record di invocazione per i batch in errore a una coda o un argomento. Per informazioni dettagliate, consultare Mappatura dell'origine evento di Lambda.	25 novembre 2019

Destinazioni per la chiamata asincrona	È ora possibile configurare Lambda in modo da inviare record di chiamate asincrone a un altro servizio. I record di invocazione contengono i dettagli sull'evento, il contesto e la risposta della funzione. È possibile inviare i record di chiamata a una coda SQS, a un argomento SNS, a una funzione Lambda o a un bus di eventi. EventBridge Per ulteriori dettagli, consultar e l'argomento relativo alla configurazione delle destinazioni per l'invocazione asincrona .	25 novembre 2019
Nuovi runtime per Node.js, Python e Java	Nuovi runtime sono disponibili per Node.js 12, Python 3.8 e Java 11. Per informazioni dettagliate, consultare Runtime di Lambda.	18 novembre 2019
Supporto per la coda FIFO per le origini eventi Amazon SQS	È ora possibile creare una mappatura di origine eventi che legge da una coda first-in, first-out (FIFO). In precedenza, erano supportate solo le code standard. Per informazioni dettagliate, consultare Uso di Lambda con Amazon SQS.	18 novembre 2019

Creazione di applicazioni nella console Lambda	La creazione di applicazioni nella console Lambda è ora disponibile a livello generale. Per le istruzioni, consulta Gestione delle applicazioni nella console Lambda.	31 ottobre 2019
Creazione di applicazioni nella console Lambda (beta)	Ora è possibile creare un'applicazione Lambda con una pipeline di distribuzione continua integrata nella console Lambda. La console fornisce applicazioni di esempio che è possibile utilizzare come punto di partenza per il proprio progetto. Scegli tra e per il controllo del codice sorgente. AWS CodeCommit GitHub Ogni volta che si inserisco no modifiche al repository, la pipeline inclusa le compila e le distribuisce automaticamente. Per le istruzioni, consulta Gestione delle applicazioni nella console Lambda.	3 ottobre 2019

Miglioramenti delle prestazioni per le funzioni connesse al VPC	Lambda ora utilizza un nuovo tipo di interfaccia di rete elastica che è condivisa da tutte le funzioni in una sottorete VPC (Virtual Private Cloud). Quando si connette una funzione a un VPC, Lambda crea un'interfaccia di rete per ogni combinazione di gruppo di sicurezza e sottorete scelta. Quando le interfacce di rete condivise sono disponibili, la funzione non deve più creare interfacce di rete aggiuntive man mano che aumentano le dimensioni. Questo migliora notevolmente i tempi di avvio. Per ulteriori informazioni, consultare Configurazione di una funzione Lambda per accedere alle risorse in un VPC.	3 settembre 2019

Impostazioni dei batch di flussi	Ora è possibile configura re un periodo di batch per le mappature dell'origine evento Amazon DynamoDB e Amazon Kinesis. Configura un periodo di batch di un massimo di cinque minuti per eseguire il buffer dei record in entrata fino a quando non diventi disponibile il batch completo. In tal modo si riduce il numero di volte in cui la funzione viene invocata quando il flusso è meno attivo.	29 agosto 2019
CloudWatch Integrazione con Logs Insights	La pagina di monitoraggio nella console Lambda ora include i report di Amazon CloudWatch Logs Insights.	18 giugno 2019
Amazon Linux 2018.03	L'ambiente di esecuzione Lambda è in fase di aggiornam ento per consentire l'uso di Amazon Linux 2018.03. Per ulteriori dettagli, consultare Ambiente di esecuzione.	21 maggio 2019

Node.js 10	Per Node.js 10 è disponibile il nuovo runtime nodejs10.x. Questo runtime usa Node.js 10.15 e viene periodicamente aggiornato con le ultime versioni di Node.js 10. Node.js 10 è anche il primo runtime per utilizzare Amazon Linux 2. Per ulteriori dettagli, consultare Compilazione di funzioni Lambda con Node.js.	13 maggio 2019
GetLayerVersionByArn API	Utilizzate l'GetLayerVersionByArnAPI per scaricare le informazioni sulla versione del layer con la versione ARN come input. Rispetto a GetLayerVersion, GetLayerVersionByArn consente di utilizzare l'ARN direttamente anziché analizzarlo per ottenere il nome del livello e il numero di versione.	25 aprile 2019
Ruby	AWS Lambda ora supporta Ruby 2.5 con un nuovo runtime. Per informazioni dettagliate, consultare Compilazione di funzioni Lambda con Ruby.	29 novembre 2018

Livelli	Con i livelli Lambda, è possibile impacchettare e distribuire librerie, runtime personalizzati e altre dipendenze separatamente dal codice della funzione. Condividere i propri livelli con gli altri account o in tutto il mondo. Per informazioni dettagliate, consultare Livelli di Lambda.	29 novembre 2018
Runtime personalizzati	Costruire un runtime personalizzato per eseguire le funzioni Lambda nel proprio linguaggio di programmazione preferito. Per informazioni dettagliate, consultare Runtime Lambda personalizzati.	29 novembre 2018
Trigger di Application Load Balancer	Elastic Load Balancing ora supporta le funzioni Lambda come target per gli Application Load Balancer. Per informazioni dettagliate, consultare Uso di Lambda con i sistemi di bilanciamento del carico delle applicazioni.	29 novembre 2018

Utilizzo dei consumer del flusso Kinesis HTTP/2 come trigger	È possibile utilizzare i consumatori del flusso Kinesis HTTP/2 per l'invio di eventi a AWS Lambda. I consumatori del flusso hanno throughput in lettura dedicato da ogni shard nel flusso di dati e utilizzano HTTP/2 per ridurre al minimo la latenza. Per informazioni dettagliate, consultare Uso di Lambda con Kinesis.	19 novembre 2018
Python 3.7	AWS Lambda ora supporta Python 3.7 con un nuovo runtime. Per ulteriori informazioni, consultare Compilazione di funzioni Lambda con Python.	19 novembre 2018
Aumento del limite di payload per il richiamo della funzione asincrona	La dimensione massima del payload per le invocazioni asincrone è aumentata da 128 KB a 256 KB, che corrisponde alla dimensione massima del messaggio da un trigger Amazon SNS. Per informazioni dettagliate, consultare Quote di Lambda.	16 novembre 2018
AWS GovCloud Regione (Stati Uniti orientali)	AWS Lambda è ora disponibile nella regione AWS GovCloud (Stati Uniti orientali).	12 novembre 2018

AWS SAM Gli argomenti sono stati spostati in una guida per sviluppatori separata	Alcuni argomenti si sono concentrati sulla creazione di applicazioni serverless utilizzando AWS Serverless Application Model (AWS SAM). Questi argomenti sono stati spostati nella Guida per gli sviluppatori di AWS Serverless Application Model.	25 ottobre 2018
Visualizzazione delle applicazioni Lambda nella console	È possibile visualizzare lo stato delle applicazioni Lambda nella pagina Applications (Applicazioni) della console Lambda. Questa pagina mostra lo stato dello CloudFormation stack. Include collegamenti a pagine con ulteriori informazioni sulle risorse dello stack. È possibile anche visualizzare i parametri di aggregazione per l'applicazione e creare pannelli di controllo di monitoraggio personalizzati.	11 ottobre 2018
Limite di timeout dell'esecuzione della funzione	Per consentire funzioni di lunga durata, il timeout di esecuzione massimo configurabile è aumentato da 5 minuti a 15 minuti. Per informazioni dettagliate, consultare Limiti di Lambda.	10 ottobre 2018

Support per il linguaggio PowerShell Core in AWS Lambda	AWS Lambda ora supporta il linguaggio PowerShell Core. Per ulteriori informazioni, consulta Modello di programmazione per la creazione di funzioni Lambda in. PowerShell	11 settembre 2018
Support per il runtime di.NET Core 2.1.0 in AWS Lambda	AWS Lambda ora supporta il runtime di.NET Core 2.1.0. Per ulteriori informazioni, consultare Interfaccia a riga di comando (CLI) di .NET Core.	9 luglio 2018
Aggiornamenti ora disponibili tramite RSS	Ora è possibile iscriverti a un feed RSS per seguire la pubblicazione di nuove versioni di questa guida.	5 luglio 2018
Supporto per Amazon SQS come origine evento	AWS Lambda ora supporta Amazon Simple Queue Service (Amazon SQS) come fonte di eventi. Per ulteriori informazioni, consultare Richiamo delle funzioni Lambda.	28 giugno 2018
Regione Cina (Ningxia)	AWS Lambda è ora disponibile nella regione Cina (Ningxia). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina Regioni ed endpoint nella Riferimenti generali di AWS.	28 giugno 2018

Aggiornamenti precedenti

La tabella seguente descrive le modifiche importanti apportate in ogni versione della Guida per gli sviluppatori AWS Lambda prima di giugno 2018.

Modifica	Descrizione	Data
Supporto runtime per il runtime del Node.js 8.10	AWS Lambda ora supporta la versione 8.10 del runtime di Node.js. Per ulteriori informazioni, consulta Compilazione di funzioni Lambda con Node.js.	2 aprile 2018
Revisione di funzioni e alias IDs	AWS Lambda ora supporta la revisione delle IDs versioni e degli alias delle funzioni. È possibile utilizzarli IDs per tracciare e applicare aggiornamenti condizionali quando si aggiorna la versione della funzione o le risorse alias.	25 gennaio 2018
Supporto del runtime per Go e .NET 2.0	AWS Lambda ha aggiunto il supporto di runtime per Go e.NET 2.0. Per ulteriori informazioni, consultare Compilazione di funzioni Lambda con Go e Compilazione di funzioni Lambda con C#.	15 gennaio 2018
Nuovo progetto della console	AWS Lambda ha introdotto una nuova console Lambda per semplificare l'esperienza e ha aggiunto un editor di codice Cloud9 per migliorare le tue capacità di debug e rivedere il codice della funzione.	30 novembre 2017
Impostazione dei limiti di concorrenza per singole funzioni	AWS Lambda ora supporta l'impostazione di limiti di concorrenza per le singole funzioni. Per ulteriori informazioni, consulta Configurazione della simultaneità riservata per una funzione.	30 novembre 2017
Trasferimento del traffico con gli alias	AWS Lambda ora supporta lo spostamento del traffico con alias. Per ulteriori informazioni, consulta Creare una implementazione continua con alias ponderati.	28 novembre 2017
Distribuzione graduale del codice	AWS Lambda ora supporta l'implementazione sicura di nuove versioni della funzione Lambda sfruttando Code	28 novembre 2017

Modifica	Descrizione	Data
	Deploy. Per ulteriori informazioni, consultare Distribuzione graduale del codice.	
Regione Cina (Pechino)	AWS Lambda è ora disponibile nella regione Cina (Pechino). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina Regioni ed endpoint nella Riferimenti generali di AWS.	9 novembre 2017
Introduzione a SAM Local	AWS Lambda introduce SAM Local (ora noto come SAM CLI), AWS CLI uno strumento che fornisce un ambiente per sviluppare, testare e analizzare le applicazioni serverless a livello locale prima di caricarle nel runtime Lambda. Per ulteriori informazioni, consultare Esecuzione di test e debug di applicazioni serverless.	11 agosto 2017
Regione Canada (Centrale)	AWS Lambda è ora disponibile nella regione Canada (Centrale). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina Regioni ed endpoint nella Riferimenti generali di AWS.	22 giugno 2017
South America (São Paulo) Region	AWS Lambda è ora disponibile nella regione Sud America (San Paolo). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina Regioni ed endpoint nella Riferimenti generali di AWS.	6 giugno 2017
AWS Lambda supporto per AWS X-Ray.	Lambda introduce il supporto per X-Ray, che consente di rilevare, analizzare e ottimizzare i problemi di prestazio ni delle applicazioni Lambda. Per ulteriori informazioni, consultare Visualizza le chiamate alla funzione Lambda utilizzando AWS X-Ray.	19 aprile 2017
Asia Pacific (Mumbai) Region	AWS Lambda è ora disponibile nella regione Asia Pacifico (Mumbai). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina Regioni ed endpoint nella Riferimenti generali di AWS.	28 marzo 2017

Modifica	Descrizione	Data
AWS Lambda ora supporta il runtime di Node.js v6.10	AWS Lambda ha aggiunto il supporto per Node.js runtime v6.10. Per ulteriori informazioni, consulta Compilazione di funzioni Lambda con Node.js.	22 marzo 2017
Europe (London) Region	AWS Lambda è ora disponibile nella regione Europa (Londra). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina Regioni ed endpoint nella Riferimenti generali di AWS.	1 febbraio 2017
AWS Lambda supporto per il runtime di.NET, Lambda @Edge (Preview), Dead Letter Queues e distribuzione automatizzata di applicazioni serverless.	AWS Lambda ha aggiunto il supporto per C#. Per ulteriori informazioni, consulta Compilazione di funzioni Lambda con C#. Lambda @Edge consente di eseguire funzioni Lambda nelle posizioni AWS Edge in risposta agli eventi. CloudFront Per ulteriori informazioni, consulta Personalizzazione sul perimetro con Lambda @Edge.	3 dicembre 2016
AWS Lambda aggiunge Amazon Lex come fonte di eventi supportata.	Utilizzando Lambda e Amazon Lex, è possibile costruire rapidamente chat bot per vari servizi come Slack e Facebook.	30 novembre 2016
US West (N. California) Region	AWS Lambda è ora disponibile nella regione Stati Uniti occidentali (California settentrionale). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina Regioni ed endpoint nella Riferimenti generali di AWS.	21 novembre 2016

Modifica	Descrizione	Data
È stato introdotto AWS SAM per la creazione e la distribuzione di applicazioni basate su Lambda e l'utilizzo di variabili di ambiente per le impostazioni di configurazione delle funzioni Lambda.	AWS SAM: È ora possibile utilizzare il AWS SAM per definire la sintassi per esprimere le risorse all'interno di un'applicazione serverless. Per distribuire l'applicazione, è sufficiente specificare le risorse necessarie come parte dell'applicazione, insieme alle policy di autorizzazione associate in un file di modello AWS CloudFormation , scritto in JSON o YAML, creare un pacchetto degli artefatti di distribuzione e distribuire il modello. Variabili di ambiente: si possono utilizzare le variabili di ambiente per specificare le impostazioni di configurazione per la funzione Lambda al di fuori del codice della funzione. Per ulteriori informazioni, consultare Utilizzo delle variabili di ambiente Lambda.	18 novembre 2016
Regione Asia Pacifico (Seoul)	AWS Lambda è ora disponibile nella regione Asia Pacifico (Seoul). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina Regioni ed endpoint nella Riferimenti generali di AWS.	29 agosto 2016
Asia Pacific (Sydney) Region	Lambda è ora disponibile nella regione Asia Pacifico (Sydney). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina Regioni ed endpoint nella Riferimenti generali di AWS.	23 giugno 2016
Aggiornamenti alla console Lambda	La console Lambda è stata aggiornata per semplificare il processo di creazione dei ruoli.	23 giugno 2016
AWS Lambda ora supporta Node.js runtime v4.3	AWS Lambda ha aggiunto il supporto per Node.js runtime v4.3. Per ulteriori informazioni, consulta Compilazione di funzioni Lambda con Node.js.	07 aprile 2016
Regione Europa (Francoforte)	Lambda è ora disponibile nella regione Europa (Francofo rte). Per ulteriori informazioni sulle regioni e sugli endpoint Lambda, consulta la pagina Regioni ed endpoint nella Riferimenti generali di AWS.	14 marzo 2016

Modifica	Descrizione	Data
Supporto per VPC	Ora è possibile configurare una funzione Lambda per accedere alle risorse del proprio VPC. Per ulteriori informazioni, consultare Consentire alle funzioni Lambda l'accesso alle risorse in un Amazon VPC.	11 febbraio 2016
Il runtime Lambda è stato aggiornato.	L'ambiente di esecuzione è stato aggiornato.	4 novembre 2015
Supporto per il controllo delle versioni, Python per lo sviluppo di codice per le funzioni Lambda, eventi pianificati e aumento del runtime	Ora è possibile sviluppare il codice della funzione Lambda tramite Python. Per ulteriori informazioni, consultare Compilazione di funzioni Lambda con Python. Funzione Versioni multiple: è possibile gestire una o più versioni della funzione Lambda. La funzione Versioni multiple consente di controllare quale versione della funzione Lambda viene eseguita in ambienti diversi (ad esempio ambienti di sviluppo, test o produzione). Per ulteriori informazioni, consultare Gestire le versioni delle funzioni Lambda. Eventi pianificati: si può anche configurare Lambda in modo da richiamare il proprio codice a intervalli regolari e pianificati utilizzando la console Lambda. È possibile specificare una frequenza fissa (numero di ore, giorni o settimane) oppure un'espressione Cron. Per ulteriori informazioni, consulta Richiamo di una funzione Lambda su una pianificazione. Aumento del runtime: è ora possibile configurare le funzioni Lambda per essere eseguite per un massimo di cinque minuti consentendo tempi di esecuzione superiori per alcune funzioni, ad esempio caricamento di grandi volumi di dati e processi di elaborazione.	08 ottobre 2015

Modifica	Descrizione	Data
Supporto per DynamoDB Streams	DynamoDB Streams è ora disponibile a livello generale e può essere utilizzato in tutte le regioni in cui è disponibi le DynamoDB. È possibile utilizzare DynamoDB Streams per la tabella in uso e utilizzare una funzione Lambda come trigger per tale tabella. I trigger sono operazioni personalizzate che si effettuano in risposta ad aggiornam enti eseguiti sulla tabella DynamoDB. Per un esempio di procedura guidata, consulta la pagina Tutorial: Utilizzo AWS Lambda con flussi Amazon DynamoDB.	14 luglio 2015
Lambda ora supporta l'invocazione di funzioni Lambda con client compatibili REST.	Fino ad ora, per richiamare la funzione Lambda dall'appl icazione Web, mobile o IoT era necessario AWS SDKs (ad esempio AWS , SDK per Java, AWS SDK per Android o AWS SDK per iOS). Ora Lambda supporta l'invocazione di una funzione Lambda con client compatibili REST attravers o un'API personalizzata che è possibile creare utilizzando Gateway Amazon API. È possibile inviare richieste all'URL dell'endpoint della funzione Lambda. È possibile configura re la sicurezza sull'endpoint per consentire l'accesso aperto, sfruttare AWS Identity and Access Management (IAM) per autorizzare l'accesso o utilizzare le chiavi API per misurare l'accesso alle proprie funzioni Lambda da parte di altri. Per un esercizio di Nozioni di base di esempio, consultar e Richiamo di funzione Lambda utilizzando un endpoint Gateway Amazon API.	09 luglio 2015
La console Lambda ora fornisce piani con cui creare facilment e funzioni Lambda e testarle.	La console Lambda fornisce un set di piani. Ciascuno di essi fornisce una configurazione di origini eventi e un codice di esempio per la funzione Lambda che è possibile utilizzare per creare facilmente applicazioni basate su Lambda. Tutti gli esercizi delle Nozioni di base di Lambda ora utilizzano i piani.	09 luglio 2015

Modifica	Descrizione	Data
Lambda ora supporta Java per la creazione delle funzioni Lambda.	Ora è possibile creare codice Lambda in Java. Per ulteriori informazioni, consultare Compilazione di funzioni Lambda con Java.	15 giugno 2015
Lambda ora supporta la specifica di un oggetto Amazon S3 come funzione .zip durante la creazione o l'aggiornamento di una funzione Lambda.	È possibile caricare un pacchetto di distribuzione della funzione Lambda (file .zip) in un bucket Amazon S3 nella stessa regione in cui si desidera creare una funzione Lambda, quindi specificare il nome del bucket e della chiave dell'oggetto quando si crea o si aggiorna una funzione Lambda.	28 maggio 2015
Disponibilità generale di Lambda con supporto aggiunto per back-end per dispositi vi mobili	Lambda è ora disponibile a livello generale per l'utilizzo in ambiente di produzione. Questa versione introduce inoltre nuove funzionalità che semplificano la creazione mediante Lambda di back-end per dispositivi mobili, tablet e Internet of Things (IoT) che scalano automaticamente senza provisioning o gestione dell'infrastruttura. Lambda ora supporta sia eventi in tempo reale (sincroni) che asincroni . Funzionalità aggiuntive includono una configurazione e una gestione più semplici delle origini eventi. Il modello di autorizzazione e di programmazione sono stati semplificati grazie all'introduzione di policy per le risorse per le funzioni Lambda.	9 aprile 2015
Versione di anteprima	Versione di anteprima della Guida per gli sviluppatori di AWS Lambda .	13 novembre 2014

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.