
AWS Lambda

Guia do desenvolvedor



AWS Lambda: Guia do desenvolvedor

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

As marcas comerciais e imagens de marcas da Amazon não podem ser usadas no contexto de nenhum produto ou serviço que não seja da Amazon, nem de qualquer maneira que possa gerar confusão entre os clientes ou que deprecie ou desprestigue a Amazon. Todas as outras marcas comerciais que não pertencem à Amazon pertencem a seus respectivos proprietários, que podem ou não ser afiliados, patrocinados pela Amazon ou ter conexão com ela.

Table of Contents

O que é o AWS Lambda?	1
Quando devo usar o Lambda?	1
Recursos do Lambda	2
Conceitos básicos do Lambda	3
Serviços relacionados	3
Acesso ao Lambda	4
Definição de preço do Lambda	4
Configuração	5
AWSConta da	5
AWS CLI	5
AWS SAM	5
AWS SAM CLI	6
Ferramentas para imagens de contêiner	6
Ferramentas de criação de código	6
Conceitos básicos	8
Criar uma função do	9
Criar a função	9
Invocar a função do Lambda	9
Limpar	10
Criar uma função do definida como uma imagem de contêiner	12
Prerequisites	12
Criar a imagem de contêiner	12
Fazer upload da imagem para o repositório do Amazon ECR	13
Atualizar as permissões do usuário	14
Criar uma função do Lambda definida como uma imagem de contêiner	14
Invocar a função do Lambda	14
Limpar	15
Fundamentos do Lambda	17
Conceitos	18
Function	18
Trigger	18
Event	18
Ambiente de execução	19
Pacote de implantação	19
Runtime	19
Layer	19
Extension	20
Concurrency	20
Qualifier	20
Destination	21
Recursos	22
Scaling	22
Controles de simultaneidade	23
Invocação assíncrona	25
Mapeamentos de origem do evento	26
Destinations	27
Esquemas de funções	28
Ferramentas de teste e implantação	29
Modelos de aplicativos	29
Modelo de programação	30
Escalabilidade de função	32
Pacotes de implantação	37
Imagens de contêiner	37
Arquivos .zip	37

Layers	38
Usando outros serviços do AWS	38
Console do Lambda	40
Applications	40
Functions	40
Assinatura de código	40
Layers	40
Editar código usando o editor do console do	40
CLI do Lambda	47
Prerequisites	47
Criar a função de execução	47
Criar a função	48
Listar as funções do Lambda na conta	51
Recuperar uma função do Lambda	51
Limpar	52
Cotas do Lambda	53
Computação e armazenamento	53
Configuração, implantação e execução de funções	53
Solicitações da API do Lambda	54
Outros serviços	55
Permissões	56
Função de execução	57
Criar uma função de execução no console do IAM	58
Conceda acesso de menor privilégio à sua função de execução do Lambda	58
Gerenciar funções com a API do IAM	59
Políticas gerenciadas da AWS para recursos do Lambda	60
Políticas baseadas em recursos	62
Conceder acesso de função aos serviços da AWS	63
Conceder acesso de função a outras contas	64
Conceder acesso de camada a outras contas	65
Limpar políticas baseadas em recursos	66
Políticas de usuário	67
Desenvolvimento da função	67
Desenvolvimento e uso da camada	70
Funções entre contas	71
Chaves de condição para configurações de VPC	71
Recursos e condições	72
Nomes de recursos de função	73
Ações de função	75
Ações de mapeamento da fonte de eventos	77
Ações de camada	77
Limites de permissões	79
Configurar funções do	81
Criação de funções (pacotes.zip)	82
Criar uma função (console)	82
Usando o editor de código do console	83
Atualizar o código da função (console)	83
Definindo configurações de execução do (console)	83
Usar a API do Lambda	84
AWS CloudFormation	84
Criar camadas	85
Criar conteúdo de camada	85
Compilar o arquivo .zip para sua camada	85
Incluir dependências da biblioteca em uma camada	85
Instruções específicas de linguagem	87
Criar uma camada	87
Excluir uma versão de camada	88

Configurar permissões de camada	88
Usar o AWS CloudFormation com camadas	89
Criar funções (imagens do contêiner)	90
Versão da função \$LATEST	90
Implantação de imagens de contêiner	91
Permissões do Amazon ECR	91
Substituir as configurações de contêiner	91
Criar uma função (console)	91
Atualizar o código da função (console)	92
Substituir os parâmetros da imagem (console)	93
Usar a API do Lambda	93
AWS CloudFormation	94
Configurar opções da função	95
Versões da função	95
Usar a visão geral da função	95
Defina as funções (console)	96
Configurar funções (API)	84
Configurar a memória da função (console)	97
Aceitar recomendações de memória de função (console)	97
Configurando gatilhos (console)	98
Funções de teste (console)	98
Variáveis de ambiente	99
Configurar variáveis de ambiente	99
Como configurar variáveis de ambiente com a API	100
Exemplo de cenário para variáveis de ambiente	100
Recupere variáveis de ambiente	101
Variáveis de ambiente com tempo de execução definido	102
Proteger variáveis de ambiente	103
Modelos e código de exemplo	105
Versões	106
Como criar versões de função	106
Gerenciar versões com a API do Lambda	106
Usar versões	107
Conceder permissões	107
Aliases	108
Como criar um alias da função (console)	108
Gerenciar aliases com a API do Lambda	108
Usar aliases	109
Políticas de recursos	109
Configuração de roteamento de alias	109
Gerenciar funções	112
Simultaneidade	113
Configurar a simultaneidade reservada	114
Configurar a simultaneidade provisionada	116
Configurar a simultaneidade com a API do Lambda	119
Rede	124
Função de execução e permissões de usuário	124
Como configurar o acesso à VPC (console)	125
Como configurar o acesso à VPC (API)	126
Usar chaves de condição do IAM para configurações de VPC	126
Acesso aos serviços e à Internet para funções conectadas à VPC	129
Tutoriais de VPC	130
Configurações de VPC de exemplo	130
VPC endpoints de interface	131
Considerações para endpoints de interface do Lambda	131
Criar um endpoint de interface para o Lambda	132
Criar uma política de endpoint de interface para o Lambda	133

Banco de dados	134
Como criar um proxy de banco de dados (console)	134
Usar as permissões da função para autenticação	135
Aplicativo de amostra	135
Sistema de arquivos	139
Como conectar-se a um sistema de arquivos (console)	139
Configurar um sistema de arquivos e ponto de acesso	140
Função de execução e permissões de usuário	140
Configurar o acesso ao sistema de arquivos com a API do Lambda	141
AWS CloudFormation e AWS SAM	142
Aplicativos de exemplo	143
Assinatura de código	144
Validação de assinatura	145
Pré-requisitos de configuração	145
Criar configurações de assinatura de código	145
Atualizar uma configuração de assinatura de código	146
Excluir uma configuração de assinatura de código	146
Habilitar a assinatura de código para uma função	146
Configurar políticas do IAM	147
Configurar assinatura de código com a API do Lambda	148
Tags	149
Como adicionar tags a uma função (console)	149
Como usar tags para filtrar funções (console)	149
Como usar tags com a AWS CLI	150
Requisitos de chave de tag e valor	151
Usar camadas	152
Configurar uma função para usar camadas	152
Acessar o conteúdo de uma camada	153
Localizar informações da camada	153
Atualizar uma versão de camada utilizada por sua função	154
Adicionar permissões de camada	155
Usar o AWS SAM para adicionar uma camada a uma função	89
Aplicativos de exemplo	155
Chamada de funções do	157
Invocação síncrona	158
Invocação assíncrona	161
Como o Lambda trabalha com invocações assíncronas	161
Configurar o tratamento de erros para invocação assíncrona	163
Configurar destinos para invocação assíncrona	164
API de configuração de invocação assíncrona	166
Filas de mensagens mortas	167
Mapeamento de origem do evento	170
Estados de função	174
Tratamento de erros	176
Usar extensões do	178
Ambiente de execução	179
Impacto na performance e nos recursos	179
Permissions	179
Configurando extensões (arquivamento de arquivo.zip)	180
Uso de extensões em imagens de contêiner	180
Próximas etapas	181
Invoker funções definidas como imagens de contêiner	182
Ciclo de vida da função	182
Invoker a função	182
Segurança de imagem	182
Mobile SDK for Android	183
Tutorial	183

Código de exemplo	189
Aplicações do Lambda	192
Gerenciar aplicativos	193
Monitorar aplicativos	193
Painéis de monitoramento personalizados	193
Tutorial: Criar uma aplicação	196
Prerequisites	197
Criar uma aplicação	197
Invocar a função do	198
Adicionar um recurso da AWS	199
Atualizar o limite de permissões	201
Atualizar o código da função	201
Próximas etapas	203
Troubleshooting	203
Limpar	204
Implantações contínuas	206
ExemploAWS SAMModelo Lambda	206
Casos de uso	208
Exemplo 1: o Amazon S3 envia eventos por push e invoca uma função do Lambda	208
Exemplo 2: o AWS Lambda extrai eventos de uma transmissão do Kinesis e invoca uma função do Lambda	209
Práticas recomendadas	211
Código da função	211
Configuração da função	212
Métricas e alarmes	213
Trabalhar com streams	213
Tempos de execução do Lambda	214
Política de suporte ao tempo de execução	217
Ambiente de execução	219
Ciclo de vida do tempo de execução	219
Imagens de contêiner	222
AWSImagens base para o Lambda	222
Imagens de base para tempos de execução personalizados	222
Clientes de interface de tempo de execução	223
Emulador de interface de tempo de execução	223
API de tempo de execução	224
Próxima invocação	224
Resposta de invocação	225
Erro de inicialização	225
Erro de invocação	226
API de extensões	228
Ciclo de vida do ambiente de execução do Lambda	229
Referência de API de extensões	237
API de registros	242
Assinando para receber registros	243
Uso de memória	243
protocolos de destino	243
Configuração de buffer	243
Exemplo de assinatura	244
Código de exemplo para Logs API	244
Referência da API Logs	245
Mensagens de log	246
Modificações do tempo de execução	250
Variáveis de ambiente específicas de linguagem	250
Scripts wrapper	252
Tempos de execução personalizados	255
Usar um tempo de execução personalizado	255

Criar um tempo de execução personalizado	255
Tutorial: tempo de execução personalizado	258
Prerequisites	258
Criar uma função do	258
Criar uma camada	260
Atualizar a função	261
Atualizar o tempo de execução	262
Compartilhar a camada	262
Limpar	263
Vetorização AVX2	264
Compilar a partir da origem	264
Ativar a AVX2 para Intel MKL	264
Compatibilidade com AVX2 em outras linguagens	264
Imagens de contêiner	266
Criar imagens	267
Tipos de imagem	267
Ferramentas de contêiner	267
Requisitos do Lambda para imagens de contêiner	268
Configurações de imagem de contêiner	268
Criar uma imagem a partir de uma imagem básica da AWS para o Lambda	268
Criar uma imagem a partir de uma imagem básica alternativa	271
Criar uma imagem usando o toolkit do AWS SAM	272
Testar imagens	274
Diretrizes para o uso do RIE	274
Variáveis de ambiente	274
Testar uma imagem com o RIE incluído nela	275
Criar o RIE na sua imagem de base	275
Testar uma imagem sem adicionar RIE a ela	276
Trabalhar com outros serviços	277
Alexa	280
API Gateway	281
Permissions	283
Tratamento de erros com uma API do API Gateway	284
Escolher um tipo de API	285
Aplicativos de exemplo	287
Tutorial	287
Código de exemplo	298
Esquema de microsserviço	300
Modelo de exemplo	302
CloudTrail	303
Logs do CloudTrail	305
Tutorial	308
Código de exemplo	313
CloudWatch Events	315
Tutorial	316
Modelo de exemplo	319
Programar expressões	320
CloudWatch Logs	322
CloudFormation	323
CloudFront (Lambda@Edge)	325
CodeCommit	327
CodePipeline	328
Permissions	329
Cognito	330
Config	331
Conecte-se	332
DynamoDB	333

Permissões da função de execução	335
Configurar um stream como fonte de eventos	335
APIs de mapeamento da fonte de eventos	336
Tratamento de erros	338
Métricas do Amazon CloudWatch	339
Janelas de tempo	339
Gerar relatórios de falhas de itens de lote	343
Tutorial	345
Código de exemplo	350
Modelo de exemplo	353
EC2	355
Permissions	355
Tutorial: instâncias spot	356
ElastiCache	366
Prerequisites	366
Criar a função de execução	366
Criar um cluster do ElastiCache	367
Criar um pacote de implantação	367
Criar a função do Lambda	368
Testar a função do Lambda	368
Limpar recursos	297
Elastic Load Balancing	370
EFS	372
Connections	372
Throughput	373
IOPS	373
IoT	374
IoT Events	375
Apache Kafka	377
Gerenciar acesso e permissões	378
Autenticação Kafka	379
Configuração de rede	380
Adicionar um cluster do Kafka como uma origem de evento	380
Usar um cluster do Kafka como uma fonte de eventos	382
Dimensionamento automático da origem do evento Kafka	383
Operações de API de origem de eventos	383
Erros de origem do evento	383
Parâmetros de configuração da fonte do	384
Kinesis Firehose	386
Kinesis Streams	387
Configurar o stream de dados e a função	389
Permissões da função de execução	389
Configurar um stream como fonte de eventos	390
API do mapeamento da fonte de eventos	391
Tratamento de erros	393
Métricas do Amazon CloudWatch	394
Janelas de tempo	394
Gerar relatórios de falhas de itens de lote	397
Tutorial	399
Código de exemplo	403
Modelo de exemplo	406
Lex	408
Funções e permissões	409
MQ	411
Grupo de consumidores do Lambda	412
Permissões da função de execução	414
Configurar um agente como uma fonte de eventos	414

API do mapeamento da fonte de eventos	415
Erros de mapeamento da fonte de eventos	417
MSK	419
Gerenciar acesso e permissões	420
Configuração de rede	380
Adicionar o Amazon MSK como uma origem de evento	421
Auto scaling da origem do evento do Amazon MSK	423
Parâmetros de configuração do Amazon MSK	423
RDS	425
Tutorial	425
Configurar a função	429
S3	431
Tutorial: Use um trigger do S3	432
Tutorial: Use um trigger do S3 para criar miniaturas	437
Modelo de exemplo do SAM	450
Lote do S3	452
Invocar funções do Lambda de operações em lote do Amazon S3	453
Secrets Manager	454
SES	455
SNS	457
Tutorial	458
Código de exemplo	462
SQS	465
Escalabilidade e processamento	466
Configurar uma fila para usar com o Lambda	467
Permissões da função de execução	467
Configurar uma fila como uma fonte de eventos	467
APIs de mapeamento da fonte de eventos	336
Tutorial	469
Código de exemplo	473
Modelo de exemplo	476
X-Ray	477
Permissões da função de execução	479
O daemon do AWS X-Ray	479
Habilitar o rastreamento ativo com a API do Lambda	479
Habilitar o rastreamento ativo com o AWS CloudFormation	480
Orquestrar funções	481
Padrões de aplicação	481
Componentes das máquinas de estado	481
Padrões de aplicação de máquinas de estado	481
Aplicação de padrões em máquinas de estado	482
Exemplo de padrão de aplicação de ramificação	482
Gerenciar máquinas de estado	485
Visualizar detalhes da máquina de estado	485
Editar uma máquina de estado	485
Executar uma máquina de estado	486
Exemplos de orquestração	486
Configuração de uma função do Lambda como tarefa.	486
Configurar uma máquina de estado como uma origem de evento	487
Manipular erros de função e de serviço	488
AWS CloudFormation e AWS SAM	488
Amostras	491
Função em branco	493
Arquitetura e código do manipulador	494
Automação de implantação com o AWS CloudFormation e a AWS CLI	495
Instrumentação com o AWS X-Ray	497
Gerenciamento de dependências com camadas	498

Processador de erros	500	
Estrutura de eventos e arquitetura	501	
Instrumentação com o AWS X-Ray	502	
AWS CloudFormation	Modelo do e recursos adicionais	503
Gerenciador de listas	505	
Estrutura de eventos e arquitetura	506	
Instrumentação com o AWS X-Ray	507	
AWS CloudFormation	Modelos do e recursos adicionais	510
Trabalho com Node.js	511	
Manipulador	514	
Manipuladores assíncronos	514	
Manipuladores não assíncronos	515	
Implantar arquivos .zip	517	
Prerequisites	517	
Atualizar uma função sem dependências	517	
Atualizar uma função com dependências adicionais	518	
Implantar imagens de contêiner	520	
AWSImagens de base da para Node.js	520	
Usar uma imagem base Node.js	520	
Clientes de interface de tempo de execução do Node.js	521	
Implantar a imagem do contêiner	521	
Contexto	522	
Registro em log	524	
Criar uma função que retorna logs	524	
Usar o console do Lambda	525	
Usando o console do CloudWatch	525	
Usar a AWS Command Line Interface (AWS CLI)	526	
Excluir logs	528	
Erros	529	
Syntax	529	
Como funcionam	529	
Usar o console do Lambda	530	
Usar a AWS Command Line Interface (AWS CLI)	530	
Tratamento de erros em outros serviços da AWS	531	
Próximas etapas	532	
Rastreamento	533	
Habilitar o rastreamento ativo com a API do Lambda	536	
Habilitar o rastreamento ativo com o AWS CloudFormation	536	
Armazenar dependências de tempo de execução em uma camada	536	
Como trabalhar com Python	538	
Manipulador	540	
Naming	540	
Como funcionam	540	
Retornar um valor	541	
Examples	541	
Implantar arquivos .zip	543	
Prerequisites	544	
O que é uma dependência de runtime?	544	
Pacote de implantação sem dependências	544	
Pacote de implantação com dependências	545	
Usar um ambiente virtual	546	
Adicionar o arquivo .zip à função	546	
Implantar imagens de contêiner	548	
AWS imagens base para Python	548	
Clientes de interface de tempo de execução Python	549	
Criar uma imagem Python a partir de uma imagem básica da AWS	549	
Criar uma imagem Python a partir de uma imagem base alternativa	550	

Implantar a imagem do contêiner	550
Contexto	551
Registro em log	553
Criar uma função que retorna logs	553
Usar o console do Lambda	554
Usando o console do CloudWatch	554
Usar a AWS Command Line Interface (AWS CLI)	554
Excluir logs	557
Biblioteca de registros em log	557
Erros	558
Como funcionam	558
Usar o console do Lambda	559
Usar a AWS Command Line Interface (AWS CLI)	559
Tratamento de erros em outros serviços da AWS	560
Exemplos de erro	560
Aplicativos de exemplo	561
Próximas etapas	532
Rastreamento	562
Habilitar o rastreamento ativo com a API do Lambda	564
Habilitar o rastreamento ativo com o AWS CloudFormation	565
Armazenar dependências de tempo de execução em uma camada	565
Trabalhar com Ruby	567
Manipulador	570
Implantar arquivos .zip	571
Prerequisites	571
Ferramentas e bibliotecas	571
Atualizar uma função sem dependências	572
Atualizar uma função com dependências adicionais	572
Implantar imagens de contêiner	574
AWSImagens de base da para Ruby	574
Usar uma imagem base Ruby	574
Clientes de interface de tempo de execução do Ruby	575
Criar uma imagem do Ruby a partir de uma imagem básica da AWS	575
Implantar a imagem do contêiner	576
Contexto	577
Registro em log	578
Criar uma função que retorna logs	578
Usar o console do Lambda	579
Usando o console do CloudWatch	579
Usar a AWS Command Line Interface (AWS CLI)	580
Excluir logs	582
Erros	583
Sintaxe	583
Como funcionam	583
Usar o console do Lambda	584
Usar a AWS Command Line Interface (AWS CLI)	584
Tratamento de erros em outros serviços da AWS	585
Aplicativos de exemplo	586
Próximas etapas	586
Rastreamento	587
Habilitar o rastreamento ativo com a API do Lambda	589
Habilitar o rastreamento ativo com o AWS CloudFormation	590
Armazenar dependências de tempo de execução em uma camada	590
Como trabalhar com Java	592
Manipulador	595
Escolher tipos de entrada e saída	596
Interfaces do manipulador	597

Código de exemplo do manipulador	598
Implantar arquivos .zip	599
Prerequisites	599
Ferramentas e bibliotecas	599
Implantar imagens de contêiner	606
AWSImagens de base da para Java	606
Usar uma imagem base Java	607
Clientes de interface de tempo de execução para Java	607
Implantar a imagem do contêiner	607
Contexto	608
Contexto em aplicativos de exemplo	609
Registro em log	611
Criar uma função que retorna logs	611
Usar o console do Lambda	612
Usando o console do CloudWatch	612
Usar a AWS Command Line Interface (AWS CLI)	613
Excluir logs	615
Registro em log avançado com Log4j 2 e SLF4J	615
Código de exemplo de registro em log	617
Erros	618
Syntax	618
Como funcionam	619
Criar uma função que retorna exceções	619
Usar o console do Lambda	620
Usar a AWS Command Line Interface (AWS CLI)	621
Tratamento de erros em outros serviços da AWS	621
Aplicativos de exemplo	622
Próximas etapas	622
Rastreamento	623
Habilitar o rastreamento ativo com a API do Lambda	625
Habilitar o rastreamento ativo com o AWS CloudFormation	626
Armazenar dependências de tempo de execução em uma camada	626
Rastreamento em aplicativos de exemplo	627
Tutorial – Eclipse IDE	628
Prerequisites	628
Criar e compilar um projeto	628
Aplicativos de amostra	631
Trabalho com Go	633
Manipulador	634
Manipulador de função do Lambda usando tipos estruturados	635
Usar o estado global	636
Contexto	638
Acessar informações do contexto de invocação	638
Implantar arquivos .zip	640
Pré-requisitos	640
Ferramentas e bibliotecas	640
Aplicativos de exemplo	640
Criando um arquivo .zip no macOS e no Linux	641
Criando um arquivo .zip no Windows	641
Implantar imagens de contêiner	643
AWSImagens de base da para Go	643
Clientes de interface de tempo de execução do Go	643
Usar a imagem base Go:1.x	644
Criar uma imagem Go a partir de uma imagem básica do provided.al2	644
Criar uma imagem Go a partir de uma imagem básica alternativa	645
Implantar a imagem do contêiner	646
Registro em log	647

Criar uma função que retorna logs	647
Usar o console do Lambda	648
Usando o console do CloudWatch	648
Usar a AWS Command Line Interface (AWS CLI)	649
Excluir logs	651
Erros	652
Criar uma função que retorna exceções	652
Como funcionam	652
Usar o console do Lambda	653
Usar a AWS Command Line Interface (AWS CLI)	653
Tratamento de erros em outros serviços da AWS	654
Próximas etapas	655
Rastreamento	656
Habilitar o rastreamento ativo com a API do Lambda	658
Habilitar o rastreamento ativo com o AWS CloudFormation	658
Variáveis de ambiente	660
Trabalho com C#	661
Manipulador	663
Tratamento de streams	663
Tratamento de tipos de dados padrão	664
Assinaturas do manipulador	665
Serializar funções do Lambda	665
Restrições do manipulador de função do Lambda	666
Usar async em funções em C# com o AWS Lambda	666
Pacote de implantação	668
CLI do .NET Core	668
AWS Toolkit for Visual Studio	671
Implantar imagens de contêiner	674
AWSImagens de base da para .NET	674
Usar uma imagem base .NET	675
Clientes de interface de tempo de execução .NET	675
Implantar a imagem do contêiner	675
Contexto	676
Registro em log	677
Criar uma função que retorna logs	677
Usar o console do Lambda	678
Usando o console do CloudWatch	678
Usar a AWS Command Line Interface (AWS CLI)	679
Excluir logs	681
Erros	682
Syntax	682
Como funcionam	684
Usar o console do Lambda	685
Usar a AWS Command Line Interface (AWS CLI)	685
Tratamento de erros em outros serviços da AWS	686
Próximas etapas	686
Rastreamento	688
Habilitar o rastreamento ativo com a API do Lambda	690
Habilitar o rastreamento ativo com o AWS CloudFormation	691
Como trabalhar com o PowerShell	692
Ambiente de desenvolvimento	693
Pacote de implantação	694
Criação de uma função do Lambda	694
Manipulador	696
Retorno de dados	696
Contexto	697
Registro em log	698

Criar uma função que retorna logs	698
Usar o console do Lambda	699
Usando o console do CloudWatch	699
Usar a AWS Command Line Interface (AWS CLI)	700
Excluir logs	702
Erros	703
Syntax	703
Como funcionam	704
Usar o console do Lambda	704
Usar a AWS Command Line Interface (AWS CLI)	705
Tratamento de erros em outros serviços da AWS	705
Próximas etapas	706
Monitoramento	707
Console de monitoramento	708
Pricing	708
Usar o console do Lambda	708
Tipos de gráficos de monitoramento	708
Visualizando gráficos no console do Lambda	708
Exibindo consultas no console do CloudWatch Logs	709
Próximas etapas	709
Insights de função	710
Como funcionam	710
Pricing	710
Tempos de execução compatíveis	710
Ativando o Lambda Insights no console	710
Ativação do Lambda Insights por programação	711
Usando o painel do Lambda Insights	711
Detecção de anomalias de função	713
Solução de problemas de função	714
Próximas etapas	709
Métricas de função	717
Visualizar métricas no console do CloudWatch	717
Tipos de métricas	717
Registros de função	720
Prerequisites	720
Pricing	720
Usar o console do Lambda	720
Usar a AWS CLI	721
Próximas etapas	721
Código profiler	721
Tempos de execução compatíveis	722
Ativando o CodeGuru Profiler do console do Lambda	722
O que acontece quando você ativa o CodeGuru Profiler no console do Lambda?	722
Próximas etapas	723
Exemplo de fluxos de trabalho	723
Prerequisites	723
Pricing	724
Exibindo um mapa de serviço	724
Visualizando detalhes de rastreamento	725
Como usar o Trusted Advisor para exibir recomendações	725
Próximas etapas	726
Segurança	727
Proteção de dados	727
Criptografia em trânsito	728
Criptografia em repouso	728
Identity and Access Management	729
Audience	729

Autenticar com identidades	730
Gerenciamento do acesso usando políticas	732
Como o AWS Lambda funciona com o IAM	733
Exemplos de políticas baseadas em identidade	733
Solução de problemas	735
Validação de conformidade	738
Resiliência	738
Segurança da infraestrutura	739
Análise de configuração e vulnerabilidade	740
Solução de problemas	741
Implantação	741
Geral: A permissão foi negada/Não é possível carregar esse arquivo	741
Geral: Ocorre um erro ao acionar o updateFunctionCode	741
Amazon S3: Código de erro PermanentRedirect	742
Geral: Não é possível localizar, não é possível carregar, não é possível importar, classe não encontrada, o arquivo ou diretório não existe	742
Geral: Handler de método indefinido	742
Lambda: InvalidParameterValueException ou RequestEntityToolargeException	743
Lambda: InvalidParameterValueException	743
Invocação	744
IAM: lambda:InvokeFunction não autorizado	744
Lambda: A operação não pode ser executada ResourceConflictException	744
Lambda: A função está paralisada em Pendente	744
Lambda: Uma função está usando toda a simultaneidade	745
Geral: Não é possível invocar a função com outras contas ou serviços	745
Geral: A invocação da função está em loop	745
Lambda: Roteamento de alias com simultaneidade provisionada	745
Lambda: As inicializações a frio começam com simultaneidade provisionada	746
Lambda: Variabilidade de latência com simultaneidade provisionada	746
Lambda: As inicializações a frio começam com novas versões	746
EFS: A função não pôde montar o sistema de arquivos do EFS	747
EFS: A função não pôde se conectar ao sistema de arquivos do EFS	747
EFS: A função não pôde montar o sistema de arquivos do EFS devido ao tempo limite	747
Lambda: O Lambda detectou um processo de E/S que estava demorando muito	747
Execução	748
Lambda: A execução leva muito tempo	748
Lambda: Os logs ou rastreamentos não aparecem	748
Lambda: A função retorna antes da conclusão da execução	748
AWS SDK: versões e atualizações	749
Python: As bibliotecas carregam incorretamente	749
Redes	749
VPC: A função perde o acesso à Internet ou atinge o tempo limite	750
VPC: a função precisa de acesso aos serviços da AWS sem usar a Internet	750
VPC: O limite foi atingido para a VPC da função	750
Imagens de contêiner	750
Contêiner: ocorre um erro no tempo de execução InvalidEntryPoint	750
Lambda: capacidade adicional de provisionamento do sistema	750
CloudFormation: ENTRYPOINT está sendo substituído por um valor nulo ou vazio	751
Versões	752
Atualizações anteriores	764
Referência de API	769
Actions	769
AddLayerVersionPermission	771
AddPermission	775
CreateAlias	779
CreateCodeSigningConfig	783
CreateEventSourceMapping	786

CreateFunction	796
DeleteAlias	808
DeleteCodeSigningConfig	810
DeleteEventSourceMapping	812
DeleteFunction	818
DeleteFunctionCodeSigningConfig	820
DeleteFunctionConcurrency	822
DeleteFunctionEventInvokeConfig	824
DeleteLayerVersion	826
DeleteProvisionedConcurrencyConfig	828
GetAccountSettings	830
GetAlias	832
GetCodeSigningConfig	835
GetEventSourceMapping	837
GetFunction	842
GetFunctionCodeSigningConfig	846
GetFunctionConcurrency	849
GetFunctionConfiguration	851
GetFunctionEventInvokeConfig	858
GetLayerVersion	861
GetLayerVersionByArn	864
GetLayerVersionPolicy	867
GetPolicy	869
GetProvisionedConcurrencyConfig	872
Invoke	875
InvokeAsync	881
ListAliases	883
ListCodeSigningConfigs	886
ListEventSourceMappings	888
ListFunctionEventInvokeConfigs	891
ListFunctions	894
ListFunctionsByCodeSigningConfig	898
ListLayers	900
ListLayerVersions	903
ListProvisionedConcurrencyConfigs	906
ListTags	909
ListVersionsByFunction	911
PublishLayerVersion	915
PublishVersion	919
PutFunctionCodeSigningConfig	927
PutFunctionConcurrency	930
PutFunctionEventInvokeConfig	933
PutProvisionedConcurrencyConfig	937
RemoveLayerVersionPermission	940
RemovePermission	942
TagResource	945
UntagResource	947
UpdateAlias	949
UpdateCodeSigningConfig	953
UpdateEventSourceMapping	956
UpdateFunctionCode	965
UpdateFunctionConfiguration	974
UpdateFunctionEventInvokeConfig	985
Tipos de dados	988
AccountLimit	990
AccountUsage	992
AliasConfiguration	993

AliasRoutingConfiguration	995
AllowedPublishers	996
CodeSigningConfig	997
CodeSigningPolicies	999
Concurrency	1000
DeadLetterConfig	1001
DestinationConfig	1002
Environment	1003
EnvironmentError	1004
EnvironmentResponse	1005
EventSourceMappingConfiguration	1006
FileSystemConfig	1011
FunctionCode	1012
FunctionCodeLocation	1014
FunctionConfiguration	1015
FunctionEventInvokeConfig	1021
ImageConfig	1023
ImageConfigError	1024
ImageConfigResponse	1025
Layer	1026
LayersListItem	1027
LayerVersionContentInput	1028
LayerVersionContentOutput	1029
LayerVersionsListItem	1030
OnFailure	1032
OnSuccess	1033
ProvisionedConcurrencyConfigListItem	1034
SelfManagedEventSource	1036
SourceAccessConfiguration	1037
TracingConfig	1039
TracingConfigResponse	1040
VpcConfig	1041
VpcConfigResponse	1042
Erros de certificado ao usar um SDK	1042
AWSGlossário da	1044

O que é o AWS Lambda?

O Lambda é um serviço de computação que permite que você execute o código sem provisionar ou gerenciar servidores. O Lambda executa seu código em uma infraestrutura de computação de alta disponibilidade e executa toda a administração dos recursos computacionais, inclusive a manutenção do servidor e do sistema operacional, o provisionamento e a escalabilidade automática da capacidade e o monitoramento e o registro em log do código. Com o Lambda, você pode executar código para praticamente qualquer tipo de aplicação ou serviço de backend. Tudo o que você precisa fazer é fornecer o código em uma das [linguagens compatíveis com o Lambda \(p. 214\)](#).

Note

No Guia do desenvolvedor do AWS Lambda, assumimos que você tem experiência com codificação, compilação e implantação de programas usando uma das linguagens compatíveis.

Você organiza seu código em [Funções do Lambda \(p. 18\)](#). O Lambda executa a função somente quando necessário e escala automaticamente, desde algumas solicitações por dia a milhares por segundo. Você paga apenas pelo tempo de computação consumido. Não haverá cobranças quando o código não estiver em execução.

Você pode chamar suas funções do Lambda usando a API do Lambda ou o Lambda pode executar suas funções em resposta a eventos de outros serviços da AWS. Por exemplo, você pode usar o Lambda para:

- Criar gatilhos de processamento de dados para [AWS Serviços](#), como o Amazon Simple Storage Service (Amazon S3) e o Amazon DynamoDB.
- Processar dados de transmissões armazenadas no Amazon Kinesis.
- Criar seu próprio back-end que opera em escala, performance e segurança da AWS.

O Lambda é um serviço altamente disponível. Para obter mais informações, consulte o [Acordo de Nível de Serviço do AWS Lambda](#).

Seções

- [Quando devo usar o Lambda? \(p. 1\)](#)
- [Recursos do Lambda \(p. 2\)](#)
- [Conceitos básicos do Lambda \(p. 3\)](#)
- [Serviços relacionados \(p. 3\)](#)
- [Acesso ao Lambda \(p. 4\)](#)
- [Definição de preço do Lambda \(p. 4\)](#)

Quando devo usar o Lambda?

O Lambda é o serviço de computação ideal para muitos cenários de aplicações, desde que você possa gravar o código da aplicação com o [ambiente do tempo de execução padrão \(p. 219\)](#) do Lambda e dentro dos recursos fornecidos pelo Lambda.

Ao usar o Lambda, você é responsável apenas pelo seu código. O Lambda gerencia a frota de computação que oferece um equilíbrio de memória, CPU, rede e outros recursos para executar seu

código. Como o Lambda gerencia esses recursos, não é possível fazer login para calcular instâncias ou personalizar o sistema operacional no [Tempo de execução fornecido \(p. 214\)](#). O Lambda executa atividades operacionais e administrativas em seu nome, incluindo gerenciamento de capacidade, monitoramento e registro de suas funções do Lambda.

Se você precisar gerenciar seus próprios recursos de computação, o AWS tem outros serviços de computação para atender às suas necessidades. Por exemplo:

- O Amazon Elastic Compute Cloud (Amazon EC2) oferece uma grande variedade de tipos de instância do EC2. Ele permite personalizar sistemas operacionais, configurações de rede e segurança e toda a pilha de software. Você é responsável por provisionar a capacidade, monitorar a integridade e a performance da frota e usar zonas de disponibilidade para tolerância a falhas.
- O AWS Elastic Beanstalk permite implantar e dimensionar aplicações no Amazon EC2. Você mantém a propriedade e o controle total sobre as instâncias subjacentes do EC2.

Recursos do Lambda

Os principais recursos a seguir ajudam você a desenvolver aplicações do Lambda escaláveis, seguras e facilmente extensíveis:

Controles de simultaneidade e escalabilidade

Os [controles de simultaneidade e escalabilidade \(p. 32\)](#), como limites de simultaneidade e simultaneidade provisionada, oferecem controle detalhado sobre a escalabilidade e a capacidade de resposta das aplicações de produção.

Funções definidas como imagens de contêiner

Use ferramentas, fluxos de trabalho e dependências de [imagem de contêiner \(p. 266\)](#) preferidos para criar, testar e implantar suas funções do Lambda.

Assinatura de código

A [assinatura de código \(p. 144\)](#) do Lambda fornece controles de confiança e integridade que permitem verificar se apenas o código inalterado publicado pelos desenvolvedores aprovados foi implantado em suas funções do Lambda.

Extensões Lambda

É possível usar [extensões do Lambda \(p. 228\)](#) para aumentar as funções do Lambda. Por exemplo, use extensões para integrar mais facilmente o Lambda com suas ferramentas favoritas de monitoramento, observabilidade, segurança e governança.

Esquemas de funções

Um esquema de função fornece código de exemplo que mostra como usar o Lambda com outros serviços da AWS ou aplicações de terceiros. Os esquemas incluem predefinições de código de exemplo e configuração de funções para tempos de execução de Node.js e Python.

Acesso ao banco de dados

Um [proxy de banco de dados \(p. 134\)](#) gerencia um grupo de conexões de banco de dados e retransmite consultas provenientes de uma função. Isso permite que uma função atinja altos níveis de simultaneidade sem esgotar conexões de banco de dados.

Acesso aos sistemas de arquivos

Você pode configurar uma função para montar um [Amazon Elastic File System \(Amazon EFS\) \(p. 139\)](#) para um diretório local. Com o Amazon EFS, o código da função pode acessar e modificar os recursos compartilhados de forma segura e com alta simultaneidade.

Conceitos básicos do Lambda

Para trabalhar de forma eficaz com o Lambda, você precisa de experiência em codificação e experiência nos seguintes domínios:

- SO Linux e comandos, bem como conceitos como processos, threads e permissões de arquivos.
- Conceitos de nuvem e conceitos de rede IP (para redes públicas e privadas).
- Conceitos de computação distribuída, como HTTP como um IPC, filas, mensagens, notificações e simultaneidade.
- Familiaridade com os serviços e conceitos de segurança: AWS Identity and Access Management (IAM) e princípios de controle de acesso, e AWS Key Management Service (AWS KMS) e infraestrutura de chave pública.
- Familiaridade com os principais serviços que interagem com o Lambda: Amazon API Gateway, Amazon S3, Amazon Simple Queue Service (Amazon SQS) e DynamoDB.
- Configuração de instâncias do EC2 com Linux.

Se você está usando o Lambda pela primeira vez, recomendamos que você comece com os seguintes tópicos para aprender o básico:

1. Leia a [Visão geral do produto do Lambda](#) e explore a página [Conceitos básicos do Lambda](#).
2. Para criar e testar uma função do Lambda usando o console do Lambda, teste o [exercício de introdução baseado no console \(p. 8\)](#). Esse exercício ensina sobre o modelo de programação do Lambda e outros conceitos.
3. Se você estiver familiarizado com os fluxos de trabalho de imagem de contêiner, tente o exercício de introdução para [criar uma função do Lambda definida como uma imagem de contêiner \(p. 12\)](#).

A AWS também fornece os seguintes recursos para aprender sobre aplicações sem servidor e do Lambda:

- O [Blog de computação da AWS](#) contém artigos úteis sobre o Lambda.
- O [AWS Serverless](#) fornece blogs, vídeos e treinamento relacionados ao desenvolvimento sem servidor da AWS.
- O canal do YouTube [AWS Online Tech Talks](#) inclui vídeos sobre tópicos relacionados ao Lambda. Para obter uma visão geral sobre aplicações sem servidor e o Lambda, consulte o vídeo [Introduction to AWS Lambda & Serverless Applications](#).

Serviços relacionados

O [Lambda integra-se a outros serviços da AWS \(p. 277\)](#) para invocar funções com base nos eventos que você especificar. Por exemplo:

- Use o [API Gateway \(p. 281\)](#) para fornecer um gateway seguro e escalável para APIs da Web que roteiam solicitações HTTP para funções do Lambda.
- Para serviços que geram uma fila ou fluxo de dados (como [DynamoDB \(p. 333\)](#) e [Kinesis \(p. 387\)](#)), o Lambda pesquisa a fila ou o fluxo de dados do serviço e invoca sua função para processar os dados recebidos.
- Defina os eventos do [Amazon S3 \(p. 431\)](#) que chamam uma função do Lambda para processar objetos do Amazon S3, por exemplo, quando um objeto é criado ou excluído.
- Usar uma função Lambda para processar [Amazon SQS \(p. 465\)](#) Mensagens ou [Amazon Simple Notification Service \(Amazon SNS\) \(p. 457\)](#) Notificações do .

- Use o [AWS Step Functions \(p. 481\)](#) para conectar funções do Lambda em conjunto em fluxos de trabalho sem servidor chamados máquinas de estado.

Acesso ao Lambda

Você pode criar, invocar e gerenciar suas funções do Lambda usando qualquer uma das seguintes interfaces:

- Console de Gerenciamento da AWS: fornece uma interface da Web para você acessar suas funções. Para obter mais informações, consulte [Console do Lambda \(p. 40\)](#).
- AWS Command Line Interface (AWS CLI): fornece comandos para um amplo conjunto de serviços da AWS, incluindo o Lambda, e é compatível com o Windows, o macOS e o Linux. Para obter mais informações, consulte [Como usar o Lambda com o AWS CLI \(p. 47\)](#).
- AWS SDKs: fornecem APIs específicas da linguagem e gerenciam muitos dos detalhes da conexão, como cálculo de assinatura, tratamento de repetições de solicitações e tratamento de erros. Para obter mais informações, consulte [AWS SDKs](#).
- AWS CloudFormation: permite criar modelos que definem suas aplicações do Lambda. Para obter mais informações, consulte [AWS LambdaAplicativos do \(p. 192\)](#). O AWS CloudFormation também é compatível com o [AWS Cloud Development Kit \(CDK\)](#).
- AWS Serverless Application Model (AWS SAM): fornece modelos e uma CLI para configurar e gerenciar aplicações sem servidor da AWS. Para obter mais informações, consulte [AWS SAM \(p. 5\)](#).

Definição de preço do Lambda

Não há custo adicional para a criação de funções do Lambda. Há cobranças para executar uma função e para transferência de dados entre o Lambda e outros serviços da AWS. Alguns recursos opcionais do Lambda (como [simultaneidade provisionada \(p. 113\)](#)) também incorrem em cobranças. Para obter mais informações, consulte [Definição de preço do AWS Lambda](#).

Configurar com o Lambda

Para usar o AWS Lambda, é necessária uma conta da AWS. Se você planeja configurar e usar funções do Lambda pela linha de comando, configure a AWS CLI. Você pode configurar outras ferramentas de desenvolvimento e criação conforme necessário para o ambiente e o idioma que você está planejando usar.

Seções

- [AWSConta da \(p. 5\)](#)
- [AWS CLI \(p. 5\)](#)
- [AWS SAM \(p. 5\)](#)
- [AWS SAM CLI \(p. 6\)](#)
- [Ferramentas para imagens de contêiner \(p. 6\)](#)
- [Ferramentas de criação de código \(p. 6\)](#)

AWSConta da

Para usar o Lambda e outros serviços da AWS, você precisa de uma conta da AWS. Se você não tiver uma conta, visite aws.amazon.com e escolha Create an AWS account (Criar uma conta da AWS). Para saber como, consulte [Como faço para criar e ativar uma nova conta da AWS?](#)

Como prática recomendada, crie um usuário do AWS Identity and Access Management (IAM) com permissões de administrador e use este usuário do IAM para todos os trabalhos que não exigem credenciais de raiz. Crie uma senha para acesso ao console e chaves de acesso para usar ferramentas da linha de comando. Para obter instruções, consulte [Criar seu primeiro usuário administrador e o grupo do IAM](#) no Guia do usuário do IAM.

AWS CLI

Se você planeja configurar e usar funções do Lambda pela linha de comando, instale a AWS Command Line Interface (AWS CLI). Os tutoriais neste guia usam a AWS CLI, que tem comandos para todas as operações de API do Lambda. Algumas funcionalidades não estão disponíveis no console do Lambda e só podem ser acessadas com a AWS CLI ou os SDKs da AWS.

Para configurar o AWS CLI, consulte os tópicos a seguir no AWS Command Line Interface Guia do usuário do.

- [Instalar, atualizar e desinstalar a AWS CLI](#)
- [Configurar a AWS CLI](#)

Para verificar se a AWS CLI está configurada corretamente, execute o comando `list-functions` para ver uma lista de suas funções do Lambda na região da AWS atual.

```
aws lambda list-functions
```

AWS SAM

O AWS Serverless Application Model (AWS SAM) é uma extensão para a linguagem do modelo do AWS CloudFormation que permite definir aplicações sem servidor em um nível superior. O AWS SAM abstrai

tarefas comuns, como a criação de função, facilitando a gravação de modelos. O AWS SAM tem suporte direto do AWS CloudFormation, e inclui funcionalidade adicional por meio da AWS CLI e da CLI do AWS SAM.

Para obter mais informações sobre AWS SAM, consulte a [AWS SAM Especificação](#) no AWS Serverless Application Model Guia do desenvolvedor.

AWS SAM CLI

A CLI do AWS SAM é uma ferramenta da linha de comando separada, que você pode usar para gerenciar e testar aplicativos do AWS SAM. Além de comandos para fazer upload de artefatos e iniciar pilhas do AWS CloudFormation, que também estão disponíveis na AWS CLI, a CLI do AWS SAM oferece comandos para validar modelos e executar aplicações localmente em um contêiner do Docker. Você pode usar a CLI do AWS SAM para criar funções implantadas como arquivos .zip ou imagens de contêiner.

Para configurar o AWS SAM CLI, consulte [Instalar o AWS SAM CLI](#) no AWS Serverless Application Model Guia do desenvolvedor.

Ferramentas para imagens de contêiner

Para criar e testar funções implantadas como imagens de contêiner, você pode usar ferramentas de contêiner nativas, como a CLI do Docker.

Para configurar a CLI do Docker, consulte [Obter Docker](#) no site do Docker Docs. Para obter uma introdução ao uso do Docker com AWS, consulte [Conceitos básicos do Amazon ECR usando o AWS CLI](#) no Amazon Elastic Container Registry.

Ferramentas de criação de código

Você pode criar o código da sua função do Lambda nas linguagens compatíveis com o Lambda. Para ver uma lista dos idiomas, consulte [Tempos de execução do Lambda \(p. 214\)](#). Existem ferramentas para criação de código, como o console do Lambda, o integrated development environment (IDE – ambiente de desenvolvimento integrado) do Eclipse e o IDE do Visual Studio. Mas as ferramentas e opções disponíveis dependem do seguinte:

- A linguagem que você usa para escrever seu código de função do Lambda.
- As bibliotecas que você usa no seu código. Os tempos de execução do Lambda oferecem algumas das bibliotecas, e você deve fazer upload das bibliotecas adicionais que for usar.

A tabela a seguir lista as linguagens compatíveis com o Lambda e as ferramentas e opções que você pode usar com elas.

Linguagem	Ferramentas e opções para criar código
Node.js	<ul style="list-style-type: none">• Console do Lambda• Visual Studio, com plugin IDE (consulte AWS Lambda Suporte no Visual Studio no Blog do desenvolvedor da AWS)• O seu próprio ambiente de criação

Linguagem	Ferramentas e opções para criar código
Java	<ul style="list-style-type: none">Eclipse, com o AWS Toolkit for EclipseIntelliJ, com o AWS Toolkit for JetBrainsO seu próprio ambiente de criação
C#	<ul style="list-style-type: none">Visual Studio, com o plugin IDE (consulte AWS Toolkit for Visual Studio).NET Core (consulte Fazer download do .NET no site da Microsoft)O seu próprio ambiente de criação
Python	<ul style="list-style-type: none">Console do LambdaPyCharm, com o AWS Toolkit for JetBrainsO seu próprio ambiente de criação
Ruby	<ul style="list-style-type: none">Console do LambdaO seu próprio ambiente de criação
Go	<ul style="list-style-type: none">O seu próprio ambiente de criação
PowerShell	<ul style="list-style-type: none">O seu próprio ambiente de criaçãoPowerShell Core 6.0 (consulte Instalar várias versões do PowerShell no site do Microsoft Docs).NET Core 3.1 SDK (consulte Fazer download do .NET no site da Microsoft)Módulo AWSLambdaPSCore (consulte AWSLambdaPSCore no site da PowerShell Gallery)

Conceitos básicos do Lambda

Para começar a usar o Lambda, use o console do Lambda para criar uma função. Em alguns minutos, você pode criar uma função, invocá-la e visualizar logs, métricas e dados de rastreamento.

Note

Para usar o Lambda e outros serviços da AWS, você precisa de uma conta da AWS. Se você não tiver uma conta, visite aws.amazon.com e escolha Create an AWS account (Criar uma conta da AWS). Para saber como, consulte [Como faço para criar e ativar uma nova conta da AWS?](#)

Como prática recomendada, crie um usuário do AWS Identity and Access Management (IAM) com permissões de administrador e use este usuário do IAM para todos os trabalhos que não exigem credenciais de raiz. Crie uma senha para acesso ao console e chaves de acesso para usar ferramentas da linha de comando. Para obter instruções, consulte [Criar seu primeiro usuário administrador e grupo de usuários do IAM](#) no Guia do usuário do IAM.

Você pode criar funções no console do Lambda ou com um toolkit IDE, ferramentas da linha de comando ou os AWS SDKs. O console do Lambda fornece um [editor de código \(p. 40\)](#) para as linguagens não compiladas que permite modificar e testar o código rapidamente. A [AWS Command Line Interface \(AWS CLI\) \(p. 47\)](#) concede acesso direto à API do Lambda para configuração avançada e casos de uso de automação.

Você implanta seu código de função para o Lambda usando um pacote de implantação. O Lambda oferece suporte a dois tipos de implantação:

- Um arquivo .zip que contém seu código de função e as dependências dele. Para obter um tutorial de exemplo, consulte [Criar uma função do Lambda com o console \(p. 9\)](#).
- Uma imagem de contêiner compatível com a especificação [Open Container Initiative \(OCI\)](#). Para obter um tutorial de exemplo, consulte [Criar uma função definida como uma imagem de contêiner \(p. 12\)](#).

Tópicos

- [Criar uma função do Lambda com o console \(p. 9\)](#)
- [Criar uma função definida como uma imagem de contêiner \(p. 12\)](#)

Criar uma função do Lambda com o console

Neste exercício de conceitos básicos, você cria uma função do Lambda usando o console. A função usa o código padrão criado pelo Lambda. O console do Lambda fornece um [editor de código \(p. 40\)](#) para as linguagens não compiladas que permite modificar e testar o código rapidamente.

Tópicos

- [Criar a função \(p. 9\)](#)
- [Invocar a função do Lambda \(p. 9\)](#)
- [Limpar \(p. 10\)](#)

Criar a função

Você criará uma função Node.js do Lambda usando o console do Lambda. O Lambda cria automaticamente o código padrão para a função.

Para criar uma função do Lambda com o console

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha Create function.
3. Em Basic information (Informações básicas), faça o seguinte:
 - a. Em Function name (Nome da função), insira **my-function**.
 - b. Para Runtime (Tempo de execução), confirme se Node.js 14.x está selecionado. Observe que o Lambda fornece tempos de execução para .NET (PowerShell, C#) Go, Java, Node.js, Python e Ruby.
4. Escolha Create function.

O Lambda cria uma função Node.js e uma [função de execução \(p. 57\)](#) que concede à função permissão para fazer upload de logs. A função do Lambda assume a função de execução quando você invoca sua função e usa a função de execução para criar credenciais para o AWS SDK e ler dados de origens de eventos.

Invocar a função do Lambda

Invoque a função do Lambda usando os exemplos de dados de eventos fornecidos no console.

Para invocar uma função

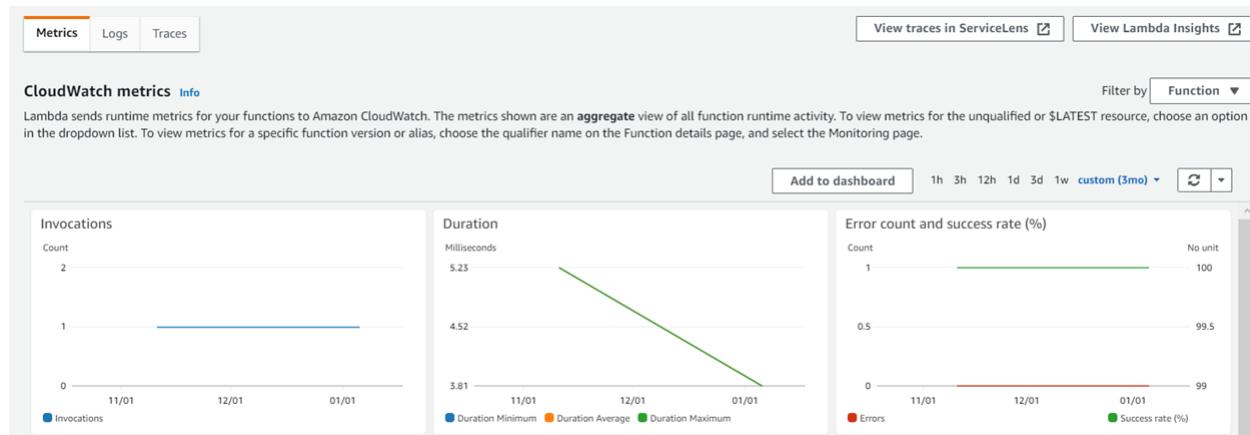
1. Depois de selecionar sua função, escolha a guia Test (Teste).
2. Na seção Test event (Evento de teste), escolha New event (Novo evento). Em Template (Modelo), deixe a opção hello-mundo padrão. Insira um Name (Nome) para esse teste e observe o seguinte exemplo de modelo de evento:

```
{  
    "key1": "value1",  
    "key2": "value2",  
    "key3": "value3"  
}
```

3. Escolha Save changes (Salvar alterações) e, em seguida, escolha Test (Testar). Cada usuário pode criar até 10 eventos de teste por função. Esses eventos de teste não estão disponíveis para outros usuários.

O Lambda executa a função em seu nome. O handler da função recebe e processa o evento de exemplo.

4. Após a conclusão bem-sucedida, visualize os resultados no console.
 - A opção Execution result (Resultado da execução) mostra o status de execução como succeeded (bem-sucedido). Expanda Details (Detalhes) para visualizar os resultados da execução da função. Observe que o link logs abre a página Log groups (Grupos de logs) no console do CloudWatch.
 - A seção Summary mostra as principais informações relatadas na seção Log output (a linha REPORT no log de execução).
 - A seção Log output (Saída de logs) mostra o log que o Lambda gera para cada invocação. A função grava esses logs no CloudWatch. O console do Lambda exibe esses logs para sua conveniência. Escolha Click here (Clique aqui) para adicionar logs ao grupo de logs do CloudWatch e abrir a página Log groups (Grupos de logs) no console do CloudWatch.
5. Execute a função (escolha Test (Testar)) mais algumas vezes para reunir algumas métricas que você possa visualizar na próxima etapa.
6. Escolha a guia Monitoring (Monitoramento). Esta página mostra gráficos relacionados às métricas que o Lambda envia ao CloudWatch.



Para obter mais informações sobre esses gráficos, consulte [Funções de monitoramento no console do AWS Lambda \(p. 708\)](#).

Limpar

Se você tiver terminado de trabalhar com a função de exemplo, exclua-a. Você também pode excluir a função de execução que o console criou e o grupo de logs que armazena os logs da função.

Para excluir uma função do Lambda

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Actions, Delete.
4. Na caixa de diálogo de confirmação Delete function (Excluir função), escolha Delete (Excluir).

Para excluir o grupo de logs

1. Abra a página [Log groups \(Grupos de log\)](#) do console do CloudWatch.
2. Selecione o grupo de logs da função (/aws/lambda/my-function).

3. Escolha Actions (Ações), Delete log group(s) (Excluir grupo(s) de log).
4. Na caixa de diálogo Delete log group(s) (Excluir grupo(s) de log), escolha Delete (Excluir).

Para excluir a função de execução

1. Abra a [página Roles](#) (Funções) no console do AWS Identity and Access Management (IAM).
2. Selecione o papel da função (`my-function-role-31example`).
3. Selecione Delete role (Excluir função).
4. Na caixa de diálogo Delete role (Excluir função), selecione Yes, delete (Sim, excluir).

Você pode automatizar a criação e a limpeza de funções, papéis e grupos de log com o AWS CloudFormation e a AWS Command Line Interface (AWS CLI). Para aplicações de exemplo totalmente funcionais, consulte [Aplicativos de exemplo do Lambda \(p. 491\)](#).

Criar uma função do definida como uma imagem de contêiner

Neste exercício de conceitos básicos, você cria uma função definida como uma imagem de contêiner. Primeiro, use a CLI do Docker para criar uma imagem de contêiner para o código da função. Em seguida, use o console do Lambda para criar uma função a partir da imagem do contêiner.

Tópicos

- [Prerequisites \(p. 12\)](#)
- [Criar a imagem de contêiner \(p. 12\)](#)
- [Fazer upload da imagem para o repositório do Amazon ECR \(p. 13\)](#)
- [Atualizar as permissões do usuário \(p. 14\)](#)
- [Criar uma função do Lambda definida como uma imagem de contêiner \(p. 14\)](#)
- [Invocar a função do Lambda \(p. 14\)](#)
- [Limpar \(p. 15\)](#)

Prerequisites

Para concluir as etapas a seguir, você precisa de um terminal de linha de comando ou de um shell para executar os comandos. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

Você deve ver a saída a seguir:

```
aws-cli/2.0.57 Python/3.7.4 Darwin/19.6.0 exe/x86_64
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

Este exercício usa comandos da CLI do Docker para criar a imagem de contêiner. Para instalar a CLI do Docker, consulte [Get Docker](#) no site Docker Docs.

Criar a imagem de contêiner

AWS fornece um conjunto de imagens de base no Amazon Elastic Container Registry (Amazon ECR). Neste exercício de conceitos básicos, usamos a imagem básica Node.js para criar uma imagem de contêiner. Para obter mais informações sobre imagens básicas, consulte [AWS Imagens base para o Lambda \(p. 222\)](#).

Nos comandos a seguir, substitua 123456789012 pelo ID da sua conta da AWS.

Para criar uma imagem usando a imagem básica Node.js 14 da AWS

1. Em sua máquina local, crie um diretório de projeto para sua nova função.
2. Crie um arquivo chamado `app.js` no diretório do projeto. Adicione o seguinte código a `app.js`:

```
exports.handler = async (event) => {
  // TODO implement
  const response = {
    statusCode: 200,
    body: JSON.stringify('Hello from Lambda!'),
  };
  return response;
};
```

3. Use um editor de texto para criar um novo arquivo chamado Dockerfile no diretório do projeto. Adicione o seguinte conteúdo ao arquivo Dockerfile:

```
FROM public.ecr.aws/lambda/nodejs:14

# Copy function code
COPY app.js ${LAMBDA_TASK_ROOT}

# Set the CMD to your handler
CMD [ "app.handler" ]
```

4. Crie a imagem do Docker: Execute o seguinte comando no diretório do projeto.

```
docker build -t hello-world .
```

5. (Opcional) As imagens básicas da AWS incluem o emulador de interface de tempo de execução do Lambda, assim você pode testar sua função localmente.

- a. Execute a imagem do Docker: No diretório do projeto, execute o comando docker run:

```
docker run -p 9000:8080 hello-world:latest
```

- b. Teste sua função do Lambda Em uma nova janela de terminal, execute um comando curl para invocar sua função:

```
curl -XPOST "http://localhost:9000/2015-03-31/functions/function/invocations" -d
'{}'
```

Fazer upload da imagem para o repositório do Amazon ECR

1. Autentique a CLI do Docker no seu registro do Amazon ECR.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-
stdin 123456789012.dkr.ecr.us-east-1.amazonaws.com
```

2. Crie um repositório no Amazon ECR usando o comando create-repository.

```
aws ecr create-repository --repository-name hello-world --image-scanning-configuration
scanOnPush=true --image-tag-mutability MUTABLE
```

3. Marque sua imagem para corresponder ao nome do repositório usando o comando docker tag.

```
docker tag hello-world:latest 123456789012.dkr.ecr.us-east-1.amazonaws.com/hello-
world:latest
```

4. Implante a imagem no Amazon ECR usando o comando docker push.

```
docker push 123456789012.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

Atualizar as permissões do usuário

Certifique-se de que as permissões para o usuário ou função do IAM que cria a função contenham as políticas gerenciadas da AWS GetRepositoryPolicy e SetRepositoryPolicy. Para obter informações sobre permissões de usuário para acessar imagens no repositório do Amazon ECR, consulte [the section called “Permissões do Amazon ECR” \(p. 91\)](#)

Por exemplo, use o console do IAM para criar uma função com a seguinte política:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "VisualEditor0",  
            "Effect": "Allow",  
            "Action": ["ecr:SetRepositoryPolicy", "ecr:GetRepositoryPolicy"],  
            "Resource": "arn:aws:ecr:<region>:<account>:repository/<repo name>/"  
        }  
    ]  
}
```

Criar uma função do Lambda definida como uma imagem de contêiner

Use o console do Lambda para criar uma função definida como uma imagem de contêiner.

Para criar uma função com o console

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha Create function.
3. Escolha a opção Container image (Imagen de contêiner).
4. Em Basic information (Informações básicas), faça o seguinte:
 - a. Em Function name (Nome da função), insira **my-function**.
 - b. Para Container image URI (URI de imagem de contêiner), insira o URI da imagem do Amazon ECR que você criou anteriormente.
5. Escolha Create function.

O Lambda cria sua função e uma [função de execução \(p. 57\)](#) que concede à função permissão para fazer upload de logs. O Lambda assume a função de execução quando você invoca sua função e usa a função de execução para criar credenciais para o AWS SDK e ler dados de fontes de eventos.

Invocar a função do Lambda

Invoque a função do Lambda usando os exemplos de dados de eventos fornecidos no console.

Para invocar uma função

1. Depois de selecionar sua função, escolha a guia Test (Teste).

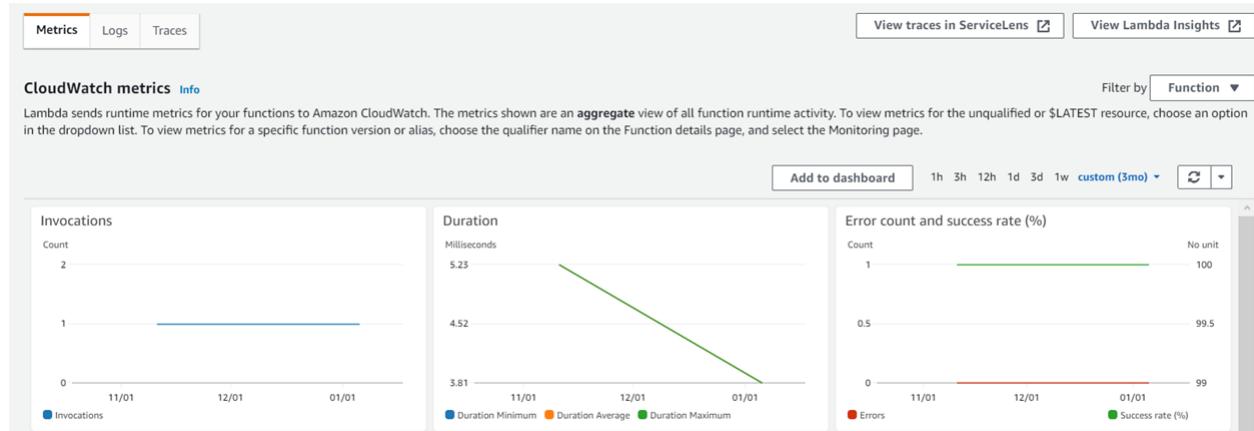
- Na seção Test event (Evento de teste), escolha New event (Novo evento). Em Template (Modelo), deixe a opção hello-mundo padrão. Insira um Name (Nome) para esse teste e observe o seguinte exemplo de modelo de evento:

```
{
    "key1": "value1",
    "key2": "value2",
    "key3": "value3"
}
```

- Escolha Save changes (Salvar alterações) e, em seguida, escolha Test (Testar). Cada usuário pode criar até 10 eventos de teste por função. Esses eventos de teste não estão disponíveis para outros usuários.

O Lambda executa a função em seu nome. O handler da função recebe e processa o evento de exemplo.

- Após a conclusão bem-sucedida, visualize os resultados no console.
 - A opção Execution result (Resultado da execução) mostra o status de execução como succeeded (bem-sucedido). Expanda Details (Detalhes) para visualizar os resultados da execução da função. Observe que o link logs abre a página Log groups (Grupos de logs) no console do CloudWatch.
 - A seção Summary mostra as principais informações relatadas na seção Log output (a linha REPORT no log de execução).
 - A seção Log output (Saída de logs) mostra o log que o Lambda gera para cada invocação. A função grava esses logs no CloudWatch. O console do Lambda exibe esses logs para sua conveniência. Escolha Click here (Clique aqui) para adicionar logs ao grupo de logs do CloudWatch e abrir a página Log groups (Grupos de logs) no console do CloudWatch.
- Execute a função (escolha Test (Testar)) mais algumas vezes para reunir algumas métricas que você possa visualizar na próxima etapa.
- Escolha a guia Monitoring (Monitoramento). Esta página mostra gráficos relacionados às métricas que o Lambda envia ao CloudWatch.



Para obter mais informações sobre esses gráficos, consulte [Funções de monitoramento no console do AWS Lambda \(p. 708\)](#).

Limpar

Se você terminou de lidar com a imagem do contêiner, consulte [Excluir uma imagem no Amazon Elastic Container Registry](#)

Se você tiver terminado de trabalhar com a função, exclua-a. Você também pode excluir a função de execução que o console criou e o grupo de logs que armazena os logs da função.

Para excluir uma função do Lambda

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Actions, Delete.
4. Na caixa de diálogo de confirmação Delete function (Excluir função), escolha Delete (Excluir).

Para excluir o grupo de logs

1. Abra a página [Log groups \(Grupos de log\)](#) do console do CloudWatch.
2. Selecione o grupo de logs da função (/aws/lambda/my-function).
3. Escolha Actions (Ações), Delete log group(s) (Excluir grupo(s) de log).
4. Na caixa de diálogo Delete log group(s) (Excluir grupo(s) de log), escolha Delete (Excluir).

Para excluir a função de execução

1. Abra a página [Roles \(Funções\)](#) no console do IAM.
2. Selecione o papel da função (my-function-role-31exxmpl).
3. Selecione Delete role (Excluir função).
4. Na caixa de diálogo Delete role (Excluir função), selecione Yes, delete (Sim, excluir).

Você pode automatizar a criação e a limpeza de funções, papéis e grupos de log com o AWS CloudFormation e a AWS CLI. Para aplicações de exemplo totalmente funcionais, consulte [Aplicativos de exemplo do Lambda \(p. 491\)](#).

Fundamentos AWS Lambda

A função do Lambda é o princípio fundamental do Lambda. Você pode configurar suas funções usando o console do Lambda, a API do Lambda, AWS CloudFormation ou AWS SAM. Você cria código para a função e carrega o código usando um pacote de implantação. Quando o evento ocorre, o Lambda invoca a função. O Lambda executa várias instâncias de sua função em paralelo, regidas por simultaneidade e limites de escalabilidade.

Tópicos

- [Conceitos Lambda \(p. 18\)](#)
- [Recursos do Lambda \(p. 22\)](#)
- [Modelo de programação do Lambda \(p. 30\)](#)
- [Escalabilidade da função do Lambda \(p. 32\)](#)
- [Pacotes de implantação do Lambda \(p. 37\)](#)
- [Console do Lambda \(p. 40\)](#)
- [Como usar o Lambda com o AWS CLI \(p. 47\)](#)
- [Cotas Lambda \(p. 53\)](#)

Conceitos Lambda

O Lambda executa instâncias de sua função para processar eventos. Para enviar eventos para a função, você pode invocá-la usando a API do Lambda ou pode configurar um serviço da AWS ou um recurso para invocá-la.

Conceitos

- [Function \(p. 18\)](#)
- [Trigger \(p. 18\)](#)
- [Event \(p. 18\)](#)
- [Ambiente de execução \(p. 19\)](#)
- [Pacote de implantação \(p. 19\)](#)
- [Runtime \(p. 19\)](#)
- [Layer \(p. 19\)](#)
- [Extension \(p. 20\)](#)
- [Concurrency \(p. 20\)](#)
- [Qualifier \(p. 20\)](#)
- [Destination \(p. 21\)](#)

Function

Uma função é um recurso que você pode invocar para executar o código no Lambda. Uma função tem código para processar os [eventos \(p. 18\)](#) que você transmite para a função ou que outros serviços da AWS enviam para a função.

Para obter mais informações, consulte [Configurar funções do AWS Lambda \(p. 81\)](#).

Trigger

Um acionador é um recurso ou uma configuração que invoca uma função do Lambda. Acionadores incluem serviços da AWS que podem ser configurados para invocar uma função e [mapeamentos da fonte do evento \(p. 170\)](#). Um mapeamento da fonte do evento é um recurso no Lambda que lê itens de um fluxo ou de uma fila e invoca uma função. Para obter mais informações, consulte [Chamada de funções do AWS Lambda \(p. 157\)](#) e [Usar o AWS Lambda com outros serviços \(p. 277\)](#).

Event

Um evento é um documento no formato JSON que contém dados para uma função do Lambda processar. O tempo de execução converte o evento em um objeto e o transmite para o código da função. Ao invocar uma função, você determina a estrutura e o conteúdo do evento.

Example evento personalizado – dados climáticos

```
{  
    "TemperatureK": 281,  
    "WindKmh": -3,  
    "HumidityPct": 0.55,  
    "PressureHPa": 1020  
}
```

Quando um serviço da AWS invoca a função, ele define a forma do evento.

Example Evento de serviço — Notificação do Amazon SNS

```
{  
  "Records": [  
    {  
      "Sns": {  
        "Timestamp": "2019-01-02T12:45:07.000Z",  
        "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEkAi6RibDsvpi+tE/1+82j...65r==",  
        "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",  
        "Message": "Hello from SNS!",  
        ...  
      }  
    }  
  ]  
}
```

Para obter mais informações sobre eventos de serviços da AWS, consulte [Usar o AWS Lambda com outros serviços \(p. 277\)](#).

Ambiente de execução

Um ambiente de execução fornece um ambiente de execução seguro e isolado para a função do Lambda. Um ambiente de execução gerencia os processos e recursos necessários para executar a função. O ambiente de execução fornece suporte ao ciclo de vida para a função e a qualquer [extensão \(p. 20\)](#) associada à função.

Para obter mais informações, consulte [AWS Lambda Ambiente de execução do \(p. 219\)](#).

Pacote de implantação

Você implanta o código da função Lambda usando um pacote de implantação. O Lambda oferece suporte a dois tipos de pacote de implantação:

- Um arquivo .zip que contém seu código de função e as dependências dele. O Lambda fornece o sistema operacional e o tempo de execução para sua função.
- Uma imagem de contêiner compatível com a especificação [Open Container Initiative \(OCI\)](#). Você adiciona o código da função e as dependências à imagem. Também é necessário incluir o sistema operacional e um tempo de execução do Lambda.

Para obter mais informações, consulte [Pacotes de implantação do Lambda \(p. 37\)](#).

Runtime

O tempo de execução fornece um ambiente específico de linguagem que é executado no ambiente de execução. O tempo de execução transmite eventos de invocação, informações de contexto e respostas entre o Lambda e a função. Você pode usar tempos de execução fornecidos pelo Lambda ou criar seus próprios. Se você empacotar seu código como um arquivo de arquivo.zip, você deve configurar sua função para usar um tempo de execução que corresponda à sua linguagem de programação. Para uma imagem de contêiner, você inclui o tempo de execução ao compilar a imagem.

Para obter mais informações, consulte [Tempos de execução do Lambda \(p. 214\)](#).

Layer

Uma camada do Lambda é um arquivo .zip que pode conter código adicional ou outro conteúdo. Uma camada pode conter bibliotecas, um [tempo de execução personalizado \(p. 255\)](#), dados ou arquivos de configuração.

As camadas fornecem uma maneira conveniente de empacotar bibliotecas e outras dependências que você pode usar com suas funções do Lambda. O uso de camadas reduz o tamanho dos arquivos

de implantação carregados e acelera a implantação do código. As camadas também promovem o compartilhamento de código e a separação de responsabilidades para que você possa iterar mais rapidamente na escrita da lógica de negócios.

Você pode incluir até cinco camadas por função. As camadas contam para as [cotas de tamanho de implantação \(p. 53\)](#) padrão do Lambda. Quando você inclui uma camada em uma função, o conteúdo é extraído para o diretório /opt no ambiente de execução.

Por padrão, as camadas que você cria são privadas na sua conta da AWS. Você pode optar por compartilhar uma camada com outras contas ou tornar a camada pública. Se suas funções consomem uma camada que uma conta diferente publicou, suas funções poderão continuar a usar a versão da camada depois que ela tiver sido excluída ou depois que sua permissão para acessar a camada for revogada. No entanto, você não pode criar uma nova função ou atualizar funções usando uma versão de camada excluída.

Funções implantadas como uma imagem de contêiner não usam camadas. Em vez disso, você empacota seu tempo de execução preferido, bibliotecas e outras dependências na imagem do contêiner ao criar a imagem.

Para obter mais informações, consulte [Criar e compartilhar camadas do Lambda \(p. 85\)](#).

Extension

As extensões do Lambda permitem que você aumente as funções. Por exemplo, você pode usar extensões para integrar as funções com suas ferramentas preferidas de monitoramento, observação, segurança e governança. Você pode escolher entre várias ferramentas que os [parceiros do AWS Lambda](#) fornecem, ou pode [criar suas próprias extensões do Lambda \(p. 228\)](#).

Uma extensão interna é executada no processo de tempo de execução e compartilha o mesmo ciclo de vida que o tempo de execução. Uma extensão externa é executada como um processo separado no ambiente de execução. A extensão externa é inicializada antes que a função seja invocada, é executada em paralelo com o tempo de execução da função e continua a ser executada após a conclusão da invocação da função.

Para obter mais informações, consulte [Uso de extensões do Lambda \(p. 178\)](#).

Concurrency

Simultaneidade é o número de solicitações que a função atende a cada momento. Quando a função é invocada, o Lambda provisiona uma instância se ela processar o evento. Quando a execução do código da função terminar, ela poderá processar outra solicitação. Se a função for invocada novamente enquanto uma solicitação ainda estiver sendo processada, outra instância será provisionada, aumentando a simultaneidade da função.

A simultaneidade está sujeita a [cotas \(p. 53\)](#) no nível da região da AWS. É possível configurar funções individuais limitando sua simultaneidade ou permitindo que elas alcancem um nível específico de simultaneidade. Para obter mais informações, consulte [Gerenciar simultaneidade para uma função do Lambda \(p. 113\)](#).

Qualifier

Ao chamar ou visualizar uma função, é possível incluir um qualificador para especificar uma versão ou um alias. Uma versão é um snapshot imutável da configuração e do código de uma função que tem um qualificador numérico. Por exemplo, `my-function:1`. Um alias é um ponteiro para uma versão que pode ser atualizado para mapear para uma versão diferente ou dividir o tráfego entre duas versões. Por exemplo, `my-function:BLUE`. É possível usar versões e aliases em conjunto para fornecer uma interface estável para os clientes chamarem sua função.

Para obter mais informações, consulte [Versões da função do Lambda \(p. 106\)](#).

Destination

A destination é um AWS Lambda que o Lambda pode enviar eventos de uma invocação assíncrona. É possível configurar um destino para eventos que tiveram falha no processamento. Alguns serviços também oferecem suporte a um destino para eventos que são processados com êxito.

Para obter mais informações, consulte [Configurar destinos para invocação assíncrona \(p. 164\)](#).

Recursos do Lambda

O Lambda fornece um console de gerenciamento e uma API para gerenciar e invocar funções. Proporciona tempos de execução que oferecem suporte a um conjunto padrão de recursos para que você possa alternar facilmente entre linguagens e frameworks, dependendo do que precisar. Além das funções, também é possível criar versões, aliases, camadas e tempos de execução personalizados.

Recursos

- [Scaling \(p. 22\)](#)
- [Controles de simultaneidade \(p. 23\)](#)
- [Invocação assíncrona \(p. 25\)](#)
- [Mapeamentos de origem do evento \(p. 26\)](#)
- [Destinations \(p. 27\)](#)
- [Esquemas de funções \(p. 28\)](#)
- [Ferramentas de teste e implantação \(p. 29\)](#)
- [Modelos de aplicativos \(p. 29\)](#)

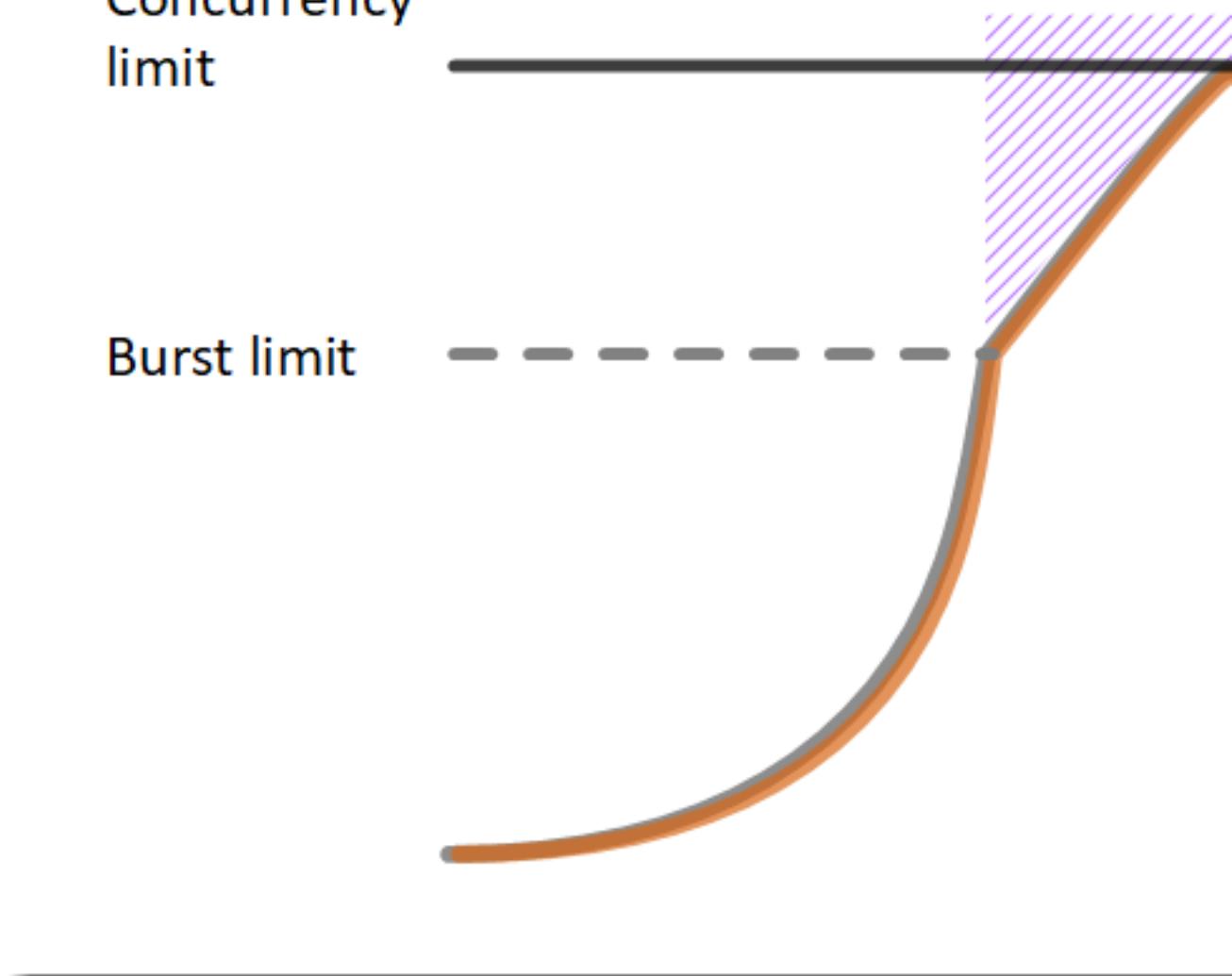
Scaling

O Lambda gerencia a infraestrutura que executa o código, e a dimensiona automaticamente em resposta às solicitações recebidas. Quando a função é invocada mais rapidamente do que a capacidade de uma única instância da função processar eventos, o Lambda faz o dimensionamento executando instâncias adicionais. Quando o tráfego diminui, as instâncias inativas são congeladas ou interrompidas. Você só paga pelo tempo em que sua função está inicializando ou processando eventos.

Function Scaling with Concurrency Limit

Concurrency
limit

Burst limit

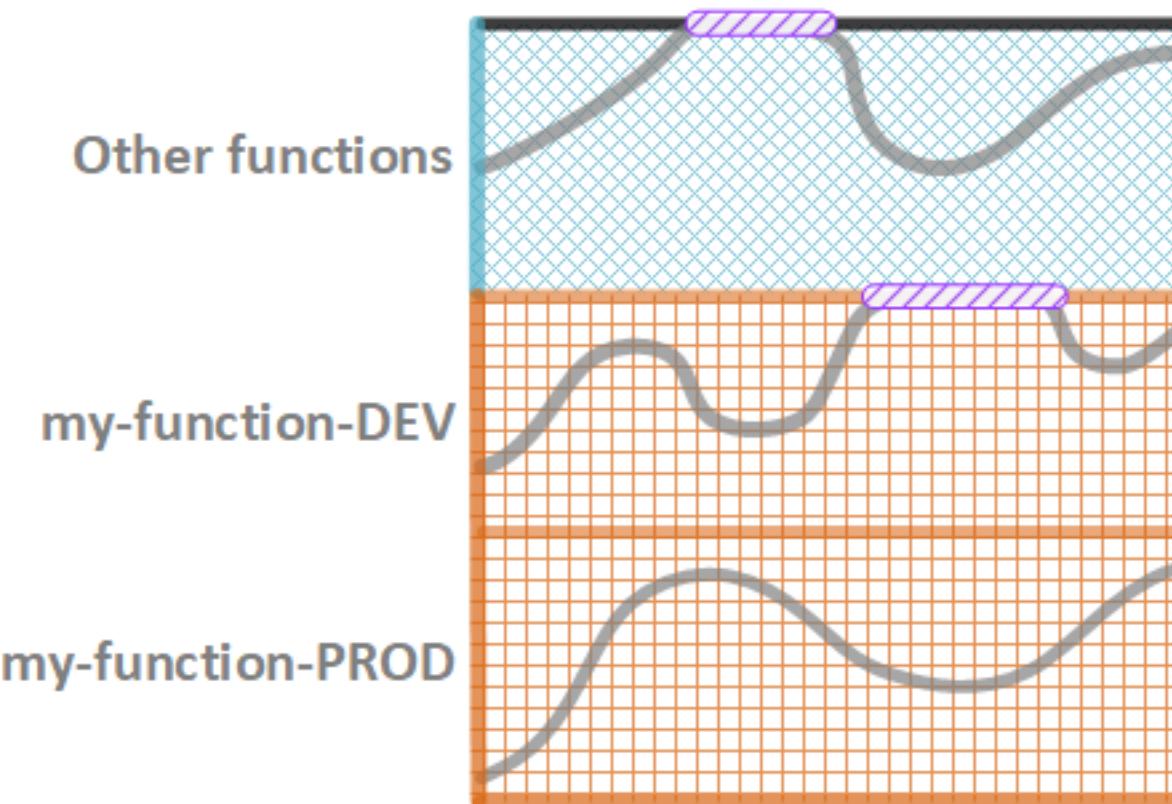


Para obter mais informações, consulte [Escalabilidade da função do Lambda \(p. 32\)](#).

Controles de simultaneidade

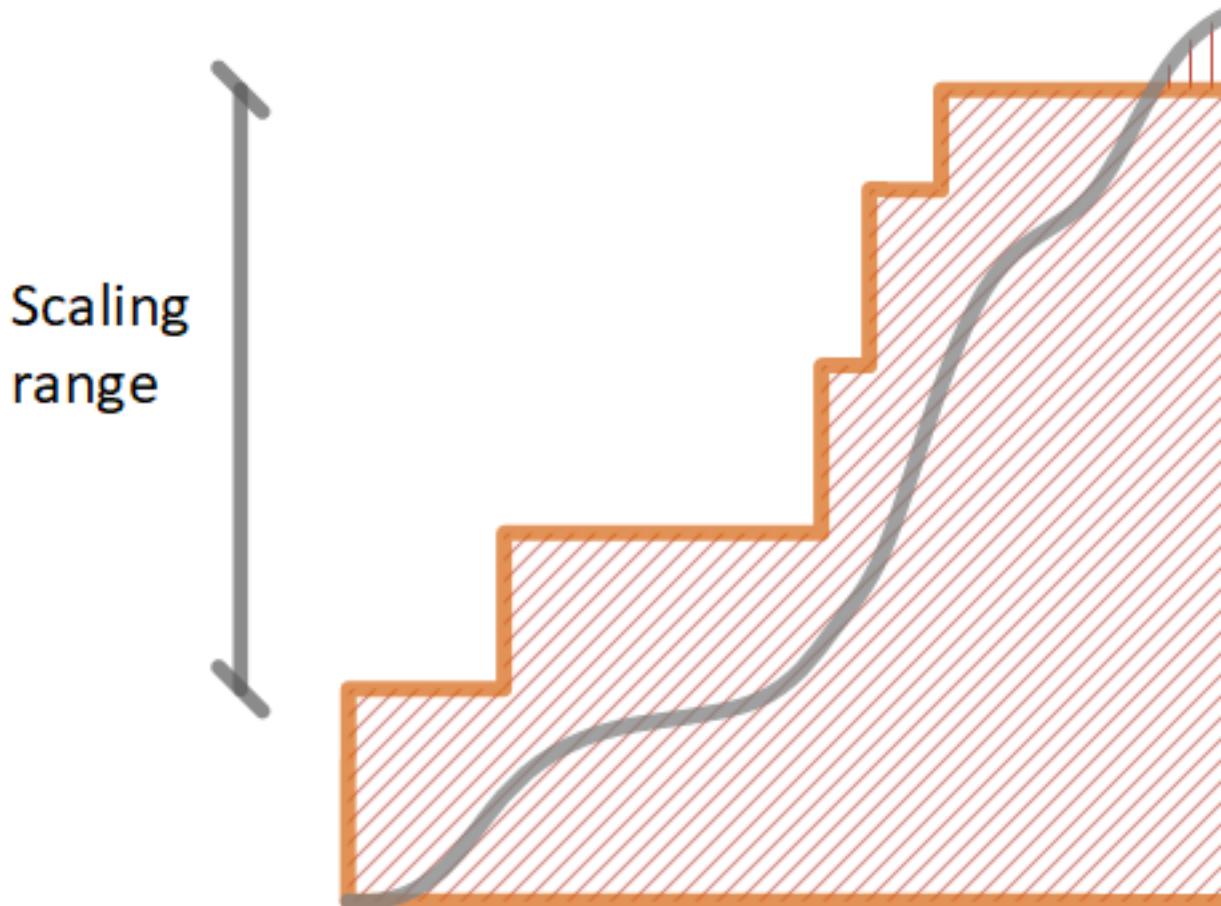
Use configurações de simultaneidade para garantir que seus aplicativos de produção estejam altamente disponíveis e altamente responsivos. Para evitar que uma função use muita simultaneidade e reservar uma parte da simultaneidade disponível da sua conta para uma função, use a simultaneidade reservada. A simultaneidade reservada divide o pool de simultaneidade disponível em subconjuntos. Uma função com simultaneidade reservada usa apenas a simultaneidade do pool dedicado dela.

Reserved Concurrency



Para permitir que as funções sejam dimensionadas sem flutuações na latência, use simultaneidade provisionada. Em caso de funções que levam muito tempo para serem inicializadas ou que exigem latência extremamente baixa para todas as chamadas, a simultaneidade provisionada permite pré-inicializar instâncias de sua função e mantê-las em execução o tempo todo. O Lambda integra-se ao Application Auto Scaling para oferecer suporte ao autoscaling para simultaneidade provisionada com base na utilização.

Autoscaling with Provisioned Concurrency

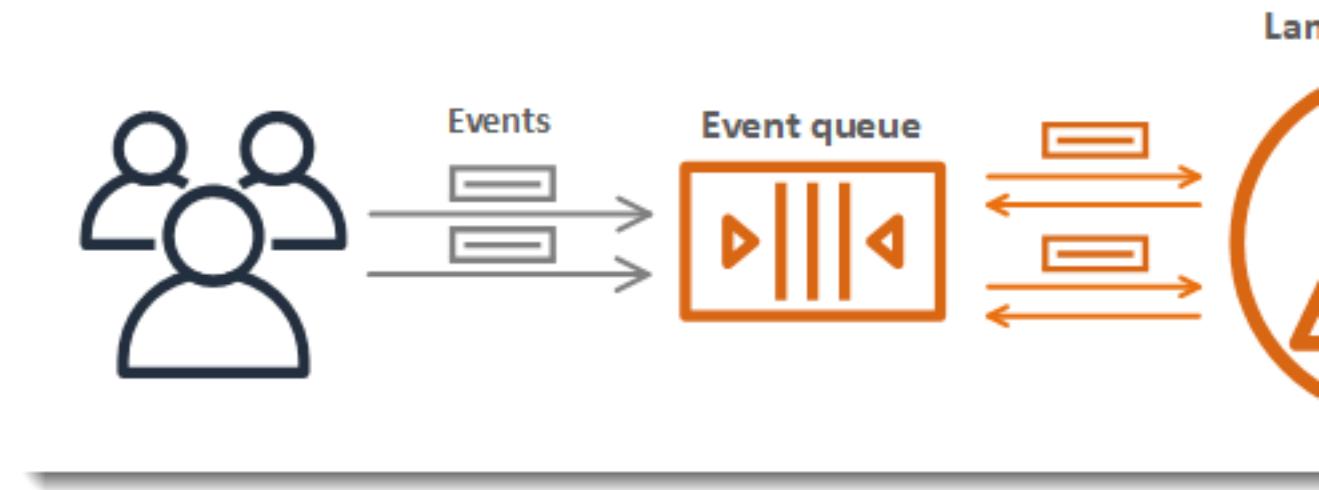


Para obter mais informações, consulte [Gerenciar simultaneidade para uma função do Lambda \(p. 113\)](#).

Invocação assíncrona

Quando você invocar uma função, poderá optar por invocá-la de forma síncrona ou assíncrona. Com a [invocação síncrona \(p. 158\)](#), você aguarda a função processar o evento e retornar uma resposta. Com a invocação assíncrona, o Lambda coloca o evento na fila para processamento e retorna uma resposta imediatamente.

Asynchronous Invocation



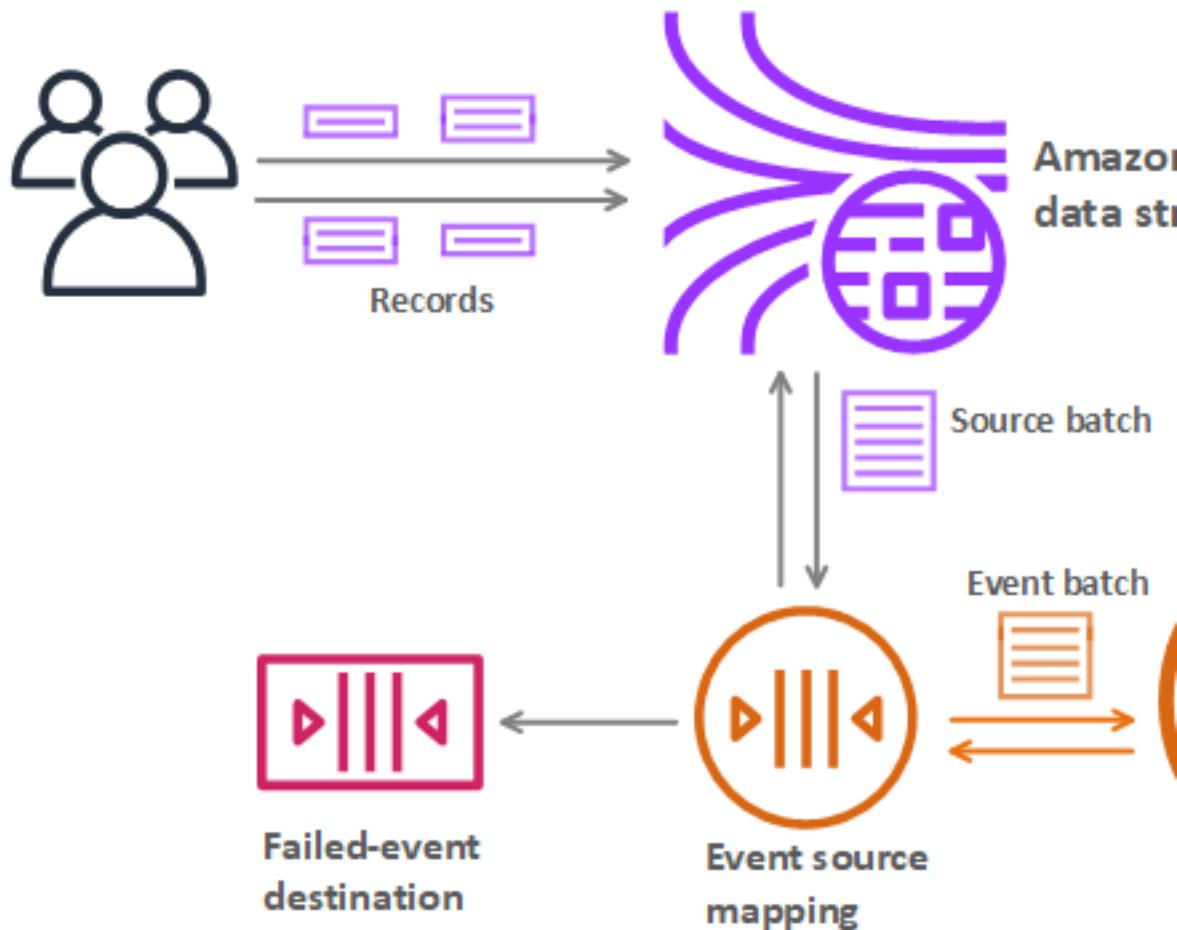
Para chamadas assíncronas, o Lambda manipula novas tentativas se a função retornar um erro ou for limitada. Para personalizar esse comportamento, você pode configurar parâmetros de tratamento de erros em uma função, versão ou alias. Também é possível configurar o Lambda para enviar eventos que falharam no processamento para uma fila de mensagens mortas ou para enviar um registro de qualquer chamada para um [destino](#) (p. 27).

Para obter mais informações, consulte [Invocação assíncrona](#) (p. 161).

Mapeamentos de origem do evento

Para processar itens de um stream ou fila, você pode criar um mapeamento de fonte de eventos. O mapeamento de uma fonte de eventos é um recurso no Lambda que lê itens de uma fila do Amazon Simple Queue Service (Amazon SQS), um Amazon Kinesis Stream ou Amazon DynamoDB Stream e os envia para sua função em lotes. Cada evento que seus processos de função pode conter centenas ou milhares de itens.

Event Source Mapping with Kinesis Stream



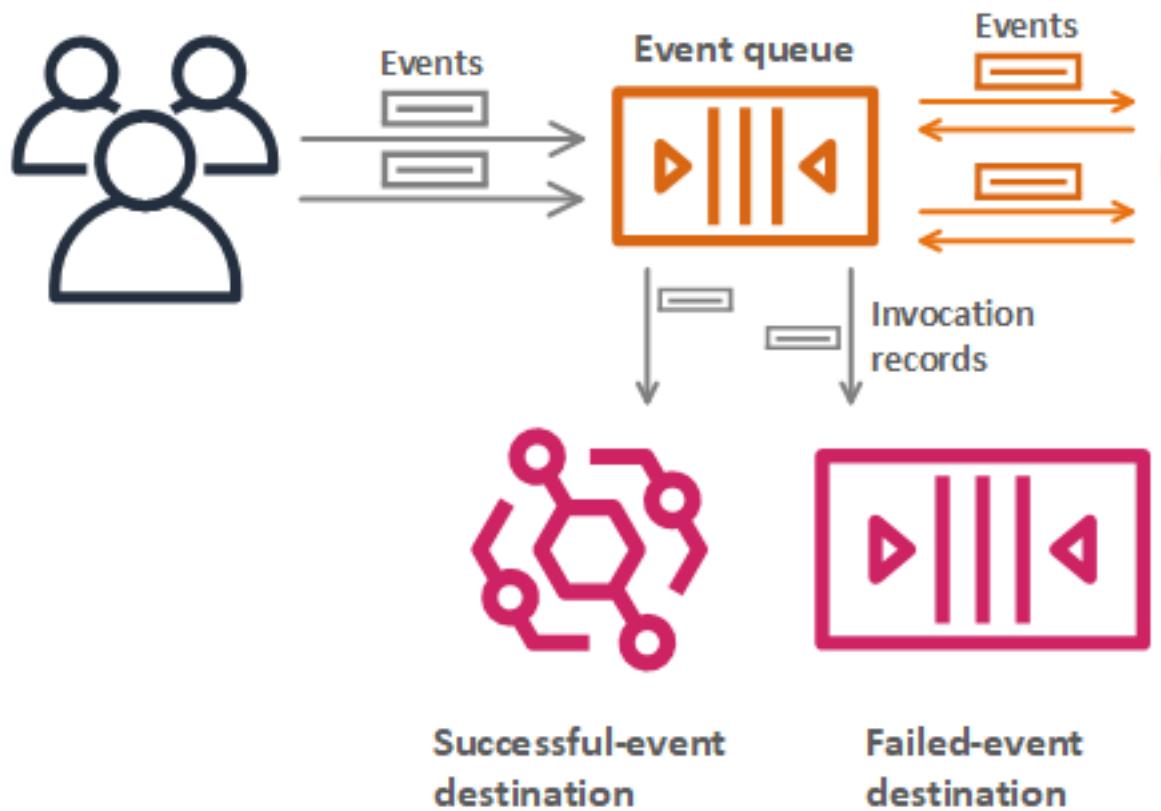
Os mapeamentos de fontes de eventos mantêm uma fila local de itens não processados e manipulam novas tentativas se a função retornar um erro ou for limitada. Você pode configurar um mapeamento de origem de evento para personalizar o tratamento de erros ou de comportamento de lote, e para enviar um registro de itens que falham no processamento para um destino.

Para obter mais informações, consulte [AWS Lambda Mapeamentos da fonte de eventos](#) (p. 170).

Destinations

Um destino é um recurso da AWS que recebe registros de chamada para uma função. Para [invocação assíncrona](#) (p. 25), você pode configurar o Lambda para enviar registros de chamada para uma fila, tópico, função ou barramento de eventos. Você pode configurar destinos separados para chamadas bem-sucedidas e eventos que falharam no processamento. O registro de chamada contém detalhes sobre o evento, a resposta da função e o motivo pelo qual o registro foi enviado.

Destinations for Asynchronous Invocation



Para [mapeamentos de fontes de eventos \(p. 26\)](#) que são lidos de transmissões, você pode configurar o Lambda para enviar para uma fila ou tópico um registro de lotes que falharam no processamento. Um registro de falhas de um mapeamento de origem de evento contém metadados sobre o lote e aponta para os itens no fluxo.

Para obter mais informações, consulte [Configurar destinos para invocação assíncrona \(p. 164\)](#) e as seções de tratamento de erros de [Usar o AWS Lambda com o Amazon DynamoDB \(p. 333\)](#) e [Usar o AWS Lambda com o Amazon Kinesis \(p. 387\)](#).

Esquemas de funções

Ao criar uma função no console do Lambda, escolha se deseja começar do zero, usar um esquema, usar uma [imagem de contêiner \(p. 37\)](#) ou implantar uma aplicação do [AWS Serverless Application Repository](#). Um esquema fornece um código de exemplo que mostra como usar o Lambda com um serviço da AWS ou uma aplicação de terceiros bem conhecida. Os esquemas incluem predefinições de código de exemplo e configuração de funções para tempos de execução de Node.js e Python.

Os esquemas são fornecidos para uso sob a [Licença de Software da Amazon](#). Eles estão disponíveis apenas no console do Lambda.

Ferramentas de teste e implantação

O Lambda é compatível com a implantação de código no estado em que encontra ou como [imagens de contêiner \(p. 37\)](#). Você pode usar um ecossistema avançado de ferramentas para desenvolver, criar e implantar suas funções do Lambda usando ferramentas de comunidade da AWS bem conhecidas, como a Command Line Interface (CLI – Interface de linha de comando) do Docker.

Para configurar a CLI do Docker, consulte [Obter Docker](#) no site do Docker Docs. Para obter uma introdução ao uso do Docker com oAWS, consulte[Conceitos básicos do Amazon ECR usando oAWS CLI](#)noAmazon Elastic Container Registry Guide.

Modelos de aplicativos

O console do Lambda pode ser usado para criar uma aplicação com um pipeline de entrega contínua. Os modelos de aplicações no console do Lambda incluem código para uma ou mais funções, um modelo de aplicação que define funções e recursos de suporte da AWS e um modelo de infraestrutura que define um pipeline do AWS CodePipeline. O pipeline tem estágios de compilação e implantação que são executados sempre que você enviar alterações para o repositório Git incluído.

Os modelos de aplicativos são fornecidos para serem utilizados sob a licença de [MIT No Attribution \(MIT sem atribuição\)](#) . Eles estão disponíveis apenas no console do Lambda.

Para obter mais informações, consulte [Criar uma aplicação com entrega contínua no console do Lambda \(p. 196\)](#).

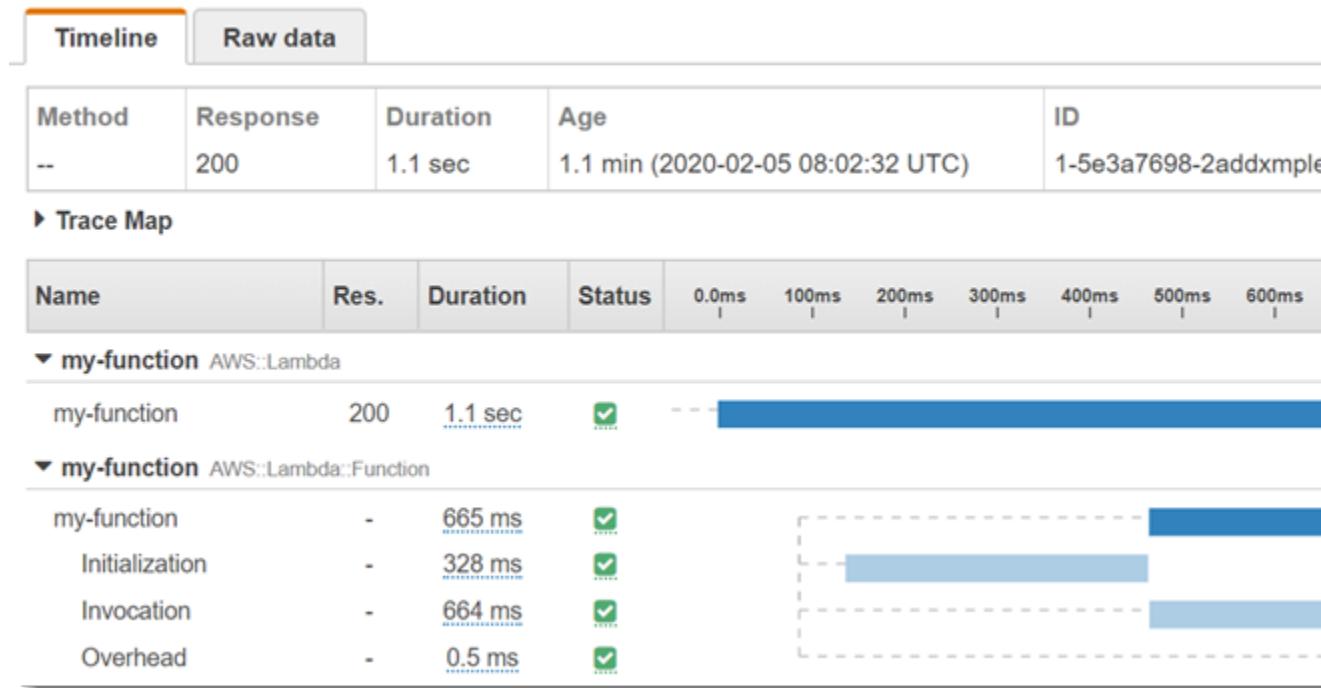
Modelo de programação do Lambda

O Lambda oferece um modelo de programação comum a todos os tempos de execução. O modelo de programação define a interface entre seu código e o sistema Lambda. Você diz ao Lambda o ponto de entrada para sua função definindo um manipulador na configuração da função. O tempo de execução transmite ao manipulador os objetos que contêm o evento de invocação e o contexto, como o nome da função e o ID da solicitação.

Quando o handler termina de processar o primeiro evento, o tempo de execução o envia outro. A classe da função permanece na memória, de forma que é possível reutilizar clientes e variáveis que são declarados fora do método do handler no código de inicialização. Para economizar tempo de processamento em eventos subsequentes, crie recursos reutilizáveis como clientes do AWS SDK durante a inicialização. Uma vez inicializada, cada instância da função pode processar milhares de solicitações.

Quando o [rastreamento do AWS X-Ray \(p. 477\)](#) está habilitado, o tempo de execução registra subsegmentos separados para inicialização e execução.

Traces > Details



A função também tem acesso ao armazenamento local no diretório /tmp. As instâncias da sua função que estão atendendo solicitações permanecem ativas por algumas horas antes de serem recicladas.

O tempo de execução captura a saída do registro em log da função e a envia para o Amazon CloudWatch Logs. Além de registrar em log a saída da função, o tempo de execução também registra em log as entradas quando a invocação da função é iniciada e termina. Isso inclui um log de relatório com o ID da solicitação, a duração faturada, a duração da inicialização e outros detalhes. Se a função lançar um erro, o tempo de execução retornará esse erro para o chamador.

Note

O registro em log está sujeito a[Cotas CloudWatch Logs](#). Os dados de log podem ser perdidos devido a limitação ou, em certos casos, quando uma instância da sua função é interrompida.

Para obter uma introdução prática do modelo de programação na linguagem de programação de sua preferência, consulte os capítulos a seguir.

- [Criar funções do Lambda com Node.js \(p. 511\)](#)
- [Criar funções do Lambda com Python \(p. 538\)](#)
- [Construir funções do Lambda com Ruby \(p. 567\)](#)
- [Construir funções do Lambda com Java \(p. 592\)](#)
- [Criar funções do Lambda com Go \(p. 633\)](#)
- [Construir funções do Lambda com C# \(p. 661\)](#)
- [Construir funções do Lambda com o PowerShell \(p. 692\)](#)

O Lambda dimensiona a função executando instâncias adicionais dela à medida que a demanda aumenta, e interrompendo instâncias à medida que a demanda diminui. Salvo indicação em contrário, as solicitações de entrada poderiam ser processadas fora de ordem ou simultaneamente. Armazene o estado do aplicativo em outros serviços e não dependa de que instâncias da função sejam de longa duração. Use o armazenamento local e os objetos do nível de classe para aumentar a performance, mas mantenha no mínimo o tamanho do pacote de implantação e da quantidade de dados transferidos para o ambiente de execução.

Escalabilidade da função do Lambda

Na primeira vez que você invoca a função, o AWS Lambda cria uma instância da função e executa seu método de handler para processar o evento. Quando a função retorna uma resposta, ela permanece ativa e aguarda para processar eventos adicionais. Se você invocar a função novamente enquanto o primeiro evento está sendo processado, o Lambda inicializará outra instância e a função processará os dois eventos simultaneamente. À medida que mais eventos são adicionados, o Lambda os direciona para instâncias disponíveis e cria novas instâncias conforme necessário. Quando o número de solicitações diminui, o Lambda interrompe instâncias não utilizadas para liberar capacidade de dimensionamento para outras funções.

A simultaneidade da função é o número de instâncias que atendem solicitações em um determinado momento. Para uma intermitência inicial de tráfego, a simultaneidade cumulativa da função em uma Região pode atingir um nível inicial de 500 a 3.000, que varia por Região. Observe que a cota de simultaneidade de intermitência não é por função; ela se aplica a todas as suas funções na região.

Cotas de simultaneidade de intermitência

- 3000—Oeste dos EUA (Oregon), Leste dos EUA (Norte da Virgínia), Europa (Irlanda)
- 1000—Ásia-Pacífico (Tóquio), Europa (Frankfurt), Leste dos EUA (Ohio)
- 500 – outras regiões.

Após a intermitência inicial, a simultaneidade da função pode ser dimensionada por mais 500 instâncias por minuto. Isso continua até que o número de instâncias seja suficiente para atender a todas as solicitações ou até que um limite de simultaneidade seja atingido. Quando as solicitações chegam mais rápido do que sua função pode escalar, ou quando sua função é a simultaneidade máxima, solicitações adicionais falham com um erro de limitação (código de status 429).

O exemplo a seguir mostra uma função processando um pico no tráfego. À medida que as invocações aumentam exponencialmente, a função aumenta. Ela inicializa uma nova instância para qualquer solicitação que não possa ser roteada para uma instância disponível. Quando o limite de simultaneidade de intermitência é atingido, a função começa a escalar linearmente. Se essa simultaneidade não for suficiente para atender a todas as solicitações, as solicitações adicionais serão limitadas e deverão ser repetidas.

Function Scaling with Concurrency Limit

Concurrency
limit

Burst limit

Legend

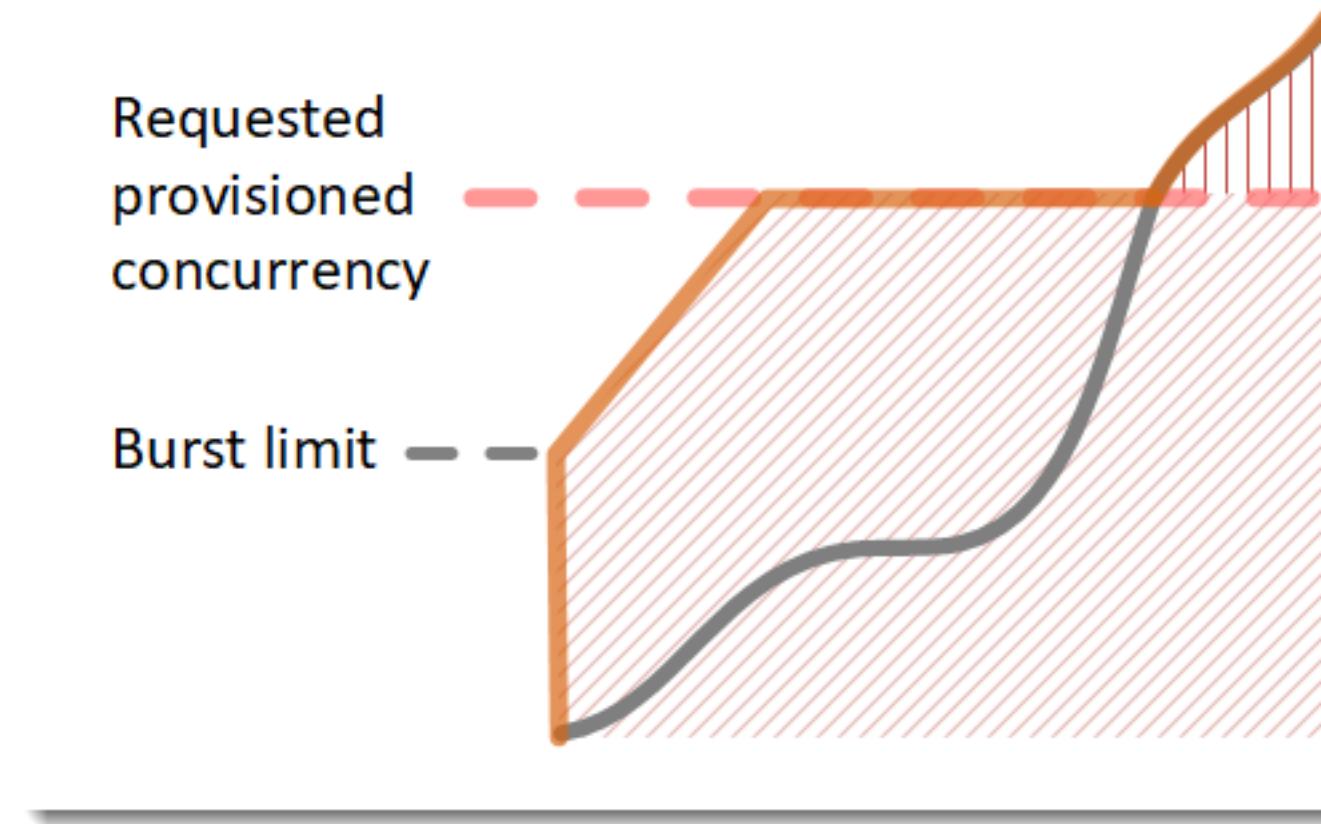
- Instâncias de função
- Solicitações abertas
- Possibilidade de limitação

A função continua a escalar até que o limite de simultaneidade da conta para a região da função seja atingido. A função alcança a demanda, as solicitações diminuem e, depois de ficarem ociosas por algum tempo, as instâncias não utilizadas da função são interrompidas. As instâncias não utilizadas são congeladas enquanto aguardam solicitações e não incorrem em nenhuma cobrança.

O limite de simultaneidade regional começa em 1.000. O limite pode ser aumentado enviando uma solicitação no [console do Centro de Suporte](#). Para alocar capacidade por função, é possível configurar funções com [simultaneidade reservada \(p. 113\)](#). A simultaneidade reservada cria um grupo que só poderá ser usado por sua função e também impede que sua função use simultaneidade não reservada.

Quando sua função é escalada, a primeira solicitação que cada instância atende é afetada pelo tempo necessário para carregar e inicializar seu código. Se o [código de inicialização \(p. 30\)](#) demorar muito, o impacto na latência média e de percentil pode ser significativo. Para permitir que sua função seja dimensionada sem flutuações na latência, use [simultaneidade provisionada \(p. 113\)](#). O exemplo a seguir mostra uma função com simultaneidade provisionada processando um pico no tráfego.

Function Scaling with Provisioned Concurrency



Legend

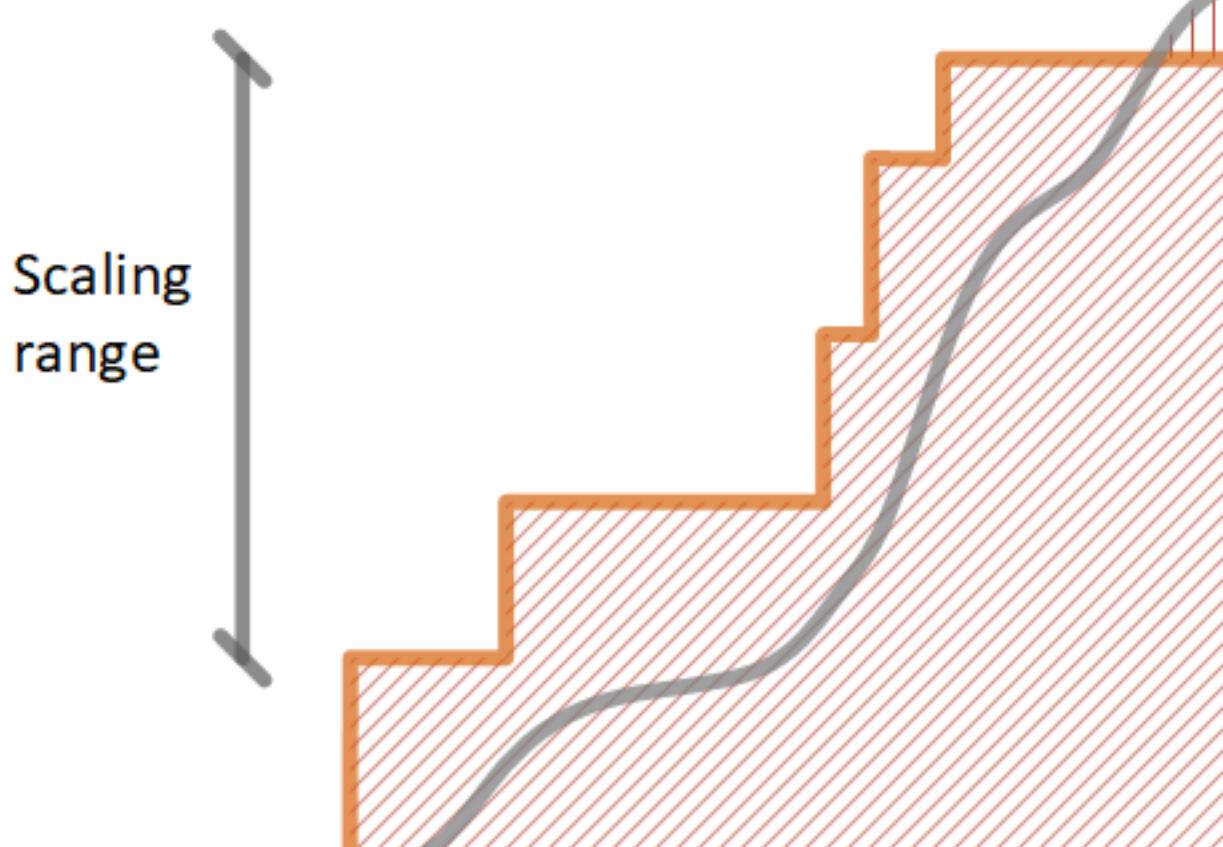
- Instâncias de função
- Solicitações abertas
- Simultaneidade provisionada
- Simultaneidade padrão

Quando você alocar simultaneidade provisionada, a função estará pronta para atender a uma intermitência de solicitações recebidas com latência muito baixa. Quando toda a simultaneidade provisionada está em uso, a função escala normalmente para lidar com todas as solicitações adicionais.

O Application Auto Scaling leva isso um passo adiante, fornecendo escalabilidade automática para a simultaneidade provisionada. Com o Application Auto Scaling, é possível uma política de escalabilidade de rastreamento de destino que ajusta automaticamente os níveis de simultaneidade provisionada, com base na métrica de utilização que o Lambda emite. [Use a API do Application Auto Scaling \(p. 119\)](#) para registrar um alias como um destino escalável e criar uma política de escalabilidade.

No exemplo a seguir, uma função é dimensionada entre uma quantidade mínima e máxima de simultaneidade provisionada com base na utilização. Quando o número de solicitações abertas aumenta, o Application Auto Scaling aumenta a simultaneidade provisionada em grandes etapas até atingir o máximo configurado. A função continua a ser dimensionada em simultaneidade padrão até que a utilização comece a cair. Quando a utilização é consistentemente baixa, o Application Auto Scaling diminui a simultaneidade provisionada em etapas periódicas menores.

Autoscaling with Provisioned Concurrency



Legend

- Instâncias de função
- Solicitações abertas
- Simultaneidade provisionada
- Simultaneidade padrão

Ao invocar a função de forma assíncrona, com um mapeamento da origem do evento ou com outro serviço da AWS, o comportamento de escalabilidade varia. Por exemplo, os mapeamentos de origem do evento que leem de um stream são limitados pelo número de fragmentos no stream. A capacidade de escalabilidade não usada por uma origem de evento está disponível para uso por outros clientes e origens de evento. Para obter mais informações, consulte os tópicos a seguir.

- [Invocação assíncrona \(p. 161\)](#)
- [AWS LambdaMapeamentos da fonte de eventos do \(p. 170\)](#)
- [Lidar com erros e novas tentativas automáticas no AWS Lambda \(p. 176\)](#)
- [Usar o AWS Lambda com outros serviços \(p. 277\)](#)

É possível monitorar os níveis de simultaneidade em sua usando as seguintes métricas:

Métricas de simultaneidade

- `ConcurrentExecutions`
- `UnreservedConcurrentExecutions`
- `ProvisionedConcurrentExecutions`
- `ProvisionedConcurrencyInvocations`
- `ProvisionedConcurrencySpilloverInvocations`
- `ProvisionedConcurrencyUtilization`

Para obter mais informações, consulte [Trabalhar com métricas de função do AWS Lambda \(p. 717\)](#).

Pacotes de implantação do Lambda

O código da função do AWS Lambda consiste em scripts ou programas compilados e as dependências deles. Você usa um pacote de implantação para implantar seu código de função no Lambda. O Lambda é compatível com dois tipos de pacotes de implantação: imagens de contêiner e arquivos .zip.

Tópicos

- [Imagens de contêiner \(p. 37\)](#)
- [Arquivos .zip \(p. 37\)](#)
- [Layers \(p. 38\)](#)
- [Usando outros serviços do AWS para criar um pacote de implantação \(p. 38\)](#)

Imagens de contêiner

Uma imagem de contêiner inclui o sistema operacional de base, o tempo de execução, as extensões do Lambda, o código da aplicação e dependências dele. Você também pode adicionar dados estáticos, como modelos de machine learning, à imagem.

O Lambda oferece um conjunto de imagens de base de código aberto que você pode usar para criar a imagem de contêiner. Para criar e testar imagens de contêiner, você pode usar a interface da linha de comando AWS Serverless Application Model (AWS SAM) (CLI) ou ferramentas nativas de contêiner, como a CLI do Docker.

Você faz o upload de suas imagens de contêiner no Amazon Elastic Container Registry (Amazon ECR), um serviço de registro de imagem do recipiente. Para implantar a imagem em sua função, especifique o URL da imagem do Amazon ECR usando o console do Lambda, a API do Lambda, as ferramentas da linha de comando ou os AWS SDKs.

Para obter mais informações sobre imagens de contêiner do Lambda, consulte [Usar imagens de contêiner com o Lambda \(p. 266\)](#).

Arquivos .zip

Um arquivo .zip inclui o código da aplicação e as dependências dele. Quando você cria funções usando o console do Lambda ou um toolkit, o Lambda cria automaticamente um arquivo .zip do seu código.

Quando você cria funções com a API do Lambda, ferramentas da linha de comando ou AWS SDKs, é necessário criar o pacote de implantação. Você também deve criar um pacote de implantação se sua função usar uma linguagem compilada ou para adicionar dependências a ela. Para implantar o código da sua função, faça upload do pacote de implantação do Amazon Simple Storage Service (Amazon S3) ou de sua máquina local.

Você pode fazer o upload de um arquivo .zip como seu pacote de implantação usando o console do Lambda, AWS Command Line Interface (AWS CLI) ou para um bucket do Amazon Simple Storage Service (Amazon S3).

Usar o console do Lambda

As etapas a seguir demonstram como fazer o upload de um arquivo .zip como seu pacote de implantação usando o console do Lambda.

Para fazer o upload de um arquivo .zip no console do Lambda

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.

2. Selecione uma função.
3. No Código-fonte, selecione Fazer upload de, em seguida.zip.
4. Selecione Upload para escolher seu arquivo .zip local.
5. Escolha Save (Salvar).

Usar a AWS CLI

Você pode fazer o upload de um arquivo .zip como seu pacote de implantação usando o AWS Command Line Interface (AWS CLI). Para obter instruções específicas de linguagem, consulte os tópicos a seguir.

- [Implantar funções do Lambda em Node.js com arquivos .zip \(p. 517\)](#)
- [Implantar funções do Lambda em Python com arquivos .zip \(p. 543\)](#)
- [Implantar funções do Lambda em Ruby com arquivos .zip \(p. 571\)](#)
- [Implantar funções do Lambda em Java com arquivos .zip ou JAR \(p. 599\)](#)
- [Implantar funções do Lambda em Go com arquivos .zip \(p. 640\)](#)
- [Implantar funções do Lambda em C# com arquivos .zip \(p. 668\)](#)
- [Implantar funções do Lambda para PowerShell com arquivos .zip \(p. 694\)](#)

Uso do Amazon S3

Você pode fazer o upload de um arquivo .zip como seu pacote de implantação usando o Amazon Simple Storage Service (Amazon S3). Para obter mais informações, consulte [the section called “Usando outros serviços do AWS”](#).

Layers

Se você implantar seu código de função usando um arquivo .zip, poderá usar camadas do Lambda como mecanismo de distribuição para bibliotecas, tempos de execução personalizados e outras dependências de função. As camadas permitem que você gerencie seu código de função no desenvolvimento de modo independente do código inalterável e dos recursos que ele usa. Você pode configurar a função para usar camadas que você criar, camadas fornecidas pela AWS ou camadas de outros clientes da AWS.

Não use camadas com imagens de contêiner. Em vez disso, você empacota seu tempo de execução preferido, bibliotecas e outras dependências na imagem do contêiner ao criar a imagem.

Para obter mais informações sobre camadas, consulte [Criar e compartilhar camadas do Lambda \(p. 85\)](#).

Usando outros serviços do AWS para criar um pacote de implantação

A seção a seguir descreve outros serviços do AWS que você pode usar para empacotar dependências para a função do seu Lambda.

Pacotes de implantação com bibliotecas C ou C++

Se o pacote de implantação contiver bibliotecas nativas, você poderá criar o pacote de implantação com o AWS Serverless Application Model (AWS SAM). É possível usar o comando AWS SAM da CLI do `sam build` com `--use-container` para criar seu pacote de implantação. Essa opção cria um pacote de implantação dentro de uma imagem do Docker compatível com o ambiente de execução do Lambda.

Para obter mais informações, consulte [sam build](#) no Guia do desenvolvedor do AWS Serverless Application Model.

Pacotes de implantação com mais de 50 MB

Se o pacote de implantação for maior que 50 MB, recomendamos fazer o upload do código de função e das dependências para um bucket do Amazon S3.

Você pode criar um pacote de implantação e fazer upload do arquivo .zip para seu bucket do Amazon S3 na região da AWS em que você quer criar uma função do Lambda. Ao criar sua função do Lambda, especifique o nome do bucket do S3 e o nome da chave do objeto no console do Lambda ou usando o AWS CLI.

Para criar um bucket usando o console do Amazon S3, consulte [Como criar um bucket do S3?](#) no Guia do usuário do console do Amazon Simple Storage Service.

Console do Lambda

É possível usar o console do Lambda para configurar aplicativos, funções, configurações de assinatura de código e camadas.

Tópicos

- [Applications \(p. 40\)](#)
- [Functions \(p. 40\)](#)
- [Assinatura de código \(p. 40\)](#)
- [Layers \(p. 40\)](#)
- [Editar código usando o editor do console do \(p. 40\)](#)

Applications

A página [Aplicativos \(p. 192\)](#) mostra uma lista de aplicativos que foram implantados usando o AWS CloudFormation ou outras ferramentas, incluindo o AWS Serverless Application Model. Aplique um filtro para encontrar aplicativos com base em palavras-chave.

Functions

A página de funções mostra uma lista de funções definidas para sua conta nesta região. O fluxo inicial do console para criar uma função depende se a função usa um [Imagem de contêiner \(p. 90\)](#) ou [Arquivo .zip \(p. 82\)](#). Para o pacote de implantação. Muitos dos [Tarefas de configuração \(p. 95\)](#) são comuns a ambos os tipos de função.

O console fornece um [Editor de código \(p. 40\)](#) para a sua conveniência.

Assinatura de código

É possível anexar um [Assinatura de código \(p. 144\)](#). Configuração de uma função. Com a assinatura de código, você pode garantir que o código foi assinado por uma fonte aprovada e não foi alterado desde a assinatura, e que a assinatura de código não expirou ou foi revogada.

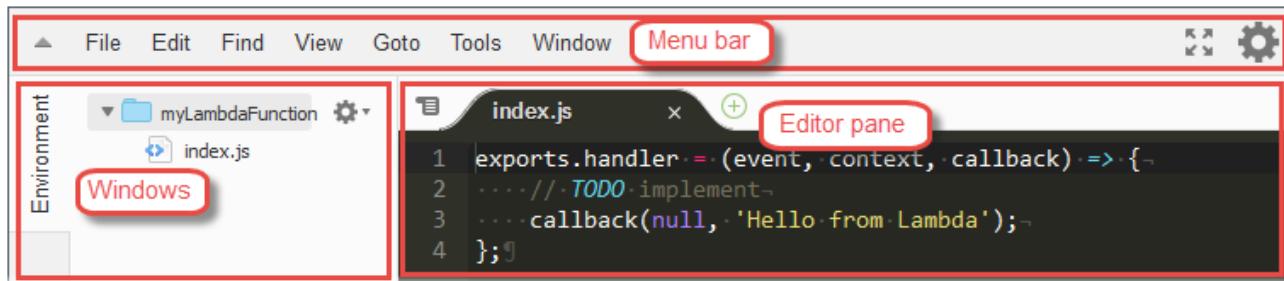
Layers

Crie [camadas \(p. 85\)](#) para separar o código de função do arquivo .zip de suas dependências. Camada é um arquivo ZIP que contém bibliotecas, um tempo de execução personalizado ou outras dependências. Com camadas, você pode usar as bibliotecas na sua função sem a necessidade de incluí-las em seu pacote de implantação.

Editar código usando o editor do console do

Use o editor de código no console do AWS Lambda para escrever, testar e visualizar os resultados de execução do seu código de função do Lambda. O editor de código é compatível com linguagens que não exigem compilação, como Node.js e Python. O editor de código suporta apenas pacotes de implementação de arquivos.zip e o tamanho do pacote de implementação deve ser inferior a 3 MB.

O editor de código inclui a barra de menu, as janelas e o painel do editor.



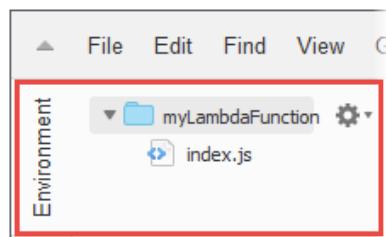
Para obter uma lista do que os comandos fazem, consulte a [Referência de comandos do menu](#) no Guia do usuário do AWS Cloud9. Alguns dos comandos listados nessa referência não estão disponíveis no editor de código.

Tópicos

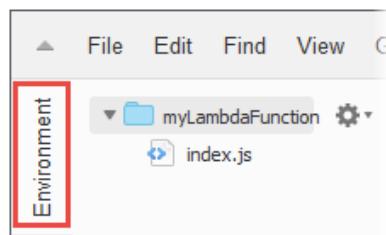
- [Trabalhar com arquivos e pastas \(p. 41\)](#)
- [Trabalhar com códigos \(p. 43\)](#)
- [Trabalhar no modo de tela cheia \(p. 46\)](#)
- [Trabalhar com preferências \(p. 46\)](#)

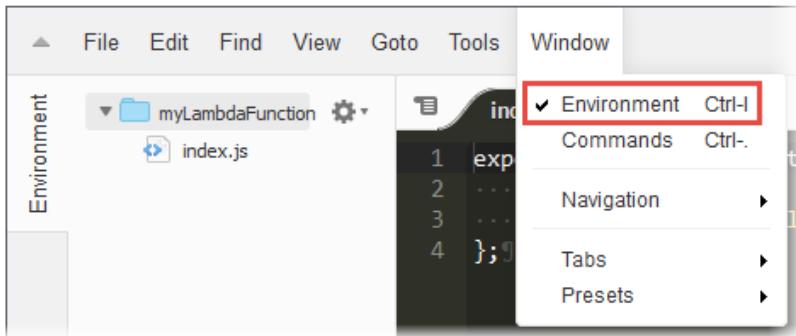
Trabalhar com arquivos e pastas

Use a janela Environment (Ambiente) no editor de código para criar, abrir e gerenciar arquivos para sua função.



Para exibir ou ocultar a janela Environment (Ambiente), escolha o botão Environment (Ambiente). Se o botão Environment (Ambiente) não estiver visível, escolha Window, Environment (Janela, ambiente) na barra de menus.



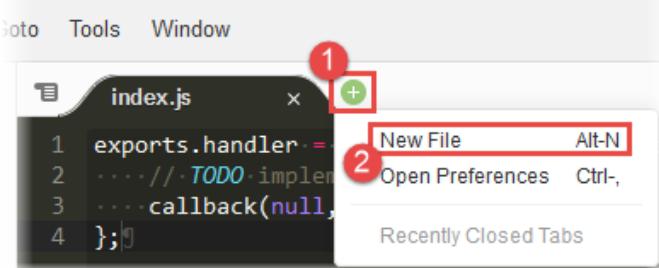


Para abrir um único arquivo e mostrar seu conteúdo no painel do editor, clique duas vezes no arquivo na janela Environment (Ambiente).

Para abrir vários arquivos e mostrar seus conteúdos no painel do editor, escolha os arquivos na janela Environment (Ambiente). Clique com o botão direito do mouse na seleção e escolha Open (Abrir).

Para criar um novo arquivo, faça o seguinte:

- Na janela Environment (Ambiente), clique com o botão direito do mouse na pasta para a qual você deseja enviar o novo arquivo e escolha New File (Novo arquivo). Digite o nome e a extensão do arquivo e, em seguida, pressione Enter.
- Escolha File, New File (Arquivo, novo arquivo) na barra de menus. Quando estiver pronto para salvar o arquivo, escolha File, Save ou File, Save As na barra de menus. Use a caixa de diálogo Save As exibida para nomear o arquivo e escolha onde salvá-lo.
- Na barra de botões da guia no painel do editor, escolha o botão + e escolha New File (Novo arquivo). Quando estiver pronto para salvar o arquivo, escolha File, Save ou File, Save As na barra de menus. Use a caixa de diálogo Save As exibida para nomear o arquivo e escolha onde salvá-lo.



Para criar uma nova pasta, clique com o botão direito do mouse na pasta na janela Environment onde você deseja que a nova pasta seja criada e escolha New Folder. Digite o nome da pasta e pressione Enter.

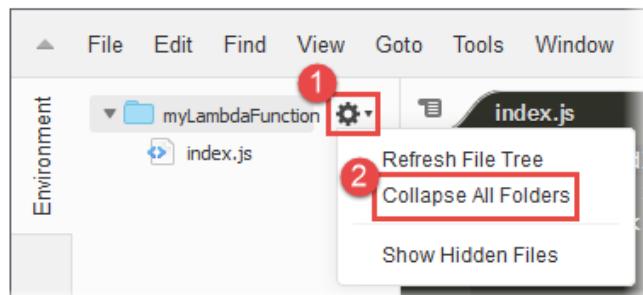
Para salvar um arquivo, com o arquivo aberto e seu conteúdo visível no painel do editor, escolha File, Save na barra de menus.

Para renomear um arquivo ou uma pasta, clique com o botão direito do mouse no arquivo ou na pasta na janela Environment. Digite o nome de substituição e pressione Enter.

Para excluir arquivos ou pastas, selecione os arquivos ou as pastas na janela Environment. Clique com o botão direito do mouse na seleção e escolha Delete. Confirme a exclusão escolhendo Yes (para seleção única) ou Yes to All.

Para recortar, copiar, colar ou duplicar arquivos ou pastas, selecione os arquivos ou as pastas na janela Environment. Clique com o botão direito do mouse na seleção e escolha Recortar, Copiar, Colar ou Duplicar, respectivamente.

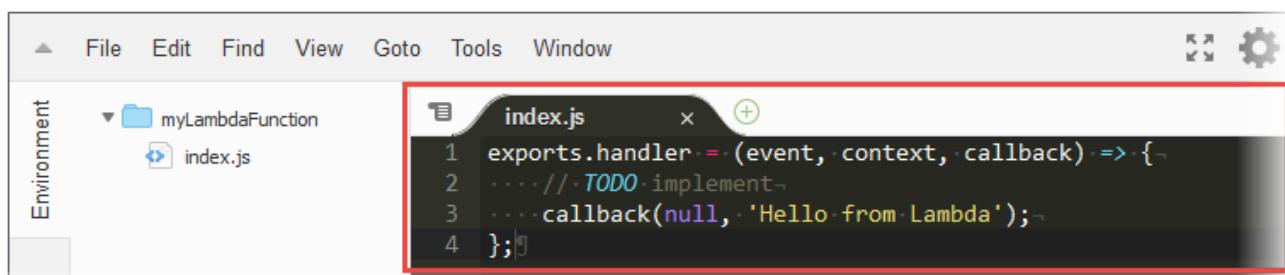
Para recolher pastas, escolha o ícone de engrenagem na janela Environment e, em seguida, Collapse All Folders.



Para exibir ou ocultar arquivos, escolha o ícone de engrenagem na janela Environment e, em seguida, Show Hidden Files.

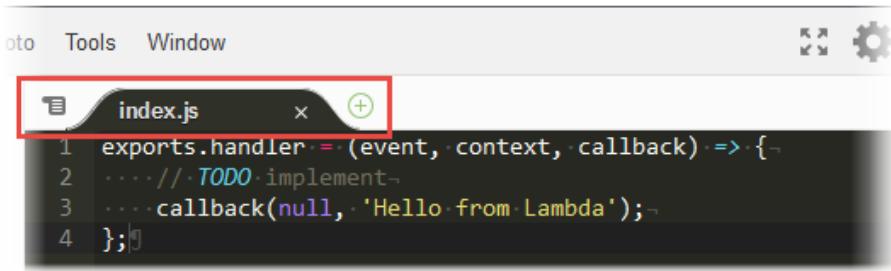
Trabalhar com códigos

Use o painel no editor de código para visualizar e escrever o código.



Trabalhar com botões da guia

Use a barra de botões da guia para selecionar, visualizar e criar arquivos.



Para exibir o conteúdo de um arquivo aberto, faça o seguinte:

- Escolha a guia do arquivo.
- Escolha o botão de menu suspenso na barra de botões da guia e, em seguida, escolha o nome do arquivo.



Para fechar um arquivo aberto, faça o seguinte:

- Escolha o ícone X na guia do arquivo.
- Escolha a guia do arquivo. Escolha o botão de menu suspenso na barra de botões da guia e, em seguida, escolha Close Pane.

Para fechar vários arquivos abertos, escolha o menu suspenso na barra de botões da guia e escolha Close All Tabs in All Panes ou Close All But Current Tab, conforme necessário.

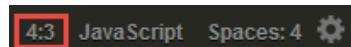
Para criar um novo arquivo, escolha o botão + na barra de botões da guia e, em seguida, escolha New File. Quando estiver pronto para salvar o arquivo, escolha File, Save ou File, Save As na barra de menus. Use a caixa de diálogo Save As exibida para nomear o arquivo e escolha onde salvá-lo.

Trabalhar com a barra de status

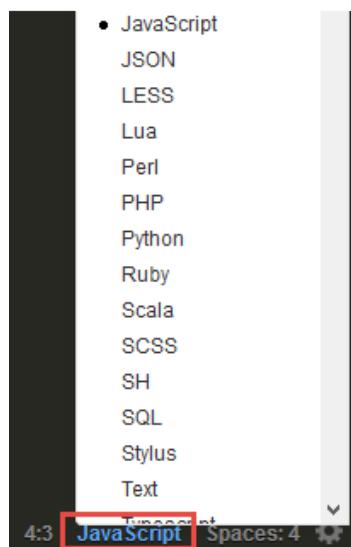
Use a barra de status para mudar rapidamente para uma linha no arquivo ativo e alterar a forma como o código é exibido.



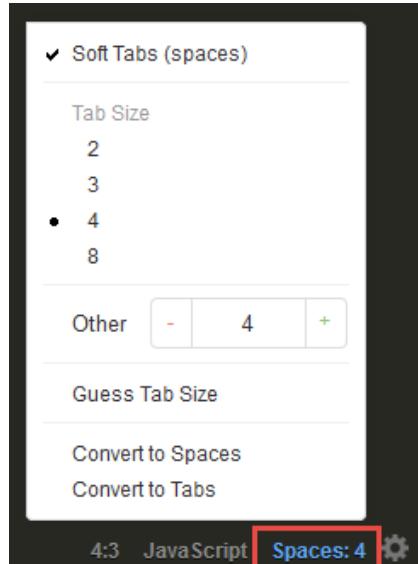
Para mudar rapidamente para uma linha no arquivo ativo, escolha o seletor de linha, digite o número da linha a seguir e pressione Enter.



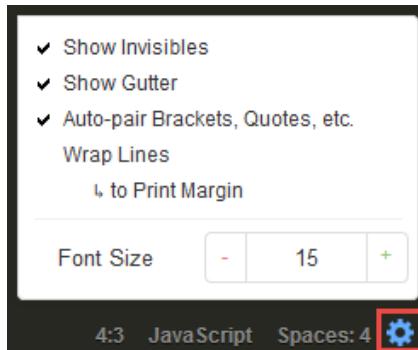
Para alterar o esquema de cores do código no arquivo ativo, escolha o seletor do esquema de cores do código e selecione o novo esquema de cores do código.



Para escolher se tabulações suaves ou espaços serão usados no arquivo ativo, além do tamanho da tabulação ou se deve haver conversão de espaços ou tabulações, escolha o seletor de espaços e tabulações e selecione as novas configurações.



Para definir se todos os arquivos devem mostrar ou ocultar caracteres invisíveis ou gutter, inserir automaticamente pares de colchetes ou aspas, fazer a quebra de linhas ou alterar o tamanho da fonte, escolha o ícone de engrenagem e selecione as novas configurações.



Trabalhar no modo de tela cheia

Você pode expandir o editor de código e ter mais espaço para trabalhar com seu código.

Para expandir o editor de código nas margens da janela do navegador da web, escolha o botão Toggle fullscreen na barra de menus.



Para diminuir o editor de código até o tamanho original, escolha novamente o botão Toggle fullscreen.

No modo de tela cheia, opções adicionais são exibidas na barra de menus: Save e Test. Save salva o código de função. Test ou Configure Events permitem que você crie ou edite os eventos de teste da função.

Trabalhar com preferências

Você pode alterar várias configurações do editor de código, por exemplo, quais dicas de codificação e avisos serão exibidos, comportamentos de code folding, comportamentos de autopreenchimento de código e muito mais.

Para alterar as configurações do editor de código, escolha o ícone de engrenagem Preferences na barra de menus.



Para obter uma lista das configurações, consulte as seguintes referências no Guia do usuário do AWS Cloud9.

- [Alterações que podem ser feitas nas configurações do projeto](#)
- [Alterações que podem ser feitas nas configurações do usuário](#)

Algumas das configurações listadas nessas referências não estão disponíveis no editor de código.

Como usar o Lambda com o AWS CLI

Você pode usar a AWS Command Line Interface para gerenciar funções e outros recursos do AWS Lambda. A AWS CLI usa o AWS SDK for Python (Boto) para interagir com a API do Lambda. Você pode usá-la para aprender sobre a API e aplicar esse conhecimento na criação de aplicações que usam o Lambda com o AWS SDK.

Neste tutorial, você gerencia e invoca as funções do Lambda com a AWS CLI. Para obter mais informações, consulte [O que é o AWS CLI?](#) no Manual do usuário do AWS Command Line Interface.

Prerequisites

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Se ainda não tiver visto as instruções, acesse-as em [the section called “Criar uma função do” \(p. 9\).](#)

Para concluir as etapas a seguir, você precisa de um terminal de linha de comando ou de um shell para executar os comandos. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

Você deve ver a saída a seguir:

```
aws-cli/2.0.57 Python/3.7.4 Darwin/19.6.0 exe/x86_64
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

Este tutorial usa a AWS Command Line Interface (AWS CLI) para chamar operações de API de serviço. Para instalar a AWS CLI, consulte [Installing the AWS CLI](#) no Manual do usuário do AWS Command Line Interface.

Criar a função de execução

Crie a [função de execução \(p. 57\)](#) que dá à sua função permissão para acessar recursos do AWS. Para criar uma função de execução com a AWS CLI, use o comando `create-role`.

No exemplo a seguir, você especifica a política de confiança em linha. Os requisitos para escapar de aspas na string JSON variam dependendo do seu 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"}]}
```

Também é possível definir a [política de confiança](#) para a função usando um arquivo JSON. No exemplo a seguir, `trust-policy.json` é um arquivo no diretório atual. Esta política de confiança permite que o Lambda use as permissões da função fornecendo o principal de serviço `lambda.amazonaws.com` permissão para chamar o método AWS Security Token Service `AssumeRole`.

Example `trust-policy.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
```

Você deve ver a saída a seguir:

```
{
  "Role": {
    "Path": "/",
    "RoleName": "lambda-ex",
    "RoleId": "AROAQFOXmpl6TZ6ITKWND",
    "Arn": "arn:aws:iam::123456789012:role/lambda-ex",
    "CreateDate": "2020-01-17T23:19:12Z",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "lambda.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
}
```

Para adicionar permissões à função, use o comando `attach-policy-to-role`. Inicie adicionando a `AWSLambdaBasicExecutionRole` política gerenciada pela.

```
aws iam attach-role-policy --role-name lambda-ex --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

A política `AWSLambdaBasicExecutionRole` tem as permissões necessárias para a função gravar logs no CloudWatch Logs.

Criar a função

O exemplo a seguir registra em log os valores das variáveis de ambiente e o objeto do evento.

Example index.js

```
exports.handler = async function(event, context) {
  console.log("ENVIRONMENT VARIABLES\n" + JSON.stringify(process.env, null, 2))
  console.log("EVENT\n" + JSON.stringify(event, null, 2))
  return context.logStreamName
```

```
}
```

Para criar a função

1. Copie o código de amostra em um arquivo chamado `index.js`.
2. Crie um pacote de implantação.

```
zip function.zip index.js
```

3. Crie uma função do Lambda com o comando `create-function`. Substitua o texto realçado no ARN da função pelo ID da conta.

```
aws lambda create-function --function-name my-function \
--zip-file file://function.zip --handler index.handler --runtime nodejs12.x \
--role arn:aws:iam::123456789012:role/lambda-ex
```

Você deve ver a saída a seguir:

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
  "Runtime": "nodejs12.x",
  "Role": "arn:aws:iam::123456789012:role/lambda-ex",
  "Handler": "index.handler",
  "CodeSha256": "FpFMvUhayLkOoVBpNuNiIVML/tuGv2iJQ7t0yWVTU8c=",
  "Version": "$LATEST",
  "TracingConfig": {
    "Mode": "PassThrough"
  },
  "RevisionId": "88ebe1e1-bfdf-4dc3-84de-3017268fa1ff",
  ...
}
```

Para obter logs para uma invocação a partir da linha de comando, use a opção `--log-type`. A resposta inclui um campo `LogResult` que contém até 4 KB de logs codificados em base64 da invocação.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Você deve ver a saída a seguir:

```
{
  "StatusCode": 200,
  "LogResult": "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRIOC1mMTU0LTEzZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
  "ExecutedVersion": "$LATEST"
}
```

Você pode usar o utilitário `base64` para decodificar os logs.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
```

Você deve ver a saída a seguir:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e [Ubuntu no Windows](#). Para macOS, o comando é `base64 -D`.

Para obter eventos de log completos a partir da linha de comando, você pode incluir o nome do fluxo de logs na saída de sua função, conforme mostrado no exemplo anterior. O script de exemplo a seguir invoca uma função chamada `my-function` e faz o download dos últimos 5 eventos de log.

Example Script get-logs.sh

Este exemplo requer que `my-function` retorne um ID de fluxo de log.

```
#!/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 $(cat
out) --limit 5
```

O script usa `sed` para remover aspas do arquivo de saída e dorme por 15 segundos para permitir que os logs estejam disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

```
./get-logs.sh
```

Você deve ver a saída a seguir:

```
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version: $LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf \tINFO\tENVIRONMENT VARIABLES\r{\r\t\"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\", \r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf \tINFO\tEVENT\r{\r\t\"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"  
}
```

Listar as funções do Lambda na conta

Execute o comando da AWS CLI `list-functions` a seguir para recuperar uma lista de funções que você criou.

```
aws lambda list-functions --max-items 10
```

Você deve ver a saída a seguir:

```
{  
    "Functions": [  
        {  
            "FunctionName": "cli",  
            "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",  
            "Runtime": "nodejs12.x",  
            "Role": "arn:aws:iam::123456789012:role/lambda-ex",  
            "Handler": "index.handler",  
            ...  
        },  
        {  
            "FunctionName": "random-error",  
            "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:random-error",  
            "Runtime": "nodejs12.x",  
            "Role": "arn:aws:iam::123456789012:role/lambda-role",  
            "Handler": "index.handler",  
            ...  
        },  
        ...  
    ],  
    "NextToken": "eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxMH0="  
}
```

Em resposta, o Lambda retorna uma lista de até 10 funções. Se há mais funções que você pode recuperar, o parâmetro `NextToken` fornece um marcador que você pode usar na próxima solicitação `list-functions`. O comando `list-functions` da AWS CLI a seguir é um exemplo que mostra o parâmetro `--starting-token`.

```
aws lambda list-functions --max-items 10 --starting-  
token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxMH0=
```

Recuperar uma função do Lambda

O comando `get-function` da CLI do Lambda retorna metadados da função do Lambda e um URL pré-assinado que você pode usar para fazer download do pacote de implantação da função.

```
aws lambda get-function --function-name my-function
```

Você deve ver a saída a seguir:

```
{  
    "Configuration": {  
        "FunctionName": "my-function",  
        "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",  
        "Runtime": "nodejs12.x",  
        "Role": "arn:aws:iam::123456789012:role/lambda-ex",  
        "CodeSha256": "FpFMvUhayLkOoVBpNuNiIVML/tuGv2iJQ7t0yWVTU8c=",  
        "Version": "$LATEST",  
        "TracingConfig": {  
            "Mode": "PassThrough"  
        },  
        "RevisionId": "88ebe1e1-bfdf-4dc3-84de-3017268fa1ff",  
        ...  
    },  
    "Code": {  
        "RepositoryType": "S3",  
        "Location": "https://awslambda-us-east-2-tasks.s3.us-east-2.amazonaws.com/  
snapshots/123456789012/my-function-4203078a-b7c9-4f35..."  
    }  
}
```

Para obter mais informações, consulte [GetFunction \(p. 842\)](#).

Limpar

Execute o comando `delete-function` a seguir para excluir a função `my-function`.

```
aws lambda delete-function --function-name my-function
```

Exclua a função do IAM que você criou no console do IAM. Para obter mais informações sobre como excluir uma função, consulte [Excluir funções ou perfis de instância](#) no Manual do usuário do IAM.

Cotas Lambda

Computação e armazenamento

O Lambda define cotas para a quantidade de recursos computacionais e de armazenamento que você pode usar para executar e armazenar funções. As cotas a seguir são aplicadas por região da AWS e podem ser aumentadas. Para obter mais informações, consulte [Solicitar um aumento da cota](#) no Manual do usuário do Service Quotas.

Recurso	Cota padrão	Pode ser aumentado até
Execuções simultâneas	1.000	Dezenas de milhares
Armazenamento para funções carregadas (arquivos.zip) e camadas. Cada versão de função e versão de camada consome armazenamento.	75 GB	Terabytes
Armazenamento para funções definidas como imagens de contêiner. Essas imagens são armazenadas no Amazon ECR.	Consulte Cotas de serviço do Amazon ECR .	
Interfaces de rede elásticas por Virtual Private Cloud (VPC) (p. 124) Note Esta cota é compartilhada com outros serviços, como o Amazon Elastic File System (Amazon EFS). Consulte Cotas da Amazon VPC .	250	Centenas

Para obter detalhes sobre a simultaneidade e sobre como o Lambda dimensiona a simultaneidade da função em resposta ao tráfego, consulte [Escalabilidade da função do Lambda \(p. 32\)](#).

Configuração, implantação e execução de funções

As cotas a seguir se aplicam à configuração de funções, às implantações e à execução. Eles não podem ser alterados.

Note

A documentação do Lambda, as mensagens de log e o console usam a abreviatura MB (em vez de MiB) para se referir a 1024 KB.

Recurso	Quota
Alocação de memória (p. 95) da função	128 MB a 10.240 MB, em incrementos de 1 MB.
Tempo-limite (p. 95) da função.	900 segundos (15 minutos)
Variáveis de ambiente (p. 99) da função	4 KB, para todas as variáveis de ambiente associadas à função, em agregado

Recurso	Quota
Política baseada em recursos (p. 62) da função	20 KB
Função camadas (p. 85)	cinco camadas
Simultaneidade de intermitência (p. 32) da função	500 a 3000 (varia por região)
Carga da invocação (p. 157) (solicitação e resposta)	6 MB (síncrona) 256 KB (assíncrona)
Tamanho do pacote de implantação (arquivo .zip) (p. 37)	50 MB (compactado, para upload direto) 250 MB (descompactado) Essa cota se aplica a todos os arquivos que você carrega, inclusive camadas e tempos de execução personalizados. 3 MB (editor de console)
Tamanho do pacote do código da imagem do contêiner (p. 266)	10 GB
Eventos de teste (editor de console)	10
/tmpArmazenamento do diretório do	512 MB
Descrições do arquivo	1,024
Processos de execução/threads	1,024

Solicitações da API do Lambda

As cotas a seguir estão associadas a solicitações de API do Lambda.

Recurso	Quota
Solicitações de invocação por região (solicitações por segundo)	10 x cota de execuções simultâneas (synchronous (p. 158) , todas as origens) 10 x cota de execuções simultâneas (Assíncrona (p. 161) , não-AWS)
Solicitações de invocação por região (solicitações por segundo) para origens de serviços da AWS (p. 277) assíncronos	Solicitações ilimitadas aceitas. A taxa de execução é baseada na simultaneidade disponível para a função. Consulte Invocação assíncrona (p. 161) .
Frequência de invocação por alias ou versão de função (solicitações por segundo)	10 x simultaneidade provisionada (p. 113) alocada

Recurso	Quota
	Note Essa cota se aplica somente às funções que usam simultaneidade provisionada.
Solicitações de API GetFunction (p. 842)	100 solicitações por segundo
Solicitações de API GetPolicy (p. 869)	15 solicitações por segundo
Restante das solicitações de API do plano de controle (exclui solicitações de invocação, GetFunction e GetPolicy)	15 solicitações por segundo

Outros serviços

Cotas para outros serviços, como oAWS Identity and Access Management(IAM), Amazon CloudFront (Lambda @Edge) e Amazon Virtual Private Cloud (Amazon VPC) podem afetar as funções do Lambda. Para obter mais informações, consulte [Cotas de serviço da AWS](#) na Referência geral da Amazon Web Services e [Usar o AWS Lambda com outros serviços \(p. 277\)](#).

AWS LambdaPermissões

Use o AWS Identity and Access Management (IAM) para gerenciar acesso à API e aos recursos do Lambda, como funções e camadas. Para usuários e aplicações na conta que usa o Lambda, gerencie permissões em uma política de permissões que é possível aplicar a usuários, grupos ou funções do IAM. Para conceder permissões a outras contas ou serviços da AWS que usem os recursos do Lambda, você usa uma política que se aplica ao recurso propriamente dito.

Uma função do Lambda também tem uma política, chamada de [função de execução \(p. 57\)](#), que concede a ela permissão para acessar recursos e serviços da AWS. No mínimo, a função precisa acessar o Amazon CloudWatch Logs para a transmissão de registros. Se você [use AWS X-Ray para rastrear sua função \(p. 477\)](#), ou sua função acessa serviços com o AWSSDK, você concede permissão para chamá-los na função de execução. O Lambda também usa a função de execução para obter permissão para ler de fontes de eventos quando você usa um [Mapeamento de origens de \(p. 170\)](#) para acionar sua função do.

Note

Se a função precisar de acesso à rede para um recurso, como um banco de dados relacional que não esteja acessível por meio de APIs da AWS ou da Internet, [configure-a para se conectar à VPC \(p. 124\)](#).

Usar o [Políticas baseadas em recursos do \(p. 62\)](#) para dar outras contas e AWS para usar seus recursos do Lambda. Entre os recursos do Lambda estão funções, versões, aliases e versões da camada. Cada um desses recursos tem uma política de permissões que se aplica quando o recurso é acessado, além de todas as políticas que se aplicam ao usuário. Quando um serviço da AWS, como o Amazon S3, chama a função do Lambda, a política baseada em recursos dá acesso a ele.

Para gerenciar permissões para usuários e aplicações nas contas, [use as políticas gerenciadas fornecidas pelo Lambda \(p. 67\)](#), ou crie as próprias. O console do Lambda usa vários serviços para obter informações sobre a configuração e os acionadores da função. Use as políticas gerenciadas no estado em que se encontram ou como um ponto de partida para políticas mais restritivas.

Restrinja permissões de usuário pelo recurso afetado por uma ação e, em alguns casos, por condições adicionais. Por exemplo, especifique um padrão para o Amazon Resource Name (ARN – Nome de recurso da Amazon) de uma função que exija que um usuário inclua o nome do usuário nos nomes das funções criadas. Além disso, é possível adicionar uma condição que exige que o usuário configure funções para usar uma camada específica para, por exemplo, extrair o software de registro. Para os recursos e as condições compatíveis com cada ação, consulte [Recursos e condições \(p. 72\)](#).

Para obter mais informações sobre o IAM, consulte [O que é o IAM?](#) no Manual do usuário do IAM.

Para obter mais informações sobre como colocar princípios de segurança em prática em aplicações do Lambda, consulte [Segurança](#) no Guia do operador do Lambda.

Tópicos

- [AWS LambdaFunção de execução do \(p. 57\)](#)
- [Uso de políticas baseadas em recursos para o AWS Lambda \(p. 62\)](#)
- [Políticas do IAM baseadas em identidade para o Lambda \(p. 67\)](#)
- [Recursos e condições para ações do Lambda \(p. 72\)](#)
- [Usar limites de permissões para aplicativos do AWS Lambda \(p. 79\)](#)

AWS LambdaFunção de execução do

A função de execução de uma função do Lambda é uma função do AWS Identity and Access Management (IAM) que concede à função permissão para acessar serviços e recursos da AWS. Você fornece essa função ao criar uma função, e o Lambda assume a função quando ela é invocada. É possível criar uma função de execução para desenvolvimento que tenha permissão para enviar logs ao Amazon CloudWatch e para fazer upload dos dados de rastreamento no AWS X-Ray.

Como visualizar a função de execução de uma função

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e, em seguida, escolha Permissions (Permissões).
4. Em Resource summary (Resumo do recurso), visualize os serviços e os recursos que a função pode acessar. O exemplo a seguir mostra as permissões do CloudWatch Logs que o Lambda adiciona a uma função de execução quando ela é criada no console do Lambda.

Resource summary



Lambda obtained this information from the following policy statements:

- Managed policy AWSLambdaBasicExecutionRole-493eae43-bffa-xmpl-9e86-cf39eeae586c,
- Managed policy AWSLambdaBasicExecutionRole-493eae43-bffa-xmpl-9e86-cf39eeae586c,

5. Escolha um serviço na lista suspensa para ver as permissões relacionadas a ele.

Adicione ou remova permissões da função de execução de uma função a qualquer momento ou configure a função para usar uma diferente. Adicione permissões para qualquer serviço que a função chama com o AWS SDK e para serviços usados pelo Lambda para habilitar recursos opcionais.

Ao adicionar permissões à sua função, também atualize o código ou a configuração. Isso força as instâncias em execução da função, com credenciais desatualizadas, a serem encerradas e substituídas.

Tópicos

- [Criar uma função de execução no console do IAM \(p. 58\)](#)
- [Conceda acesso de menor privilégio à sua função de execução do Lambda \(p. 58\)](#)
- [Gerenciar funções com a API do IAM \(p. 59\)](#)
- [Políticas gerenciadas da AWS para recursos do Lambda \(p. 60\)](#)

Criar uma função de execução no console do IAM

Por padrão, o Lambda cria uma função de execução com permissões mínimas quando você [cria uma função no console do Lambda \(p. 9\)](#). Também é possível criar uma função de execução no console do IAM.

Como criar uma função de execução no console do IAM

1. Abra a página [Roles \(Funções\)](#) no console do IAM.
2. Selecione Create role.
3. Em Common use cases (Casos de uso comuns), selecione Lambda.
4. Escolha Próximo: Permissões.
5. Em Attach permissions policies (Anexar políticas de permissões), selecione as políticas gerenciadas da AWS AWSLambdaBasicExecutionRole e AWSXRayDaemonWriteAccess.
6. Escolha Next: Tags (Próximo: tags).
7. Selecione Next: Review.
8. Em Role name (Nome da função), insira **lambda-role**.
9. Selecione Create role.

Para obter instruções, consulte [Criar uma função para um serviço da AWS \(console\)](#) no Guia do usuário do IAM.

Conceda acesso de menor privilégio à sua função de execução do Lambda

Quando você cria uma função do IAM pela primeira vez para sua função do Lambda durante a fase de desenvolvimento, às vezes você pode conceder permissões além do que é necessário. Antes de publicar sua função no ambiente de produção, a prática recomendada é ajustar a política para incluir somente as permissões necessárias. Para obter mais informações, consulte [conceder privilégio mínimo](#).

Use o IAM Access Analyzer para ajudar a identificar as permissões necessárias para a política de função de execução do IAM. O IAM Access Analyzer revisa sua AWS CloudTrail registra no intervalo de datas especificado e gera um modelo de política com apenas as permissões que a função utilizou durante esse período. Você pode usar o modelo para criar uma política gerenciada com permissões refinadas e anexá-la à função do IAM. Dessa forma, você concede apenas as permissões necessárias à interação com os recursos da AWS, de acordo com a especificidade do caso de uso.

Para saber mais, consulte [Gerar políticas com base na atividade de acesso](#) no Guia do usuário do IAM.

Gerenciar funções com a API do IAM

Para criar uma função de execução com a AWS Command Line Interface (AWS CLI), use o comando `create-role`.

No exemplo a seguir, você especifica a política de confiança em linha. Os requisitos para escapar de aspas na string JSON variam dependendo do seu 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"}]}'
```

Também é possível definir a [política de confiança](#) para a função usando um arquivo JSON. No exemplo a seguir, `trust-policy.json` é um arquivo no diretório atual. Esta política de confiança permite que o Lambda use as permissões da função fornecendo o principal de serviço `lambda.amazonaws.com` permissão para chamar o método AWS Security Token Service `AssumeRole` .

Example `trust-policy.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
```

Você deve ver a saída a seguir:

```
{
  "Role": {
    "Path": "/",
    "RoleName": "lambda-ex",
    "RoleId": "AROAQFOXMPL6TZ6ITKWND",
    "Arn": "arn:aws:iam::123456789012:role/lambda-ex",
    "CreateDate": "2020-01-17T23:19:12Z",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "lambda.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
}
```

Para adicionar permissões à função, use o comando `attach-policy-to-role`. Inicie adicionando o `AWSLambdaBasicExecutionRole` política gerenciada pela.

```
aws iam attach-role-policy --role-name lambda-ex --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

Políticas gerenciadas da AWS para recursos do Lambda

As seguintes políticas gerenciadas da AWS oferecem permissões obrigatórias para que se usem os recursos do Lambda:

- `AWSLambdaBasicExecutionRole`: permissão para fazer upload de logs para o CloudWatch
- `AWSLambdaDynamoDBExecutionRole`: permissão para ler registros do Amazon DynamoDB Streams.
- `AWSLambdaKinesisExecutionRole`: permissão para ler eventos de um fluxo de dados ou consumidor do Amazon Kinesis.
- `AWSLambdaMQExecutionRole`: permissão para ler registros de um agente do Amazon MQ.
- `AWSLambdaDamskExecutionRole`— Permissão para ler registros de um cluster do Amazon Managed Streaming for Apache Kafka (Amazon MSK).
- `AWSLambdaDasqSqueueExecutionRole`— Permissão para ler uma mensagem de uma fila do Amazon Simple Queue Service (Amazon SQS).
- `AWSLambdaVPCAccessExecutionRole`: permissão para gerenciar interfaces de rede elástica a fim de conectar a função a uma virtual private cloud (VPC).
- `AWSXRayDaemonWriteAccess`: permissão para fazer upload dos dados de rastreamento no X-Ray.
- `CloudWatchLambdaInsightsExecutionRolePolicy`: permissão para gravar métricas de tempo de execução no CloudWatch Lambda Insights.

Para alguns recursos, o console do Lambda tenta adicionar permissões ausentes à sua função de execução em uma política gerenciada pelo cliente. Essas políticas podem tornar-se numerosas. Para evitar a criação de políticas adicionais, adicione as políticas gerenciadas da AWS relevantes à função de execução antes de habilitar os recursos.

Quando você usa um [mapeamento de fontes de eventos \(p. 170\)](#) para invocar a função, o Lambda usa a função de execução a fim de ler dados de eventos. Por exemplo, um mapeamento de origem do evento para o Kinesis lê eventos de um fluxo de dados e os envia para a função em lotes. Use mapeamentos de origem do evento com os seguintes serviços:

Serviços dos quais o Lambda lê eventos

- [Amazon DynamoDB \(p. 333\)](#)
- [Amazon Kinesis \(p. 387\)](#)
- [Amazon MQ \(p. 411\)](#)
- [Amazon Managed Streaming for Apache Kafka \(p. 419\)](#)
- [Apache Kafka autogerenciado \(p. 377\)](#)
- [Amazon Simple Queue Service \(p. 465\)](#)

Além das políticas gerenciadas da AWS, o console do Lambda fornece modelos para criar uma política personalizada com as permissões para casos de uso adicionais. Ao criar uma função no console do Lambda, opte por criar uma nova função de execução com permissões de um ou mais modelos. Esses modelos também são aplicados automaticamente quando você cria uma função a partir de um esquema,

ou quando configura opções que exijam acesso a outros serviços. Os modelos de exemplo estão disponíveis no [repositório do GitHub](#) deste guia.

Uso de políticas baseadas em recursos para o AWS Lambda

O AWS Lambda oferece suporte a políticas de permissões baseadas em recursos para funções e camadas do Lambda. As políticas baseadas em recursos permitem conceder permissão de uso a outras contas da AWS por recurso. Também é possível usar uma política baseada em recurso para permitir que um serviço da AWS invoque a função em seu nome.

Para funções do Lambda, [conceda uma permissão à conta \(p. 64\)](#) para invocar ou gerenciar uma função. É possível adicionar várias instruções para conceder acesso a várias contas ou permitir que qualquer conta invoque a função. Também é possível usar a política para [conceder permissão de invocar a um serviço da AWS \(p. 63\)](#) que invoca uma função em resposta à atividade em sua conta.

Como visualizar a política baseada em recursos de uma função

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e, em seguida, escolha Permissions (Permissões).
4. Role para baixo até Política baseada em recursos e escolha Exibir documento de política. A política baseada em recursos mostra as permissões aplicadas quando outra conta ou serviço da AWS tenta acessar a função. O exemplo a seguir mostra uma instrução que permite ao Amazon S3 invocar uma função chamada `my-function` para um bucket chamado `my-bucket` na conta `123456789012`.

Example Política baseada em recurso

```
{  
    "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:::my-bucket"  
                }  
            }  
        }  
    ]  
}
```

Para camadas do Lambda, é possível usar somente uma política baseada em recurso em uma versão específica da camada, em vez de toda a camada. Além de políticas que concedem permissão a uma única conta ou a várias as contas, para camadas, também é possível conceder permissão a todas as contas em uma organização.

Note

Só é possível atualizar políticas baseadas em recursos para recursos do Lambda dentro do escopo das ações de API [AddPermission \(p. 775\)](#) e [AddLayerVersionPermission \(p. 771\)](#).

Atualmente, não é possível criar políticas para os recursos do Lambda em JSON ou usar condições não mapeadas para parâmetros dessas ações.

As políticas baseadas em recursos se aplicam a uma única função, versão, alias ou versão da camada. Elas concedem permissão para um ou mais serviços e contas. Para contas confiáveis que você deseja que tenham acesso a vários recursos, ou para usar ações de API não compatíveis com políticas baseadas em recursos, use [funções entre contas \(p. 67\)](#).

Tópicos

- [Conceder acesso de função aos serviços da AWS \(p. 63\)](#)
- [Conceder acesso de função a outras contas \(p. 64\)](#)
- [Conceder acesso de camada a outras contas \(p. 65\)](#)
- [Limpar políticas baseadas em recursos \(p. 66\)](#)

Conceder acesso de função aos serviços da AWS

Ao [usar um serviço da AWS para invocar a função \(p. 277\)](#), você concede permissão em uma instrução em uma política baseada em recursos. É possível aplicar a instrução a toda a função a ser invocada ou gerenciada ou limitar a instrução a uma única versão ou alias.

Note

Quando você adiciona um acionador à função com o console do Lambda, este atualiza a política baseada em recursos da função para permitir que o serviço a invoque. Para conceder permissões a outras contas ou serviços que não estejam disponíveis no console do Lambda, é possível usar a AWS CLI.

Adicione uma instrução com o comando `add-permission`. A instrução de política baseada em recursos mais simples permite que um serviço invoque uma função. O comando a seguir concede permissão ao Amazon SNS para invocar uma função chamada `my-function`.

```
aws lambda add-permission --function-name my-function --action lambda:InvokeFunction --statement-id sns \
--principal sns.amazonaws.com --output text
```

Você deve ver a saída a seguir:

```
{"Sid":"sns","Effect":"Allow","Principal": \
{"Service":"sns.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-east-2:123456789012:function:my-function"}
```

Isso permite que o Amazon SNS chame a API `lambda:Invoke` para a função, mas não restrinja o tópico do Amazon SNS que aciona a invocação. Para garantir que a função só seja invocada por um recurso específico, especifique o Amazon Resource Name (ARN – Nome de recurso da Amazon) do recurso com a opção `source-arn`. O comando a seguir só permite que o Amazon SNS invoque a função para assinaturas de um tópico chamado `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
```

Alguns serviços podem invocar funções em outras contas. Se você especificar um ARN de origem que tenha o ID da conta, isso não será um problema. No entanto, para o Amazon S3 a origem é um bucket cujo ARN não tem um ID da conta. É possível que você consiga excluir o bucket e outra conta consiga criar um bucket com o mesmo nome. Use a opção `source-account` com o ID da sua conta para garantir que apenas os recursos na conta possam invocar a função.

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

Conceder acesso de função a outras contas

Para conceder permissões a outra conta da AWS, especifique o ID da conta como o `principal`. O exemplo a seguir concede permissão à conta 210987654321 para invocar `my-function` com o alias `prod`.

```
aws lambda add-permission --function-name my-function:prod --statement-id xaccount --action lambda:InvokeFunction \
--principal 210987654321 --output text
```

Você deve ver a saída a seguir:

```
{"Sid": "xaccount", "Effect": "Allow", "Principal": {"AWS": "arn:aws:iam::210987654321:root"}, "Action": "lambda:InvokeFunction", "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function"}
```

A política baseada em recursos concede permissão para a outra conta acessar a função, mas não permite que os usuários nessa conta excedam suas permissões. Os usuários na outra conta devem ter as [permissões de usuário \(p. 67\)](#) correspondentes para usar a API do Lambda.

Para limitar o acesso a um usuário ou função em outra conta, especifique o ARN completo da identidade como o `principal`. Por exemplo, `arn:aws:iam::123456789012:user/developer`.

O [alias \(p. 108\)](#) limita qual versão a outra conta pode chamar. Ele exige que a outra conta inclua o alias no ARN da função.

```
aws lambda invoke --function-name arn:aws:lambda:us-west-2:123456789012:function:my-function:prod out
```

Você deve ver a saída a seguir:

```
{
  "StatusCode": 200,
  "ExecutedVersion": "1"
}
```

Depois disso, o proprietário da função pode atualizar o alias para apontar para uma nova versão sem que o chamador precise alterar a maneira como eles invocam sua função. Isso garante que a outra conta não precise alterar o código para usar a nova versão, e ela tem permissão somente para invocar a versão da função associada ao alias.

Conceda acesso entre contas para a maioria das ações de API que [operam em uma função existente \(p. 75\)](#). Por exemplo, é possível conceder acesso a `lambda>ListAliases` para obter uma lista de aliases ou a `lambda:GetFunction` para permitir que eles façam download do código da função.

Adicione cada permissão separadamente ou use `lambda:*` para conceder acesso a todas as ações da função especificada.

APIs entre contas

Atualmente, o Lambda não oferece suporte a ações entre contas para todas as APIs por meio de políticas baseadas em recursos. As seguintes APIs são compatíveis:

- [Invoke \(p. 875\)](#)
- [GetFunction \(p. 842\)](#)
- [GetFunctionConfiguration \(p. 851\)](#)
- [UpdateFunctionCode \(p. 965\)](#)
- [DeleteFunction \(p. 818\)](#)
- [PublishVersion \(p. 919\)](#)
- [ListVersionsByFunction \(p. 911\)](#)
- [CreateAlias \(p. 779\)](#)
- [GetAlias \(p. 832\)](#)
- [ListAliases \(p. 883\)](#)
- [UpdateAlias \(p. 949\)](#)
- [DeleteAlias \(p. 808\)](#)
- [GetPolicy \(p. 869\)](#)
- [PutFunctionConcurrency \(p. 930\)](#)
- [DeleteFunctionConcurrency \(p. 822\)](#)
- [ListTags \(p. 909\)](#)
- [TagResource \(p. 945\)](#)
- [UntagResource \(p. 947\)](#)

Para conceder permissão a outras contas para várias funções ou para ações que não operem em uma função, recomendamos usar as [funções do IAM \(p. 67\)](#).

Conceder acesso de camada a outras contas

Para conceder permissão de uso de camadas a outra conta, adicione uma instrução à política de permissões da versão da camada usando o comando `add-layer-version-permission`. Em cada instrução, você pode conceder permissão a uma única conta, a todas as contas ou a uma organização.

```
aws lambda add-layer-version-permission --layer-name xray-sdk-nodejs --statement-id
xaccount \
--action lambda:GetLayerVersion --principal 210987654321 --version-number 1 --output text
```

Você deve ver saída semelhante a:

```
e210ffdc-e901-43b0-824b-5fcfd0dd26d16 {"Sid":"xaccount","Effect":"Allow","Principal":
{"AWS":"arn:aws:iam::210987654321:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:lambda:
east-2:123456789012:layer:xray-sdk-nodejs:1"}
```

As permissões se aplicam apenas a uma única versão de camada. Repita o processo sempre que criar uma nova versão da camada.

Para conceder permissão a todas as contas em uma organização, use a opção `organization-id`. O exemplo a seguir concede a todas as contas em uma organização permissão para usar a versão 3 de uma camada.

```
aws lambda add-layer-version-permission --layer-name my-layer \
--statement-id engineering-org --version-number 3 --principal '*' \
--action lambda:GetLayerVersion --organization-id o-t194hfs8cz --output text
```

Você deve ver a saída a seguir:

```
b0cd9796-d4eb-4564-939f-de7fe0b42236 {"Sid":"engineering-
org","Effect":"Allow","Principal":"*","Action":"lambda:GetLayerVersion","Resource":"arn:aws:lambda:us-
east-2:123456789012:layer:my-layer:3","Condition":{"StringEquals":{"aws:PrincipalOrgID":"o-
t194hfs8cz"}}}"
```

Para conceder permissão a todas as contas da AWS, use * para a entidade principal e omita o ID da organização. Para várias contas ou organizações, é necessário adicionar várias instruções.

Limpar políticas baseadas em recursos

Para exibir a política baseada em recursos de uma função, use o comando `get-policy`.

```
aws lambda get-policy --function-name my-function --output text
```

Você deve ver a saída a seguir:

```
{"Version":"2012-10-17","Id":"default","Statement": [
{"Sid":"sns","Effect":"Allow","Principal":
{"Service":"s3.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-
east-2:123456789012:function:my-function","Condition":{"ArnLike":
{"AWS:SourceArn":"arn:aws:sns:us-east-2:123456789012:lambda*"}}}]}
```

Para versões e aliases, acrescente o número da versão ou o alias ao nome da função.

```
aws lambda get-policy --function-name my-function:PROD
```

Para remover permissões da função, use `remove-permission`.

```
aws lambda remove-permission --function-name example --statement-id sns
```

Use o comando `get-layer-version-policy` para visualizar as permissões em uma camada.

```
aws lambda get-layer-version-policy --layer-name my-layer --version-number 3 --output text
```

Você deve ver a saída a seguir:

```
b0cd9796-d4eb-4564-939f-de7fe0b42236 {"Sid":"engineering-
org","Effect":"Allow","Principal":"*","Action":"lambda:GetLayerVersion","Resource":"arn:aws:lambda:us-
west-2:123456789012:layer:my-layer:3","Condition":{"StringEquals":{"aws:PrincipalOrgID":"o-
t194hfs8cz"}}}"
```

Use `remove-layer-version-permission` para remover instruções da política.

```
aws lambda remove-layer-version-permission --layer-name my-layer --version-number 3 --
statement-id engineering-org
```

Políticas do IAM baseadas em identidade para o Lambda

Use políticas baseadas em identidade no AWS Identity and Access Management (IAM) para conceder a usuários na conta acesso ao Lambda. As políticas baseadas em identidade podem se aplicar diretamente aos usuários ou a funções e grupos associados a um usuário. Também é possível conceder a usuários em outra conta permissão para assumir uma função na conta e acessar os recursos do Lambda.

LambdaAWSpolíticas gerenciadas que concedem acesso a ações da API do Lambda e, em alguns casos, acesso a outrasAWSserviços usados para desenvolver e gerenciar recursos do Lambda. Lambda atualiza as políticas gerenciadas conforme necessário para garantir que os usuários tenham acesso a novos recursos quando eles forem lançados.

Note

As políticas gerenciadas da AWS AWSLambdaFullAccess e AWSLambdaReadOnlyAccess se tornarão [defasadas](#) no dia 1º de março de 2021. Após essa data, você não pode anexar essas políticas a novos usuários do IAM. Para obter mais informações, consulte o [tópico sobre solução de problemas \(p. 737\)](#).

- AWSLambda_FullAccess: concede acesso total a ações do Lambda e a outros serviços da AWS usados para desenvolver e manter recursos do Lambda. Esta política foi criada com a redução do escopo da política anterior AWSLambdaFullAccess.
- AWSLambda_ReadOnlyAccess: concede acesso somente leitura aos recursos do Lambda. Esta política foi criada com a redução do escopo da política anterior AWSLambdaReadOnlyAccess.
- AWSLambdaRole: concede permissões para invocar funções do Lambda.

As políticas gerenciadas do AWS concedem permissão a ações da API sem restringir as funções ou as camadas do Lambda que um usuário pode modificar. Para um controle refinado, crie as próprias políticas que limitam o escopo das permissões de um usuário.

Seções

- [Desenvolvimento da função \(p. 67\)](#)
- [Desenvolvimento e uso da camada \(p. 70\)](#)
- [Funções entre contas \(p. 71\)](#)
- [Chaves de condição para configurações de VPC \(p. 71\)](#)

Desenvolvimento da função

Use políticas baseadas em identidade para permitir que os usuários executem operações em funções do Lambda.

Note

Para uma função definida como uma imagem de contêiner, a permissão do usuário para acessar a imagem DEVE ser configurada no Amazon Elastic Container Registry. Para obter um exemplo, consulte [Permissões do Amazon ECR. \(p. 91\)](#)

A seguir, um exemplo de uma política de permissões com escopo limitado. Ele permite que um usuário crie e gerencie funções do Lambda com um prefixo designado (`intern-`) e configuradas com uma função de execução designada.

Example Política de desenvolvimento da função

```
{  
    "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": {  
                "StringLike": {  
                    "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": "*"  
        }  
    ]  
}
```

```
        "Resource": "arn:aws:logs:*:log-group:/aws/lambda/intern-*"
    ]
}
```

As permissões na política são organizadas em declarações com base nos [recursos e nas condições \(p. 72\)](#) compatíveis.

- **ReadOnlyPermissions**: o console do Lambda usa essas permissões quando você procura e exibe funções. Elas não oferecem suporte a padrões ou condições de recursos.

```
"Action": [
    "lambda:GetAccountSettings",
    "lambda:GetEventSourceMapping",
    "lambda:GetFunction",
    "lambda:GetFunctionConfiguration",
    "lambda:GetFunctionCodeSigningConfig",
    "lambda:GetFunctionConcurrency",
    "lambda>ListEventSourceMappings",
    "lambda>ListFunctions",
    "lambda>ListTags",
    "iam>ListRoles"
],
"Resource": "*"
```

- **DevelopFunctions**: use qualquer ação do Lambda que opere em funções prefixadas com `intern-`, exceto `AddPermission` e `PutFunctionConcurrency`. `AddPermission` modifica a [política baseada em recursos \(p. 62\)](#) na função e pode ter implicações de segurança. `PutFunctionConcurrency` reserva capacidade de escalabilidade para uma função e pode consumir a capacidade de outras funções.

```
"NotAction": [
    "lambda:AddPermission",
    "lambda:PutFunctionConcurrency"
],
"Resource": "arn:aws:lambda:*:function:intern-*"
```

- **DevelopEventSourceMappings** – Gerencie mapeamentos de origem do evento em funções prefixadas com `intern-`. Essas ações operam em mapeamentos de origem do evento, mas é possível restringi-las por função com uma condição.

```
"Action": [
    "lambda>DeleteEventSourceMapping",
    "lambda:UpdateEventSourceMapping",
    "lambda>CreateEventSourceMapping"
],
"Resource": "*",
"Condition": {
    "StringLike": {
        "lambda:FunctionArn": "arn:aws:lambda:*:function:intern-*"
    }
}
```

- **PassExecutionRole**: exiba e passe apenas uma função chamada `intern-lambda-execution-role`, que deve ser criada e gerenciada por um usuário com permissões do IAM. `PassRole` é usado quando você atribui uma função de execução a uma função.

```
"Action": [
    "iam>ListRolePolicies",
    "iam>ListAttachedRolePolicies",
    "iam>GetRole",
    "iam>GetRolePolicy",
    "iam>PassRole",
    "iam>SimulatePrincipalPolicy"
],
"Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
```

- **ViewLogs:** use o CloudWatch Logs para exibir logs de funções prefixadas com `intern-`.

```
"Action": [
    "logs:)"
],
"Resource": "arn:aws:logs:***:log-group:/aws/lambda/intern-*"
```

Essa política permite que um usuário começar a usar com o Lambda, sem colocar os recursos de outros usuários em risco. Ela não permite que um usuário configure uma função para ser disparada ou acione outros serviços da AWS, o que exige permissões mais amplas do IAM. Ela também não inclui permissão para serviços que não ofereçam suporte a políticas de escopo limitado, como CloudWatch e X-Ray. Use as políticas somente leitura desses serviços para conceder ao usuário acesso a métricas e dados de rastreamento.

Ao configurar triggers para a sua função, você precisa de acesso para usar o produto da AWS que invoca a sua função. Por exemplo, para configurar um acionador do Amazon S3, é necessária a permissão para ações do Amazon S3 a fim de gerenciar notificações do bucket. Muitas dessas permissões estão incluídas na política gerenciada `AWSLambdaFullAccess`. As políticas de exemplo estão disponíveis no [repositório do GitHub](#) deste guia.

Desenvolvimento e uso da camada

A política a seguir concede permissões a um usuário para criar camadas e usá-las com funções. Os padrões de recursos permitem que o usuário trabalhe em qualquer região da AWS e com qualquer versão de camada, desde que o nome da camada comece com `test-`.

Example política de desenvolvimento da camada

```
{
    "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-*:*"
        }
    ]
}
```

}

Também é possível impor o uso da camada durante a criação da função e a configuração com a condição `lambda:Layer`. Por exemplo, evite que os usuários usem camadas publicadas por outras contas. A política a seguir adiciona uma condição às ações `CreateFunction` e `UpdateFunctionConfiguration` para exigir que todas as camadas especificadas venham da conta 123456789012.

```
{  
    "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:*:*"  
                    ]  
                }  
            }  
        }  
    ]  
}
```

Para garantir que a condição se aplique, verifique se não há outras instruções concedendo ao usuário permissão para essas ações.

Funções entre contas

Aplice qualquer uma das políticas e instruções anteriores a uma função, que é possível acabar compartilhando com outra conta para conceder a ela acesso aos recursos do Lambda. Diferentemente de um usuário do IAM, uma função não tem credenciais para autenticação. Em vez disso, ela tem uma política confiável que especifica quem pode assumir a função e usar as permissões.

Use funções entre contas para dar a contas nas quais você confia acesso a ações e recursos do Lambda. Se você só quiser conceder permissão para invocar uma função ou usar uma camada, use [políticas baseadas em recursos \(p. 62\)](#).

Para obter mais informações, consulte [Funções do IAM](#) no Manual do usuário do IAM.

Chaves de condição para configurações de VPC

É possível usar chaves de condição para configurações de VPC a fim de fornecer controles de permissão adicionais para as funções do Lambda. Por exemplo, é possível impor que todas as funções do Lambda em sua organização estejam conectadas a uma VPC. Você também pode especificar as sub-redes e os grupos de segurança que as funções têm ou não têm permissão para usar.

Para obter mais informações, consulte [the section called “Usar chaves de condição do IAM para configurações de VPC” \(p. 126\)](#).

Recursos e condições para ações do Lambda

Você pode restringir o escopo das permissões de um usuário especificando recursos e condições em uma política do IAM. Cada ação de API oferece suporte a uma combinação de tipos de condição e recurso que varia de acordo com o comportamento da ação.

Cada instrução de política do IAM concede permissão a uma ação realizada em um recurso. Quando a ação não atua em um recurso indicado, ou quando você concede permissão para executar a ação em todos os recursos, o valor do recurso na política é um curinga (*). Para muitas ações de API, restrinja os recursos que um usuário pode modificar especificando o Amazon Resource Name (ARN – Nome de recurso da Amazon) de um recurso ou um padrão de ARN correspondente a vários recursos.

Para restringir as permissões por recurso especifique o recurso por ARN.

Formato do ARN de recurso Lambda

- Funçã – arn:aws:lambda:**us-west-2:123456789012:function:my-function**
- Versão da função – arn:aws:lambda:**us-west-2:123456789012:function:my-function:1**
- Alias da função – arn:aws:lambda:**us-west-2:123456789012:function:my-function:TEST**
- Mapeamento de origens de eventos d – arn:aws:lambda:**us-west-2:123456789012:eventsource-mapping:fa123456-14a1-4fd2-9fec-83de64ad683de6d47**
- Camada – arn:aws:lambda:**us-west-2:123456789012:layer:my-layer**
- Versão da camad – arn:aws:lambda:**us-west-2:123456789012:layer:my-layer:1**

Por exemplo, a política a seguir permite que um usuário na conta 123456789012 invoque uma função chamada **my-function** na região Oeste dos EUA (Oregon).

Example invocar política de função

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Invoke",  
            "Effect": "Allow",  
            "Action": [  
                "lambda:InvokeFunction"  
            ],  
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-function"  
        }  
    ]  
}
```

Trata-se de um caso especial em que o identificador da ação (`lambda:InvokeFunction`) é diferente da operação da API ([Invoke \(p. 875\)](#)). Para outras ações, o identificador da ação é o nome da operação com o prefixo `lambda:`.

As condições são um elemento opcional da política que aplica lógica adicional para determinar se uma ação é permitida. Além de [condições comuns](#) compatíveis com todas as ações, o Lambda define os tipos de condição que você pode usar para restringir os valores dos parâmetros adicionais em algumas ações.

Por exemplo, a condição `lambda:Principal` permite restringir o serviço ou a conta para que um usuário pode conceder acesso de invocação em uma política baseada em recursos da função. A política a seguir permite que um usuário conceda permissão para que tópicos do SNS invoquem uma função chamada `test`.

Example gerenciar permissões de política de função

```
{  
    "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"  
                }  
            }  
        }  
    ]  
}
```

A condição requer que o principal seja o Amazon SNS e não outro serviço ou outra conta. O padrão do recurso exige que o nome da função seja `test` e inclua um número de versão ou alias. Por exemplo, `test:v1`.

Para obter mais informações sobre recursos e condições do Lambda e outros serviços da AWS, consulte [Ações, recursos e chaves de condição](#) no Guia do usuário do IAM.

Seções

- [Nomes de recursos de função \(p. 73\)](#)
- [Ações de função \(p. 75\)](#)
- [Ações de mapeamento da fonte de eventos \(p. 77\)](#)
- [Ações de camada \(p. 77\)](#)

Nomes de recursos de função

Você faz referência a uma função do Lambda em uma instrução de política usando um nomes de recurso da Amazon (ARN). O formato de um ARN de função depende se você estiver fazendo referência a toda a função, a [versão \(p. 106\)](#) de uma função ou um [alias \(p. 108\)](#).

Ao fazer chamadas de API do Lambda, os usuários podem especificar uma versão ou alias passando um ARN de versão ou um ARN de alias no parâmetro [GetFunction \(p. 842\)](#) do `FunctionName` ou definindo um valor no parâmetro [GetFunction \(p. 842\)](#) do `Qualifier`. O Lambda toma decisões de autorização comparando o elemento de recurso na política do IAM com o `FunctionName` passado nas chamadas de API.

É necessário usar os tipos de ARN de função corretos em suas políticas para obter os resultados esperados, especialmente em políticas que negam acesso. Recomendamos seguir as melhores práticas para usar instruções Deny com funções.

Melhores práticas para usar instruções Deny com funções

A tabela a seguir resume os recursos a serem usados nos efeitos Deny. Na coluna Recurso, `MyFunction` é o nome da função, `:1` faz referência à versão 1 da função e `MyAlias` é o nome de um alias de função.

Melhores práticas de recurso

Objetivo da política	Recurso		
Negar acesso a todas as versões de uma função	MyFunction*		
Negar acesso a uma conta específica	MyFunction:MyAlias e MyFunction		
Negar acesso a uma versão específica de uma função	MyFunction:1 e MyFunction		

As seções a seguir fornecem instruções de política de exemplo para cada um dos objetivos da política.

Note

É possível usar somente políticas baseadas em identidade para negar recursos de função específicos. Atualmente, o Lambda não oferece suporte ao efeito Deny em políticas baseadas em recursos.

Para obter a lista de ações em uma instrução de política, é possível adicionar qualquer uma das [ações definidas pelo Lambda](#) que agem em um recurso de função.

Negar acesso a todas as versões de função

A seguinte instrução de política baseada em identidade nega o acesso à ação `lambda:GetFunctionConfiguration` para todas as versões da função `my-function`. O caractere curinga no final da função do ARN garante que essa política se aplique a todos os ARNs de versão e alias.

Example Exemplo de política baseada em identidade

```
{  
    "Version": "2020-07-20",  
    "Statement": [  
        {  
            "Effect": "Deny",  
            "Action": [  
                "lambda:GetFunctionConfiguration"  
            ],  
            "Resource": "arn:aws:lambda:us-west-2:123456789012:function:my-function*"  
        }  
    ]  
}
```

Negar acesso a um alias de função específico

Para negar acesso a um alias específico, é necessário especificar o ARN de alias e o ARN de função não qualificado na política. Isso impede que os usuários acessem o alias específico passando o ARN não qualificado como o `FunctionName` e o alias como o `Qualifier`.

Note

Se você criar esse tipo de política, as chamadas de API precisarão fazer referência à versão não publicada da função especificando um ARN qualificado com o sufixo `$LAST` no parâmetro `FunctionName`.

A seguinte instrução de política baseada em identidade nega o acesso à ação `lambda:InvokeFunction` no alias `my-alias` da função `my-function`.

Example Exemplo de política baseada em identidade

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "DenySpecificAlias",  
            "Effect": "Deny",  
            "Action": "lambda:InvokeFunction",  
            "Resource": [  
                "arn:aws:lambda:us-west-2:123456789012:function:my-function:my-alias",  
                "arn:aws:lambda:us-west-2:123456789012:function:my-function"  
            ]  
        }  
    ]  
}
```

Negar acesso a uma versão de função específica

Para negar acesso a uma versão específica, é necessário especificar o ARN qualificado e o ARN não qualificado na política. Isso impede que os usuários accessem a versão específica passando o ARN não qualificado como o `FunctionName` e a versão como o `Qualifier`.

Note

Se você criar esse tipo de política, as chamadas de API precisarão fazer referência à versão não publicada da função especificando um ARN qualificado com o sufixo `$LASTT` no parâmetro `FunctionName`.

A seguinte instrução de política baseada em identidade nega o acesso à ação invocar na versão 1 da função `my-function`.

Example Exemplo de política baseada em identidade

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "DenySpecificFunctionVersion",  
            "Effect": "Deny",  
            "Action": [  
                "lambda:InvokeFunction"  
            ],  
            "Resource": [  
                "arn:aws:lambda:us-west-2:123456789012:function:my-function:1",  
                "arn:aws:lambda:us-west-2:123456789012:function:my-function"  
            ]  
        }  
    ]  
}
```

Ações de função

As ações que operam em uma função podem ser restritas a uma função específica pelo ARN da função, da versão ou do alias, conforme descrito na tabela a seguir. As ações que não oferecem suporte a restrições de recurso só podem ser concedidas a todos os recursos (*).

Functions

Ação	Recurso	Condição
AddPermission (p. 775) RemovePermission (p. 942)	Função Versão da função Alias da função	<code>lambda:Principal</code>
Invoke (p. 875) Permissão: <code>lambda:InvokeFunction</code>	Função Versão da função Alias da função	Nenhum
CreateFunction (p. 796) UpdateFunctionConfiguration (p. 974)	Função	<code>lambda:CodeSigningConfigArn</code> <code>lambda:Layer</code> <code>lambda:VpcIds</code> <code>lambda:SubnetIds</code> <code>lambda:SecurityGroupIds</code>
CreateAlias (p. 779) DeleteAlias (p. 808) DeleteFunction (p. 818) DeleteFunctionConcurrency (p. 822) GetAlias (p. 832) GetFunction (p. 842) GetFunctionConfiguration (p. 851) GetPolicy (p. 869) ListAliases (p. 883) ListVersionsByFunction (p. 911) PublishVersion (p. 919) PutFunctionConcurrency (p. 930) UpdateAlias (p. 949) UpdateFunctionCode (p. 965)	Função	Nenhum
getAccountSettings (p. 830) listFunctions (p. 894) ListTags (p. 909) TagResource (p. 945)	*	Nenhum

Ação	Recurso	Condição
UntagResource (p. 947)		

Ações de mapeamento da fonte de eventos

Para mapeamentos da fonte do evento, permissões para excluir e atualizar podem ser restritas a uma fonte de evento específica. A condição `lambda:FunctionArn` permite restringir quais funções um usuário pode configurar uma fonte de evento para invocar.

Para essas ações, o recurso é o mapeamento da fonte do evento, portanto, o Lambda fornece uma condição que permite restringir permissões com base na função que o mapeamento da fonte do evento invoca.

Mapeamentos de origem do evento

Ação	Recurso	Condição
DeleteEventSourceMapping (p. 812)	Mapeamento de origem do evento	<code>lambda:FunctionArn</code>
UpdateEventSourceMapping (p. 956)		
CreateEventSourceMapping (p. 786)	*	<code>lambda:FunctionArn</code>
GetEventSourceMapping (p. 837)	*	Nenhum
ListEventSourceMappings (p. 888)		

Ações de camada

Ações de camadas permitem restringir as camadas que um usuário pode gerenciar ou usar com uma função. Ações relacionadas a permissões e uso de camadas atuam em uma versão de uma camada, enquanto `PublishLayerVersion` atua no nome de uma camada. Use ambos com curingas para restringir as camadas com as quais um usuário pode trabalhar por nome.

Note

Observações: o [GetLayerVersion \(p. 861\)](#) A ação abrange também [GetLayerVersionByArn \(p. 864\)](#). O Lambda não oferece suporte para `GetLayerVersionByArn` como uma ação do IAM.

Layers

Ação	Recurso	Condição
AddLayerVersionPermission (p. 771)	Versão da camada	Nenhum
RemoveLayerVersionPermission (p. 940)		
GetLayerVersion (p. 861)		
GetLayerVersionPolicy (p. 867)		
DeleteLayerVersion (p. 826)		
ListLayerVersions (p. 903)	Camada	Nenhum
PublishLayerVersion (p. 915)		

Ação	Recurso	Condição
ListLayers (p. 900)	*	Nenhum

Usar limites de permissões para aplicativos do AWS Lambda

Quando você [cria uma aplicação](#) (p. 196) no console do AWS Lambda, o Lambda aplica um limite de permissões às funções do IAM da aplicação. O limite de permissões limita o escopo da [função de execução](#) (p. 57) criada pelo modelo da aplicação para cada uma de suas funções e quaisquer papéis que sejam adicionados ao modelo. O limite de permissões impede que os usuários com acesso de gravação ao repositório Git do aplicativo escalem as permissões do aplicativo além do escopo de seus próprios recursos.

Os modelos de aplicação no console do Lambda incluem uma propriedade global que aplica um limiar de permissões a todas as funções que eles criam.

```
Globals:  
  Function:  
    PermissionsBoundary: !Sub 'arn:${AWS::Partition}:iam::${AWS::AccountId}:policy/${AppId}-${AWS::Region}-PermissionsBoundary'
```

O limiar limita as permissões dos papéis das funções. É possível adicionar permissões ao papel de execução de uma função no modelo, mas essa permissão só será efetiva se também for permitida pelo limiar de permissões. A função que o AWS CloudFormation assume para implantar o aplicativo aplica a utilização do limiar de permissões. Essa função só tem permissão para criar e passar funções que tenham o limiar de permissões do aplicativo anexado.

Por padrão, o limiar de permissões de um aplicativo permite que as funções executem ações nos recursos do aplicativo. Por exemplo, se a aplicação incluir uma tabela do Amazon DynamoDB, o limiar permitirá acesso a qualquer ação de API que possa ser restrita para operar em tabelas específicas com permissões em nível de recurso. Você só pode usar ações que não suportam permissões em nível de recurso se elas forem especificamente permitidas no limiar. Estes incluem Amazon CloudWatch Logs e ações de API do AWS X-Ray para registro e rastreamento.

Example limite de permissões

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Action": [  
        "*"  
      ],  
      "Resource": [  
        "arn:aws:lambda:us-east-2:123456789012:function:my-app-getAllItemsFunction-*",  
        "arn:aws:lambda:us-east-2:123456789012:function:my-app getByIdFunction-*",  
        "arn:aws:lambda:us-east-2:123456789012:function:my-app-putItemFunction-*",  
        "arn:aws:dynamodb:us-east-1:123456789012:table/my-app-SampleTable-*"  
      ],  
      "Effect": "Allow",  
      "Sid": "StackResources"  
    },  
    {  
      "Action": [  
        "logs>CreateLogGroup",  
        "logs>CreateLogStream",  
        "logs>DescribeLogGroups",  
        "logs>PutLogEvents",  
        "xray:Put*"  
      ],  
    }]
```

```
        "Resource": "*",
        "Effect": "Allow",
        "Sid": "StaticPermissions"
    },
    ...
}
```

Para acessar outros recursos ou ações de API, você ou um administrador deve expandir o limiar de permissões para incluir esses recursos. Você também pode precisar atualizar a função de execução ou a função de implantação de um aplicativo para permitir o uso de ações adicionais.

- Limiar de permissões: estenda o limite de permissões da aplicação ao adicionar recursos à aplicação ou se a função de execução precisar de acesso a mais ações. No IAM, adicione recursos ao limiar para permitir o uso de ações de API que oferecem suporte a permissões em nível de recurso no tipo desse recurso. Para ações que não suportam permissões em nível de recurso, adicione-as em uma instrução que não tem escopo para nenhum recurso.
- Função de execução: estenda a função de execução de uma função quando ela precisa usar ações adicionais. No modelo de aplicativo, adicione políticas à função de execução. A interseção de permissões no limiar e função de execução é concedida à função.
- Função de implantação: estenda a função de implantação da aplicação quando ela precisar de permissões adicionais para criar ou configurar recursos. No IAM, adicione políticas à função de implantação do aplicativo. A função de implantação precisa das mesmas permissões de usuário que você precisa para implantar ou atualizar um aplicativo no AWS CloudFormation.

Para obter um tutorial que acompanha a adição de recursos a um aplicativo e estendendo suas permissões, consulte [???](#) (p. 196).

Para obter mais informações sobre limites de permissões, consulte [Limites de permissões para identidades do IAM](#) no Manual do usuário do IAM.

Configurar funções do AWS Lambda

Você pode usar a API do AWS Lambda ou o console para criar funções e configurar as configurações de função. O fluxo de trabalho para criar uma função é diferente para uma função implantada como [imagem de contêiner \(p. 90\)](#) e para uma função implantada como [arquivo .zip \(p. 82\)](#).

Depois de criar a função, você poderá configurar muitas [capacidades e opções de função, \(p. 95\)](#) como permissões, variáveis de ambiente, etiquetas e camadas.

Para manter os segredos fora do seu código de função, armazene-os na configuração da função e leia-os no ambiente de execução durante a inicialização. As [variáveis de ambiente \(p. 99\)](#) são sempre criptografadas em repouso e podem ser criptografadas em no lado do cliente também. Use variáveis de ambiente para tornar seu código de função portátil removendo strings de conexão, senhas e endpoints para recursos externos.

[Versões e aliases \(p. 106\)](#) são recursos secundários que você pode criar para gerenciar a implantação e a chamada de funções. Publique [versões \(p. 106\)](#) da sua função para armazenar seu código e configuração como um recurso separado que não pode ser alterado, e crie um [alias \(p. 108\)](#) que aponte para uma versão específica. Em seguida, você pode configurar seus clientes para invocar um alias de função e atualizar o alias quando quiser apontar o cliente para uma nova versão, em vez de atualizar o cliente.

À medida que você adiciona bibliotecas e outras dependências à sua função, criar e fazer upload de um pacote de implantação pode diminuir a velocidade do desenvolvimento. Use [camadas \(p. 85\)](#) para gerenciar dependências da sua função de forma independente e manter seu pacote de implantação pequena. Você também pode usar camadas para compartilhar suas próprias bibliotecas com outros clientes e usar camadas publicamente disponíveis com suas funções.

Criação de funções do Lambda definidas como arquivos .zip

Quando você cria uma função do Lambda, você empacota o código da função em um[pacote de implantação \(p. 37\)](#). O Lambda oferece suporte a dois tipos de pacotes de implantação:[Imagens do contêiner do \(p. 90\)](#)[Arquivos .zip](#).

Você pode usar o console do Lambda e a API do Lambda para criar uma função definida com um arquivo de arquivo.zip. Você também pode carregar um arquivo.zip atualizado para alterar o código da função.

Note

Não é possível converter uma função de imagem de contêiner existente para usar um arquivo .zip. É necessário criar uma nova função.

Tópicos

- [Criar uma função \(console\) \(p. 82\)](#)
- [Usando o editor de código do console \(p. 83\)](#)
- [Atualizar o código da função \(console\) \(p. 83\)](#)
- [Definindo configurações de execução do \(console\) \(p. 83\)](#)
- [Usar a API do Lambda \(p. 84\)](#)
- [AWS CloudFormation \(p. 84\)](#)

Criar uma função (console)

Ao criar uma função definida com um arquivo de arquivo.zip, você escolhe um modelo de código, a versão de idioma e a função de execução da função. Você adiciona o código da função depois do Lambda cria a função.

Para criar a função

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha Create function.
3. Selecione [Author from scratch \(Começar do zero\)](#) ou [Use a blueprint \(Usar um esquema\)](#) [Como criar sua função do.](#)
4. Em [Basic information \(Informações básicas\)](#), faça o seguinte:
 - a. Em [Function name \(Nome da função\)](#), insira o nome da função.
 - b. para [Tempo de execução](#) Escolha a versão do idioma a ser usada para sua função do.
5. (Opcional) Em [Permissions \(Permissões\)](#), expanda [Change default execution role \(Alterar função de execução padrão\)](#). Crie uma função de execução ou use uma existente.
6. (Opcional) Expand [Advanced settings \(Configurações avançadas\)](#). Você pode escolher [um Configuração de assinatura de código](#) para a função do. Você também pode configurar um (Amazon VPC) para que a função acesse.
7. Escolha Create function.

O Lambda cria a nova função. Agora é possível usar o console do para adicionar o código da função e configurar outros parâmetros e recursos da função.

Usando o editor de código do console

O console cria uma função do Lambda com um único arquivo de origem. Para linguagens de desenvolvimento de scripts, você pode editar esse arquivo e adicionar mais arquivos usando o [editor de códigos \(p. 40\)](#) integrado. Para salvar suas alterações, selecione Save (Salvar). Em seguida, para executar seu código, escolhaTeste.

Note

O console do Lambda usa o AWS Cloud9 para fornecer um ambiente de desenvolvimento integrado no navegador. Você também pode usar o AWS Cloud9 para desenvolver funções do Lambda em seu próprio ambiente. Para obter mais informações, consulte [Working with Lambda Functions](#) no guia do usuário do AWS Cloud9.

Quando você salva seu código de função, o console do Lambda cria um pacote de implantação de arquivos.zip. Ao desenvolver seu código de função fora do console (usando um SDE), você precisa[Criar um pacote de implantação \(p. 517\)](#)para carregar seu código para a função do Lambda.

Atualizar o código da função (console)

Para linguagens de desenvolvimento de scripts, você pode editar o código de sua função no [editor \(p. 40\)](#) de código incorporado. Para adicionar bibliotecas, ou para linguagens incompatíveis com o editor (Java, Go, C#), você deve fazer upload do código da função como um arquivo.zip. Você pode fazer upload do arquivo zip da sua máquina local. Se o arquivo.zip for maior que 50 MB, faça upload do arquivo para a função a partir de um bucket do Amazon S3.

Para carregar código de função como um arquivo.zip

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha a função a ser atualizada e escolha aCódigoGuia.
3. Em Code source (Fonte do código), escolha Upload from (Fazer upload de).
4. Escolha .zip file (Arquivo .zip) e, em seguida, escolha Upload (Fazer upload).
 - No seletor de arquivos, selecione a nova versão da imagem e escolhaAberto, depois, escolhaSave (Salvar).
5. (Alternativa à etapa 4) EscolhaLocalização do Amazon S3.
 - Na caixa de texto, insira o URL do link do S3 do arquivamento de arquivo.zip e, depois, escolhaSave (Salvar).

Definindo configurações de execução do (console)

Ao criar uma função que usa um pacote de implantação de arquivo.zip, você deve especificar o tempo de execução a ser usado e o nome do manipulador de funções.

Para atualizar as configurações de tempo de execução

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha a função a ser atualizada e escolha aCódigoGuia.
3. SobConfigurações de execução, escolhaEdite.
 - a. para oTempo de execução, selecione a versão do tempo de execução.
 - b. Em Handler (Manipulador), especifique o manipulador de sua função.
4. Escolha Save (Salvar).

Usar a API do Lambda

Para criar e configurar uma função que usa um arquivo .zip, use as seguintes operações de API:

- [CreateFunction \(p. 796\)](#)
- [UpdateFunctionCode \(p. 965\)](#)
- [UpdateFunctionConfiguration \(p. 974\)](#)

AWS CloudFormation

Você pode usar AWS CloudFormation para criar uma função do Lambda que usa um arquivo .zip. No seu modelo do AWS CloudFormation, o recurso `AWS::Lambda::Function` especifica a função do Lambda. Para obter descrições das propriedades no recurso `AWS::Lambda::Function`, consulte [AWS::Lambda::Function](#) no Manual do usuário do AWS CloudFormation.

No recurso `AWS::Lambda::Function`, defina as seguintes propriedades para criar uma função definida como um arquivo .zip:

- `AWS::Lambda::Function`
 - `PackageType`: definido como `zip`.
 - `Código` — Insira o nome do bucket do Amazon S3 e o nome do arquivo.zip na caixa `S3Bucket` e `S3Key`. Campos. Para Node.js ou Python, você pode fornecer código-fonte inline da sua função do Lambda.
 - `Tempo de execução` — Defina o valor do runtime.

Criar e compartilhar camadas do Lambda

Uma camada do Lambda é um arquivo.zip arquivo que pode conter código ou dados adicionais. Uma camada pode conter bibliotecas, um [tempo de execução personalizado \(p. 255\)](#), dados ou arquivos de configuração. As camadas promovem o compartilhamento de código e a separação de responsabilidades para que você possa iterar mais rapidamente na escrita da lógica de negócios.

Você pode usar camadas somente com funções do Lambda [implantadas como um arquivo.zip \(p. 37\)](#). Para funções [definidas como uma imagem de contêiner \(p. 266\)](#), você empacota seu tempo de execução preferido e todas as dependências de código ao criar a imagem de contêiner. Para obter mais informações, consulte a publicação [Working with Lambda layers and extensions in container images](#) no Blog de computação da AWS.

É possível criar camadas usando o console do Lambda, a API do LambdaAWS CloudFormation, ou oAWS Serverless Application Model(AWS SAM). Para obter mais informações sobre como criar camadas comAWS SAM, consulte[Trabalhar com camadas](#)noAWS Serverless Application ModelGuia do desenvolvedor.

Seções

- [Criar conteúdo de camada \(p. 85\)](#)
- [Compilar o arquivo .zip para sua camada \(p. 85\)](#)
- [Incluir dependências da biblioteca em uma camada \(p. 85\)](#)
- [Instruções específicas de linguagem \(p. 87\)](#)
- [Criar uma camada \(p. 87\)](#)
- [Excluir uma versão de camada \(p. 88\)](#)
- [Configurar permissões de camada \(p. 88\)](#)
- [Usar o AWS CloudFormation com camadas \(p. 89\)](#)

Criar conteúdo de camada

Ao criar uma camada, agrupe todo o conteúdo em um arquivo .zip. Você carrega o arquivo .zip para sua camada a partir do Amazon Simple Storage Service (Amazon S3) ou da sua máquina local. O Lambda extrai o conteúdo da camada para o /optAo configurar o ambiente de execução para a função.

Compilar o arquivo .zip para sua camada

Você cria seu código de camada em um arquivo.zip usando o mesmo procedimento que você faria para um pacote de implantação da função. Se sua camada incluir bibliotecas de código nativo, você deve compilar e criar essas bibliotecas usando uma máquina de desenvolvimento Linux para que os binários sejam compatíveis com o [Amazon Linux \(p. 214\)](#).

Uma maneira de garantir que você empacote bibliotecas corretamente para o Lambda é usar o [AWS Cloud9](#). Para obter mais informações, consulte [Using Lambda layers to simplify your development process](#) no Blog de computação da AWS.

Incluir dependências da biblioteca em uma camada

Para cada [tempo de execução do Lambda \(p. 214\)](#), a variável PATH inclui pastas específicas no diretório /opt. Se você definir a mesma estrutura de pastas em seu arquivo .zip de camada, o código da função poderá acessar o conteúdo da camada sem a necessidade de especificar o caminho.

A tabela a seguir lista os caminhos de pasta compatíveis com cada tempo de execução.

Caminhos de camada para cada tempo de execução do Lambda

Tempo de execução	Caminho		
Node.js	nodejs/node_modules		
	nodejs/node14/node_modules (NODE_PATH)		
Python	python		
	python/lib/python3.9/site-packages(diretórios do site)		
Java	java/lib (CLASSPATH)		
Ruby	ruby/gems/2.7.0 (GEM_PATH)		
	ruby/lib (RUBYLIB)		
Todos os tempos de execução	bin (PATH)		
	lib (LD_LIBRARY_PATH)		

Os exemplos a seguir mostram como você pode estruturar as pastas no seu arquivo .zip da camada.

Node.js

Example estrutura de arquivos do AWS X-Ray SDK for Node.js

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

Python

Example estrutura de arquivos para a biblioteca Pillow

```
pillow.zip  
# python/PIL  
# python/Pillow-5.3.0.dist-info
```

Ruby

Example estrutura de arquivos da gem JSON

```
json.zip  
# ruby/gems/2.5.0/  
| build_info  
| cache  
| doc  
| extensions  
| gems  
| # json-2.1.0  
# specifications  
# json-2.1.0.gemspec
```

Java

Example estrutura de arquivos do arquivo JAR do Jackson

```
jackson.zip  
# java/lib/jackson-core-2.2.3.jar
```

All

Example estrutura de arquivos da biblioteca jq

```
jq.zip  
# bin/jq
```

Para obter mais informações sobre as configurações de caminhos no ambiente de execução do Lambda, consulte [Variáveis de ambiente com tempo de execução definido \(p. 102\)](#).

Instruções específicas de linguagem

Para obter instruções específicas de linguagem sobre como criar um arquivo .zip, consulte os tópicos a seguir.

- [Implantar funções do Lambda em Node.js com arquivos .zip \(p. 517\)](#)
- [Implantar funções do Lambda em Python com arquivos .zip \(p. 543\)](#)
- [Implantar funções do Lambda em Ruby com arquivos .zip \(p. 571\)](#)
- [Implantar funções do Lambda em Java com arquivos .zip ou JAR \(p. 599\)](#)
- [Implantar funções do Lambda em Go com arquivos .zip \(p. 640\)](#)
- [Implantar funções do Lambda em C# com arquivos .zip \(p. 668\)](#)
- [Implantar funções do Lambda para PowerShell com arquivos .zip \(p. 694\)](#)

Criar uma camada

Você pode criar camadas usando o console do Lambda ou a API do Lambda.

As camadas podem ter uma ou mais versões. Quando você cria uma camada, o Lambda define a versão da camada para a versão 1. Você pode configurar permissões em uma versão de camada existente, mas para atualizar o código ou fazer outras alterações de configuração, você deve criar uma nova versão da camada.

Para criar uma camada (console)

1. Abra a [página Layers](#) (Camadas) do console do Lambda.
2. Escolha Create layer (Criar camada).
3. Em Layer configuration (Configuração de camada), insira um nome para sua camada em Name (Nome).
4. (Opcional) Em Description (Descrição), insira uma descrição para a sua camada.
5. Para fazer upload do código da camada, siga um destes procedimentos:
 - Para carregar um arquivo.zip do seu computador, escolha Upload a .zip file (Fazer upload de um arquivo .zip). Selecione Upload (Fazer upload) para escolher seu arquivo .zip local.
 - Para fazer upload de um arquivo do Amazon S3, escolha Para fazer upload de um arquivo do Amazon S3. Então, em Amazon S3 link URL (URL do link do Amazon S3), insira um link para o arquivo.

6. (Opcional) Em Compatible runtimes (Tempos de execução compatíveis), escolha até 15 tempos de execução.
7. (Opcional) Em License (Licença), insira todas as informações necessárias sobre licença.
8. Escolha Create (Criar).

Para criar uma camada (API)

Para criar uma camada, use o comando `publish-layer-version` com um nome, descrição, arquivo .zip e uma lista de [tempos de execução \(p. 214\)](#) que sejam compatíveis com a camada. A lista de tempos de execução é opcional.

```
aws lambda publish-layer-version --layer-name my-layer --description "My layer" \
--license-info "MIT" --content S3Bucket=lambda-layers-us-
east-2-123456789012,S3Key=layer.zip \
--compatible-runtimes python3.6 python3.7 python3.8
```

Você deve ver saída semelhante a:

```
{
  "Content": {
    "Location": "https://awslambda-us-east-2-layers.s3.us-east-2.amazonaws.com/
snapshots/123456789012/my-layer-4aaa2fbb-ff77-4b0a-ad92-5b78a716a96a?
versionId=27iWyA73cCAYqyH...",
    "CodeSha256": "tv9jJO+rPbXUUXuRKi7CwHzKtLDkDRJLB3cC3Z/ouXo=",
    "CodeSize": 169
  },
  "LayerArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer",
  "LayerVersionArn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer:1",
  "Description": "My layer",
  "CreatedDate": "2018-11-14T23:03:52.894+0000",
  "Version": 1,
  "LicenseInfo": "MIT",
  "CompatibleRuntimes": [
    "python3.6",
    "python3.7",
    "python3.8"
  ]
}
```

Note

Cada vez que acionar a `publish-layer-version`, você cria uma nova versão dessa camada.

Excluir uma versão de camada

Para excluir uma versão de camada, use o comando `delete-layer-version`.

```
aws lambda delete-layer-version --layer-name my-layer --version-number 1
```

Ao excluir uma versão de camada, você não pode mais configurar uma função do Lambda para usá-la. No entanto, todas as funções que já usem a versão continua a ter acesso a ela. Os números de versão nunca são reutilizados para um nome de camada.

Configurar permissões de camada

Por padrão, uma camada que você cria é privada para sua conta da AWS. No entanto, você tem a opção de compartilhar a camada com outras contas ou torná-la pública.

Para conceder permissão de uso de camadas a outra conta, adicione uma instrução à política de permissões da versão da camada usando o comando add-layer-version-permission. Em cada instrução, você pode conceder permissão a uma única conta, a todas as contas ou a uma organização.

```
aws lambda add-layer-version-permission --layer-name xray-sdk-nodejs --statement-id xaccount \
--action lambda:GetLayerVersion --principal 210987654321 --version-number 1 --output text
```

Você deve ver saída semelhante a:

```
e210ffdc-e901-43b0-824b-5fc0dd26d16 {"Sid":"xaccount","Effect":"Allow","Principal":{"AWS":"arn:aws:iam::210987654321:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:lambda:us-east-2:123456789012:layer:xray-sdk-nodejs:1"}
```

As permissões se aplicam apenas a uma única versão de camada. Repita o processo sempre que criar uma nova versão da camada.

Para obter mais exemplos, consulte [Conceder acesso de camada a outras contas \(p. 65\)](#).

Usar o AWS CloudFormation com camadas

Você pode usar o AWS CloudFormation para criar uma camada e associá-la à sua função do Lambda. O modelo de exemplo a seguir cria uma camada chamada blank-nodejs-lib e a anexa à função do Lambda usando a propriedade Layers.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: A Lambda application that calls the Lambda API.
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs12.x
      CodeUri: function/
      Description: Call the Lambda API
      Timeout: 10
      # Function's execution role
      Policies:
        - AWSLambdaBasicExecutionRole
        - AWSLambda_ReadOnlyAccess
        - AWSXrayWriteOnlyAccess
      Tracing: Active
      Layers:
        - !Ref libs
  libs:
    Type: AWS::Serverless::LayerVersion
    Properties:
      LayerName: blank-nodejs-lib
      Description: Dependencies for the blank sample app.
      Content:
        S3Bucket: my-bucket-region-123456789012
        S3Key: layer.zip
      CompatibleRuntimes:
        - nodejs12.x
```

Criar funções do Lambda definidas como imagens de contêiner

Quando você cria uma função do Lambda, você usa um [pacote de implantação \(p. 37\)](#) Para implantar o código da função. O Lambda oferece suporte a dois tipos de pacotes de implantação:[Arquivos .zip \(p. 82\)](#)Imagens do contêiner e.

Você pode usar o console do Lambda e a API do Lambda para criar uma função definida como uma imagem de contêiner, atualizar e testar o código da imagem e configurar outras configurações de função.

Note

Não é possível converter uma função de imagem de contêiner existente para usar um arquivo .zip. É necessário criar uma nova função.

Quando você seleciona uma imagem usando uma tag de imagem, o Lambda traduz a tag para o resumo de imagem subjacente. Para recuperar o resumo da imagem, use a operação da API [GetFunctionConfiguration \(p. 851\)](#). Para atualizar a função para uma versão de imagem mais recente, você deve usar o console do Lambda para [atualizar o código da função \(p. 92\)](#) ou usar a operação da API [UpdateFunctionCode \(p. 965\)](#). Operações de configuração, como [UpdateFunctionConfiguration \(p. 974\)](#), não atualizam a imagem do contêiner da função.

Note

No Amazon ECR, se você reatribuir a tag de imagem para outra imagem, o Lambda não atualizará a versão da imagem.

Tópicos

- [Versão da função \\$LATEST \(p. 90\)](#)
- [Implantação de imagens de contêiner \(p. 91\)](#)
- [Permissões do Amazon ECR \(p. 91\)](#)
- [Substituir as configurações de contêiner \(p. 91\)](#)
- [Criar uma função \(console\) \(p. 91\)](#)
- [Atualizar o código da função \(console\) \(p. 92\)](#)
- [Substituir os parâmetros da imagem \(console\) \(p. 93\)](#)
- [Usar a API do Lambda \(p. 93\)](#)
- [AWS CloudFormation \(p. 94\)](#)

Versão da função \$LATEST

Ao publicar uma versão da função, o código e a maioria das configurações são bloqueados, para manter uma experiência consistente para os usuários dessa versão. Você pode alterar o código e muitas configurações apenas na versão não publicada da função. Por padrão, o console exibe informações de configuração para a versão não publicada da função. Para exibir as versões de uma função, escolha Qualifiers (Qualificadores). A versão não publicada é chamada \$LATEST.

Observe que o Amazon Elastic Container Registry (Amazon ECR) também usa uma etiqueta mais recente para denotar a versão mais recente da imagem do contêiner. Tenha cuidado para não confundir esta tag com a versão da função \$LATEST .

Para obter mais informações sobre como gerenciar versões, consulte [Versões da função do Lambda \(p. 106\)](#).

Implantação de imagens de contêiner

Quando você implanta código como uma imagem de contêiner em uma função do Lambda, a imagem passa por um processo de otimização para execução do Lambda. Esse processo pode levar alguns segundos, durante os quais a função ficará no estado pendente. Quando o processo de otimização é concluído, a função entra no estado ativo.

Permissões do Amazon ECR

Para que sua função acesse a imagem do contêiner no Amazon ECR, você pode adicionar `ecr:BatchGetImage` e `ecr:GetDownloadUrlForLayer` para seu repositório do Amazon ECR. O exemplo a seguir mostra a política mínima:

```
{  
    "Sid": "LambdaECRImageRetrievalPolicy",  
    "Effect": "Allow",  
    "Principal": {  
        "Service": "lambda.amazonaws.com"  
    },  
    "Action": [  
        "ecr:BatchGetImage",  
        "ecr:GetDownloadUrlForLayer"  
    ]  
}
```

Para obter mais informações sobre permissões do repositório do Amazon ECR, consulte [Repository policies](#) no Guia do usuário do Amazon Elastic Container Registry.

Se o repositório do Amazon ECR não incluir essas permissões, o Lambda adicionará `ecr:BatchGetImage` e `ecr:GetDownloadUrlForLayer` para as permissões do repositório de imagens do contêiner. O Lambda pode adicionar essas permissões somente se o principal chamando o Lambda tiver `ecr:getRepositoryPolicy` e `ecr:setRepositoryPolicy` permissões.

Para visualizar ou editar suas permissões de repositório do Amazon ECR, siga as instruções em [Definir uma instrução da política do repositório](#) no Amazon Elastic Container Registry Guide.

Substituir as configurações de contêiner

Você pode usar o console do Lambda ou a API do Lambda para substituir as seguintes configurações de imagem de contêiner:

- ENTRYPPOINT: especifica o caminho absoluto do ponto de entrada para a aplicação.
- CMD: especifica parâmetros adicionais que você deseja transmitir com ENTRYPPOINT.
- WORKDIR: especifica o caminho absoluto do diretório de trabalho.
- ENV: especifica uma variável de ambiente para a função do Lambda.

Todos os valores fornecidos no console do Lambda ou na API do Lambda substituem os valores no [Dockerfile](#) (p. 268).

Criar uma função (console)

Para criar uma função definida como uma imagem de contêiner, primeiro é necessário [criar a imagem](#) (p. 267) e, em seguida, armazenar a imagem no repositório do Amazon ECR.

Para criar a função

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha Create function.
3. Escolha a opção Container image (Imagem de contêiner).
4. Em Basic information (Informações básicas), faça o seguinte:
 - a. Em Function name (Nome da função), insira o nome da função.
 - b. Para Container image URI (URI de imagem de contêiner), insira o URI da imagem do Amazon ECR.
 - Ou, para a imagem em um repositório do Amazon ECR, escolha Browse images (Procurar imagens). Selecione o repositório do Amazon ECR na lista suspensa e, em seguida, selecione a imagem.
 - c. (Opcional) Para substituir as configurações incluídas no Dockerfile, expanda Container image overrides (Substituições de imagem de contêiner). Você pode substituir qualquer uma das seguintes configurações:
 - Para Entrypoint (Ponto de entrada), insira o caminho completo do executável de tempo de execução. O exemplo a seguir mostra um ponto de entrada para uma função Node.js:

```
"/usr/bin/npx", "aws-lambda-ric"
```

- Para Command (Comando), insira os parâmetros adicionais a transferir para a imagem com Entrypoint (Ponto de entrada). O exemplo a seguir mostra um comando para uma função Node.js:

```
"app.handler"
```

- Em Working directory (Diretório de trabalho), insira o caminho completo do diretório de trabalho para a função. O exemplo a seguir mostra o diretório de trabalho para uma imagem básica da AWS para o Lambda:

```
"/var/task"
```

Note

Para as configurações de substituição, certifique-se colocar cada string entre aspas ("").

5. (Opcional) Em Permissions (Permissões), expanda Change default execution role (Alterar função de execução padrão). Em seguida, escolha criar uma nova Execution role (Função de execução) ou usar uma função existente.
6. Escolha Create function.

Atualizar o código da função (console)

Depois de implantar uma imagem de contêiner em uma função, a imagem é somente leitura. Para atualizar o código da função, primeiro é necessário implantar uma nova versão da imagem. [Crie uma nova versão da imagem \(p. 267\)](#) e armazene-a no repositório do Amazon ECR.

Para configurar a função para usar uma imagem de contêiner atualizada

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha a função a ser atualizada.

3. Em Image (Imagem), escolha Deploy new image (Implantar nova imagem).
4. Escolha Browse images (Procurar imagens).
5. Na caixa de diálogo Select container image (Selecionar imagem do contêiner), selecione o repositório do Amazon ECR na lista suspensa e, em seguida, selecione a nova versão da imagem.
6. Escolha Save (Salvar).

Substituir os parâmetros da imagem (console)

Você pode usar o console do Lambda para substituir os valores de configuração na imagem do contêiner.

Para substituir os valores de configuração na imagem do contêiner

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha a função a ser atualizada.
3. Sob a Image configuration (Configuração da imagem), escolha Edit (Editar).
4. Insira novos valores para qualquer uma das configurações de substituição e escolha Save (Salvar).
5. (Opcional) Para adicionar ou substituir variáveis de ambiente, em Environment variables (Variáveis de ambiente), escolha Edit (Editar).

Para obter mais informações, consulte [the section called “Variáveis de ambiente” \(p. 99\)](#).

Usar a API do Lambda

Para gerenciar funções definidas como imagens de contêiner, use as seguintes operações de API:

- [CreateFunction \(p. 796\)](#)
- [UpdateFunctionCode \(p. 965\)](#)
- [UpdateFunctionConfiguration \(p. 974\)](#)

Para criar uma função definida como imagem de contêiner, use o comando `create-function`. Defina o `package-type` para `Image` e especifique o URI de sua imagem de contêiner usando o parâmetro `code`. Observe que você deve criar a função a partir da mesma conta que o registro de contêiner no Amazon EFS.

```
aws lambda create-function --region sa-east-1 --function-name my-function \
--package-type Image \
--code ImageUri=<ECR Image URI> \
--role arn:aws:iam::123456789012:role/lambda-ex
```

Para atualizar o código da função, use o comando `update-function-code`. Especifique o local da imagem do contêiner usando o parâmetro `image-uri`.

Note

Você não pode alterar o `package-type` de uma função.

```
aws lambda update-function-code --region sa-east-1 --function-name my-function \
--image-uri <ECR Image URI> \
```

Para atualizar os parâmetros da função, use a operação `update-function-configuration`. Especifique `EntryPoint` e `Command` como matrizes de strings, e `WorkingDirectory` como uma string.

```
aws lambda update-function-configuration --function-name my-function \
--image-config '{"EntryPoint": ["/usr/bin/npx", "aws-lambda-ric"], \
"Command": ["app.handler"], \
"WorkingDirectory": "/var/task"}'
```

AWS CloudFormation

Você pode usar o AWS CloudFormation para criar funções do Lambda definidas como imagens de contêiner. No seu modelo do AWS CloudFormation, o recurso `AWS::Lambda::Function` especifica a função do Lambda. Para obter descrições das propriedades no recurso `AWS::Lambda::Function`, consulte [AWS::Lambda::Function](#) no Manual do usuário do AWS CloudFormation .

No recurso `AWS::Lambda::Function`, defina as seguintes propriedades para criar uma função definida como uma imagem de contêiner:

- `AWS::Lambda::Function`
 - `PackageType`: definido como `Image`.
 - `Code`: digite seu URI de imagem de contêiner no campo `ImageUri`.
 - `ImageConfig`: (opcional) substitua as propriedades de configuração da imagem do contêiner.

A propriedade `ImageConfig` em na `AWS::Lambda::Function` contém os seguintes campos:

- `Command`: especifica os parâmetros com os quais você deseja transmitir o `EntryPoint`.
- `EntryPoint`: especifica o ponto de entrada da aplicação.
- `WorkingDirectory`: especifica o diretório de trabalho.

Note

Se você declarar uma propriedade de `ImageConfig` no seu modelo do AWS CloudFormation, deverá fornecer valores para todas as três propriedades de `ImageConfig`.

Para obter mais informações, consulte [ImageConfig](#)no AWS CloudFormation Guia do usuário do.

Configurar as opções da função do Lambda

Depois de criar uma função, você pode configurar recursos adicionais para a função, como gatilhos, acesso à rede e acesso ao sistema de arquivos. Você também pode ajustar recursos associados à função, como memória e simultaneidade. Essas configurações se aplicam a funções definidas como arquivos.zip e a funções definidas como imagens de contêiner.

Você também pode criar e editar eventos de teste para testar sua função usando o console.

Para ver as práticas recomendadas de configuração da função, consulte [Configuração da função \(p. 212\)](#).

Seções

- [Versões da função \(p. 95\)](#)
- [Usar a visão geral da função \(p. 95\)](#)
- [Defina as funções \(console\) \(p. 96\)](#)
- [Configurar funções \(API\) \(p. 84\)](#)
- [Configurar a memória da função \(console\) \(p. 97\)](#)
- [Aceitar recomendações de memória de função \(console\) \(p. 97\)](#)
- [Configurando gatilhos \(console\) \(p. 98\)](#)
- [Funções de teste \(console\) \(p. 98\)](#)

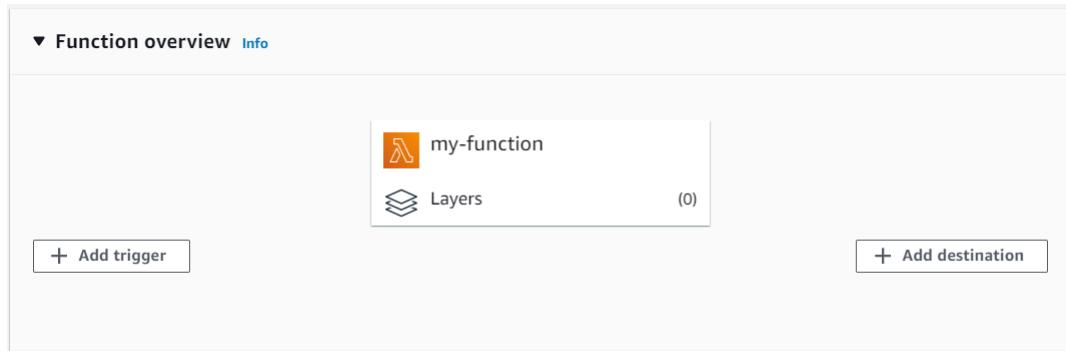
Versões da função

Uma função tem uma versão não publicada e pode ter versões e aliases publicados. Por padrão, o console exibe informações de configuração para a versão não publicada da função. Alterar a versão não publicada quando actualiza o código e a configuração da função.

Uma versão publicada é um snapshot do código de função e configuração que não pode ser alterado (exceto para alguns itens de configuração relevantes para uma versão de função, como simultaneidade provisionada).

Usar a visão geral da função

A Function overview (Visão geral da função) mostra uma visualização da sua função e seus recursos upstream e downstream. Você pode usá-la para acessar diretamente a configuração de acionadores e destinos. Você pode usá-la para acessar diretamente a configuração da camada para funções definidas como arquivos.zip.



Defina as funções (console)

Para as configurações de função a seguir, é possível alterar as configurações somente para a versão não publicada da função. No console do, a função configuração fornece as seguintes seções:

- Configuração geral— Definamemory (p. 97) ou optar pela AWSCompute Optimizer (p. 97). Você também pode configurar o tempo limite da função e a função de execução.
- Permissões— Configura a função de execução e outrospermissions (p. 56).
- Environment variables (Variáveis de ambiente): pares de chave-valor que o Lambda define no ambiente de execução. Use variáveis de ambiente (p. 99) para estender a configuração da função fora do código.
- Etiquetas: pares de chave-valor que o Lambda anexa ao recurso da função. Use etiquetas (p. 149) para organizar as funções do Lambda em grupos de relatórios de custo e filtragem no console do Lambda.

As tags se aplicam a toda a função, incluindo todas as versões e aliases.

- Virtual Private Cloud (VPC): se a função precisa de acesso à rede para recursos que não estão disponíveis pela Internet, configure-a para se conectar a uma Virtual Private Cloud (VPC) (p. 124).
- Ferramentas de monitoramento e operações— configure o CloudWatch e outrosFerramentas de monitoramento.
- Concurrency (Simultaneidade): reserve simultaneidade para uma função (p. 113) para definir o número máximo de execuções simultâneas para uma função. Provisione simultaneidade para garantir que uma função possa ser escalada sem flutuações na latência. A simultaneidade reservada se aplica a toda a função, incluindo todas as versões e aliases.

Você pode configurar as opções a seguir em uma função, uma versão de função ou um alias.

- Gatilhos— Definagatilhos do (p. 98).
- Destinos— Definadestinos (p. 164) Para invocações assíncronas.
- Invocação assíncrona—Configurar o comportamento de tratamento de erros (p. 161) para reduzir o número de tentativas que o Lambda tenta ou a quantidade de tempo que os eventos não processados permanecem na fila antes que o Lambda os descarta. Configure uma fila de mensagens mortas (p. 167) para reter eventos excluídos.
- Assinatura de código: para usar Assinatura de código (p. 144) com sua função, configure a função para incluir uma configuração de assinatura de código.
- Database proxies (Proxies de banco de dados): crie um proxy de banco de dados (p. 134) para funções que usam uma instância de banco de dados ou cluster do Amazon RDS.
- Sistemas de arquivos— Connect sua função a um sistema de arquivos (p. 139).
- Máquinas de estado: use uma máquina de estado para orquestrar e aplicar o tratamento de erros à sua função.

O console fornece guias separadas para configurar aliases e versões:

- Aliases do— Um alias é um recurso nomeado que mapeia para uma versão de função. Você pode alterar um alias para mapear para uma versão de função diferente.
- Versões do :— O Lambda atribui um novo número de versão cada vez que você publica sua função. Para obter mais informações sobre como gerenciar versões, consulte Versões da função do Lambda (p. 106).

Você pode configurar os seguintes itens para uma versão de função publicada:

- Gatilhos

- Destinos
- Simultaneidade provisionada
- Invocação assíncrona
- Proxies de banco de dados

Configurar funções (API)

Para configurar funções com a API do Lambda, use as seguintes ações:

- [UpdateFunctionCode \(p. 965\)](#): atualize o código da função.
- [UpdateFunctionConfiguration \(p. 974\)](#): atualize as configurações específicas da versão.
- [TagResource \(p. 945\)](#): etiquete uma função.
- [AddPermission \(p. 775\)](#): modifique a [política baseada em recursos \(p. 62\)](#) de uma função, de uma versão ou de um alias.
- [PutFunctionConcurrency \(p. 930\)](#): configure a simultaneidade reservada da função.
- [PublishVersion \(p. 919\)](#): crie uma versão imutável com a configuração e o código atual.
- [CreateAlias \(p. 779\)](#): crie aliases para versões da função.
- [PutFunctionEventInvokeConfig](#): configure o tratamento de erros para invocação assíncrona.

Configurar a memória da função (console)

O Lambda aloca capacidade da CPU na proporção da quantidade de memória configurada. Memória é a quantidade de memória disponível para a função do Lambda no tempo de execução. Você pode aumentar a memória e a potência da CPU alocada para a função usando a configuração Memory MB (Memória (MB)). Para configurar a memória para sua função, defina um valor entre 128 MB e 10.240 MB em incrementos de 1 MB. Com 1.769 MB, uma função tem o equivalente a um vCPU (um vCPU-segundo de créditos por segundo).

Você pode configurar a memória da sua função no console do Lambda.

Para atualizar a memória de uma função

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Na página de configuração da função, no Configuração geral, escolha **Edita**.
4. Em **Memory (MB) (Memória (MB))**, defina um valor de 128 MB como 10.240 MB.
5. Escolha **Save (Salvar)**.

Aceitar recomendações de memória de função (console)

Se você tiver permissões de administrador no AWS Identity and Access Management (IAM), será possível optar por receber recomendações de memória de funções do Lambda do AWS Compute Optimizer. Para obter instruções sobre como aceitar recomendações de memória para sua conta ou organização, consulte [Opting in your account](#) no Manual do usuário do AWS Compute Optimizer.

Quando você tiver optado e sua [função do Lambda atender aos requisitos do Compute Optimizer](#), será possível visualizar e aceitar recomendações de memória de funções do Compute Optimizer no console do Lambda.

Para aceitar uma recomendação de memória de função

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Na página de configuração da função, no Configuração geral, escolha **Edite**.
4. Em Memory (MB) (Memória [MB]), no alerta de memória, escolha **Update (Atualizar)**.
5. Escolha **Save (Salvar)**.

Configurando gatilhos (console)

Você pode configurar outras AWS para acionar sua função cada vez que um evento específico ocorre.

Para obter detalhes sobre como os serviços acionam funções do Lambda, consulte [Usar o AWS Lambda com outros serviços \(p. 277\)](#).

Para adicionar um acionador à sua função

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha a função a ser atualizada.
3. Em Function overview (Visão geral da função), escolha **Add trigger (Adicionar gatilho)**.
4. Na lista suspensa de triggers, escolha um trigger. O console exibe campos de configuração adicionais necessários para este gatilho.
5. Escolha **Adicionar**.

Funções de teste (console)

Você pode criar eventos de teste para sua função a partir do [Teste Guia](#).

Para criar um evento de teste

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha a função a ser testada e escolha **Teste**.
3. Under **Evento de teste**, selecione **Novo evento**.
4. Selecione um **Template (Modelo)**.
5. Em **Name (Nome)**, insira um nome para o teste. Na caixa de entrada de texto, insira o evento de teste JSON.
6. Selecione **Save changes**.

Eventos de teste salvos também estão disponíveis no [Código](#), sob a guia [Teste Menu](#). Depois de criar um ou mais eventos de teste, você pode chamar sua função usando um de seus testes como um evento.

Para testar a função do

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha a função a ser testada e escolha **Teste**.
3. Under **Evento de teste**, selecione **Eventos salvos** e selecione o evento que deseja usar.
4. Escolha **Test** (Testar).
5. Expand **the Resultado da execução** para exibir detalhes sobre o teste.

Usar variáveis de ambiente do AWS Lambda

É possível usar variáveis de ambiente para ajustar o comportamento da função sem atualizar o código. Uma variável de ambiente é um par de strings armazenadas na configuração específica da versão de uma função. O tempo de execução do Lambda disponibiliza variáveis de ambiente para o código e define variáveis de ambiente adicionais que contêm informações sobre a função e a solicitação de chamada.

Note

Para aumentar a segurança do banco de dados, recomendamos que você use o AWS Secrets Manager em vez de variáveis de ambiente para armazenar credenciais de banco de dados. Para obter mais informações, consulte [Configurar o acesso ao banco de dados para uma função do Lambda](#).

As variáveis de ambiente não são avaliadas antes da invocação da função. Qualquer valor definido é considerado uma string literal e não expandido. Execute a avaliação da variável no seu código de função.

Seções

- [Configurar variáveis de ambiente \(p. 99\)](#)
- [Como configurar variáveis de ambiente com a API \(p. 100\)](#)
- [Exemplo de cenário para variáveis de ambiente \(p. 100\)](#)
- [Recupere variáveis de ambiente \(p. 101\)](#)
- [Variáveis de ambiente com tempo de execução definido \(p. 102\)](#)
- [Proteger variáveis de ambiente \(p. 103\)](#)
- [Modelos e código de exemplo \(p. 105\)](#)

Configurar variáveis de ambiente

Você define variáveis de ambiente na versão não publicada da sua função. Quando você publica uma versão, as variáveis de ambiente são bloqueadas para aquela versão com outras [configurações específicas da versão \(p. 95\)](#).

Para criar uma variável de ambiente na sua função, defina uma chave e um valor. A sua função usa o nome da chave para recuperar o valor da variável de ambiente.

Para definir variáveis de ambiente no console do Lambda

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e, em seguida, Environment variables (Variáveis de ambiente).
4. Em Environment variables (Variáveis de ambiente), selecione Edit (Editar).
5. Escolha Add environment variable (Adicionar variável de ambiente).
6. Insira um par de chave e valor.

Requirements

- As chaves começam com uma letra e têm pelo menos dois caracteres.
- As chaves e os valores contêm somente letras, números e o caractere de sublinhado (_).
- As chaves não são [reservadas pelo Lambda \(p. 102\)](#).
- O tamanho total de todas as variáveis de ambiente não excede 4 KB.

7. Escolha Save (Salvar).

Como configurar variáveis de ambiente com a API

Para gerenciar variáveis de ambiente com a AWS CLI ou o AWS SDK, use as operações de API a seguir.

- [UpdateFunctionConfiguration \(p. 974\)](#)
- [GetFunctionConfiguration \(p. 851\)](#)
- [CreateFunction \(p. 796\)](#)

O exemplo a seguir define duas variáveis de ambiente em uma função denominada `my-function`.

```
aws lambda update-function-configuration --function-name my-function \
--environment "Variables={BUCKET=my-bucket,KEY=file.txt}"
```

Quando você aplica variáveis de ambiente com o comando `update-function-configuration`, todo o conteúdo da estrutura `Variables` é substituído. Para manter variáveis de ambiente existentes ao adicionar uma nova, inclua todos os valores existentes em sua solicitação.

Para obter a configuração atual, use o comando `get-function-configuration`.

```
aws lambda get-function-configuration --function-name my-function
```

Você deve ver a saída a seguir:

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
  "Runtime": "nodejs12.x",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Environment": {
    "Variables": {
      "BUCKET": "my-bucket",
      "KEY": "file.txt"
    }
  },
  "RevisionId": "0894d3c1-2a3d-4d48-bf7f-abade99f3c15",
  ...
}
```

Para garantir que os valores não mudam entre a leitura da configuração e a atualização desta, você pode passar o ID de revisão da saída de `get-function-configuration` como um parâmetro para `update-function-configuration`.

Para configurar a chave de criptografia de uma função, defina a opção `KMSKeyARN`.

```
aws lambda update-function-configuration --function-name my-function \
--kms-key-arn arn:aws:kms:us-east-2:123456789012:key/055efbb4-xmpl-4336-
ba9c-538c7d31f599
```

Exemplo de cenário para variáveis de ambiente

Você pode usar variáveis de ambiente para personalizar o comportamento da função no ambiente de teste e no ambiente de produção. Por exemplo, você pode criar duas funções com o mesmo código, mas configurações diferentes. Uma função se conecta a um banco de dados de teste e a outra se conecta a um banco de dados de produção. Nessa situação, você usa variáveis de ambiente para informar à função o nome do host e outros detalhes de conexão do banco de dados.

O exemplo a seguir mostra como definir o host do banco de dados e o nome do banco de dados como variáveis de ambiente.

ENVIRONMENT	DEVELOPMENT	Remove
databaseHost	lambdadb	Remove
databaseName	rd1owwlydynnm5.cuovuayfg087	Remove
Key	Value	Remove

Se quiser que o ambiente de teste gere mais informações de depuração do que o ambiente de produção, você poderá definir uma variável de ambiente para configurar o ambiente de teste para usar um registro em log mais detalhado ou um rastreamento mais detalhado.

Recupere variáveis de ambiente

Para recuperar variáveis de ambiente em seu código de função, use o método padrão para a sua linguagem de programação.

Node.js

```
let region = process.env.AWS_REGION
```

Python

```
import os
region = os.environ['AWS_REGION']
```

Note

Em alguns casos, talvez seja necessário usar o seguinte 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
```

O Lambda armazena as variáveis de ambiente de forma segura, criptografando-as em repouso. É possível [configurar o Lambda para usar uma chave de criptografia diferente \(p. 103\)](#), criptografar valores de variáveis de ambiente no lado do cliente ou definir variáveis de ambiente em um modelo do AWS CloudFormation com o AWS Secrets Manager.

Variáveis de ambiente com tempo de execução definido

Os [tempos de execução \(p. 214\)](#) do Lambda definem diversas variáveis de ambiente durante a inicialização. A maioria das variáveis de ambiente fornece informações sobre a função ou o tempo de execução. As chaves para essas variáveis de ambiente são reservadas e não podem ser definidas na configuração de sua função.

Variáveis de ambiente reservadas

- `_HANDLER`: o local do handler configurado na função.
- `_X_AMZN_TRACE_ID`— [OCabeçalho do X-Ray \(p. 477\)](#).
- `AWS_REGION`: a região da AWS onde a função do Lambda é executada.
- `AWS_EXECUTION_ENV`: o [identificador de tempo de execução \(p. 214\)](#), prefixado por `AWS_Lambda_`, por exemplo, `AWS_Lambda_java8`.
- `AWS_LAMBDA_FUNCTION_NAME`: o nome da função.
- `AWS_LAMBDA_FUNCTION_MEMORY_SIZE`: a quantidade de memória disponível para a função em MB.
- `AWS_LAMBDA_FUNCTION_VERSION`: a versão da função que está sendo executada.

`AWS_LAMBDA_INITIALIZATION_TYPE`: o tipo de inicialização da função, que é `on-demand` ou `provisioned-concurrency`. Para obter informações, consulte [Configurar a simultaneidade provisionada \(p. 116\)](#).

- `AWS_LAMBDA_LOG_GROUP_NAME`, `AWS_LAMBDA_LOG_STREAM_NAME`— o nome do grupo Amazon CloudWatch Logs e do fluxo do para a função.
- `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN`: as chaves de acesso obtidas da [função de execução \(p. 57\)](#) da função.
- `AWS_LAMBDA_RUNTIME_API`: ([Tempo de execução personalizado \(p. 255\)](#)) o host e a porta da [API de tempo de execução \(p. 224\)](#).
- `LAMBDA_TASK_ROOT`: o caminho para o código da função do Lambda.
- `LAMBDA_RUNTIME_DIR`: o caminho para bibliotecas de tempo de execução.
- `TZ` – o fuso horário do ambiente (UTC). O ambiente de execução usa o NTP para sincronizar o relógio do sistema.

As variáveis de ambiente adicionais a seguir não são reservadas e podem ser estendidas na configuração de sua função.

Variáveis de ambiente não reservadas

- `LANG` – esta é a localidade do tempo de execução (`en_US.UTF-8`).
- `PATH` – O caminho de execução (`/usr/local/bin:/usr/bin:/bin:/opt/bin`).

- **LD_LIBRARY_PATH** – o caminho da biblioteca do sistema (`/lib64:/usr/lib64:$LAMBDA_RUNTIME_DIR:$LAMBDA_RUNTIME_DIR/lib:$LAMBDA_TASK_ROOT:$LAMBDA_TASK_ROOT/lib:/opt/lib`).
- **NODE_PATH**: ([Node.js \(p. 511\)](#)) o caminho da biblioteca Node.js (`/opt/nodejs/node12/node_modules:/opt/nodejs/node_modules:$LAMBDA_RUNTIME_DIR/node_modules`).
- **PYTHONPATH**: ([Python 2.7, 3.6, 3.8 \(p. 538\)](#)) o caminho da biblioteca Python (`$LAMBDA_RUNTIME_DIR`).
- **GEM_PATH**: ([Ruby \(p. 567\)](#)) o caminho da biblioteca Ruby (`$LAMBDA_TASK_ROOT/vendor/bundle/ruby/2.5.0:/opt/ruby/gems/2.5.0`).
- **AWS_XRAY_CONTEXT_MISSING**: para o rastreamento do X-Ray, o Lambda define isso como `LOG_ERROR` para evitar a execução de erros de tempo de execução no X-Ray SDK.
- **AWS_XRAY_DAEMON_ADDRESS**: para o rastreamento do X-Ray, o endereço IP e a porta do daemon do X-Ray.
- **AWS_LAMBDA_DOTNET_PREJIT**: para o tempo de execução do .NET 3.1, defina essa variável para ativar ou desativar otimizações específicas de tempo de execução do .NET 3.1. Os valores incluem `always`, `never` e `provisioned-concurrency`. Para obter informações, consulte [Configurar a simultaneidade provisionada \(p. 116\)](#).

Os valores de exemplo mostrados refletem os tempos de execução mais recentes. A presença de variáveis específicas ou seus valores pode variar nos tempos de execução anteriores.

Proteger variáveis de ambiente

O Lambda criptografa variáveis de ambiente com uma chave que ele cria na sua conta (uma chave mestra do cliente (CMK) gerenciada pela AWS). O uso dessa chave é gratuito. Você também pode optar por fornecer sua própria chave para ser usada pelo Lambda em vez da chave padrão.

Quando você fornece a chave, somente os usuários em sua conta com acesso à chave podem visualizar ou gerenciar variáveis de ambiente na função. Sua organização também pode ter requisitos internos ou externos para gerenciar as chaves usadas para criptografia e para controlar quando elas são rotacionadas.

Para usar uma CMK gerenciada pelo cliente

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e, em seguida, Environment variables (Variáveis de ambiente).
4. Em Environment variables (Variáveis de ambiente), selecione Edit (Editar).
5. Expanda Encryption configuration (Configuração de criptografia).
6. Escolha Uso de chave mestra do cliente.
7. Escolha a CMK gerenciada pelo cliente.
8. Escolha Save (Salvar).

As CMKs gerenciadas pelo cliente incorrem em [cobranças do AWS KMS](#) padrão.

Para usar a chave de criptografia padrão, não é necessária nenhuma permissão do AWS KMS para o usuário ou para a função de execução da função. Para usar uma CMK gerenciada pelo cliente, é necessário permissão para usar a chave. O Lambda usa essas permissões para criar uma concessão na chave. Isso permite que o Lambda use-a para criptografia.

- `kms>ListAliases`: para visualizar as teclas no console do Lambda.
- `kms>CreateGrant`, `kms:Encrypt`: para configurar uma CMK gerenciada pelo cliente em uma função.

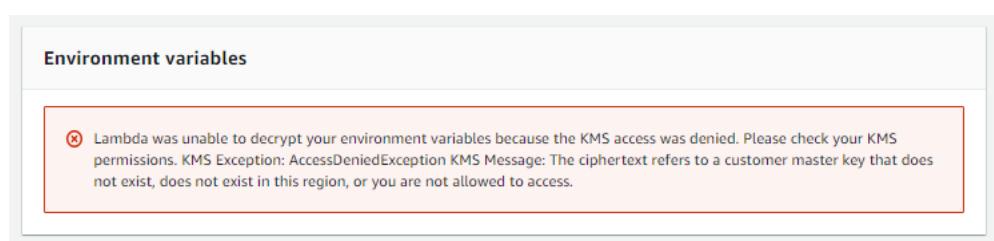
- `kms:Decrypt`: para visualizar e gerenciar variáveis de ambiente criptografadas com uma CMK gerenciada pelo cliente.

É possível obter essas permissões da sua conta de usuário ou da política de permissão baseada em recursos de uma chave. `ListAliases` é fornecido pelas [políticas gerenciadas para o Lambda \(p. 67\)](#). As políticas de chave concedem as permissões restantes aos usuários no grupo Usuários de chaves.

Os usuários sem permissões `Decrypt` ainda poderão gerenciar funções, mas não poderão visualizar variáveis de ambiente nem gerenciá-las no console do Lambda. Para impedir que um usuário visualize variáveis de ambiente, adicione uma declaração às permissões do usuário que negue o acesso à chave padrão, a uma chave gerenciada pelo cliente ou a todas as chaves.

Example Política do IAM: negar acesso pelo ARN da chave

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "VisualEditor0",  
            "Effect": "Deny",  
            "Action": [  
                "kms:Decrypt"  
            ],  
            "Resource": "arn:aws:kms:us-east-2:123456789012:key/3be10e2d-xmpl-4be4-  
bc9d-0405a71945cc"  
        }  
    ]  
}
```



Para obter detalhes sobre o gerenciamento de permissões de chaves, consulte [Usar políticas de chaves no AWS KMS](#).

Também é possível criptografar valores de variáveis de ambiente no lado do cliente antes de enviá-los para o Lambda, e descriptografá-los em seu código de função. Isso obscurece valores secretos na saída do console e da API do Lambda, mesmo para usuários que têm permissão para usar a chave. Em seu código, você recupera o valor criptografado do ambiente e o descriptografa a API do AWS KMS.

Como criptografar variáveis de ambiente no lado do cliente

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e, em seguida, Environment variables (Variáveis de ambiente).
4. Em Environment variables (Variáveis de ambiente), selecione Edit (Editar).
5. Expanda Encryption configuration (Configuração de criptografia).
6. Escolha Enable helpers for encryption in transit (Ativar auxiliares para criptografia em trânsito).
7. Escolha Encrypt (Criptografar) ao lado de uma variável, para criptografar seu valor.
8. Escolha Save (Salvar).

Note

Quando você usa os auxiliares de criptografia do console, sua função precisa de permissão para chamar a operação de API `kms:Decrypt` na [função de execução \(p. 57\)](#).

Para exibir o código de exemplo para o idioma da sua função, escolha **Code (Código)** ao lado de uma variável de ambiente. O código de exemplo mostra como recuperar uma variável de ambiente em uma função e descriptografar seu valor.

Outra opção é armazenar senhas nos segredos do AWS Secrets Manager. É possível fazer referência ao segredo em seus modelos do AWS CloudFormation para definir senhas em bancos de dados. Também é possível definir o valor de uma variável de ambiente na função do Lambda. Consulte a próxima seção para ver um exemplo.

Modelos e código de exemplo

Os aplicativos de exemplo no repositório do GitHub deste guia demonstram a utilização de variáveis de ambiente em modelos de código de função e AWS CloudFormation.

Aplicativos de exemplo

- [Função em branco \(p. 493\)](#): crie uma função e um tópico do Amazon SNS no mesmo modelo. Passe o nome do tópico para a função em uma variável de ambiente. Leia variáveis de ambiente no código (vários idiomas).
- [RDS MySQL](#): crie uma VPC e uma instância de banco de dados do Amazon RDS em um modelo, com uma senha armazenada no Secrets Manager. No modelo de aplicação, importe detalhes do banco de dados da pilha da VPC, leia a senha do Secrets Manager e passe toda a configuração da conexão para a função em variáveis de ambiente.

Versões da função do Lambda

É possível usar versões para gerenciar a implantação das suas funções. Por exemplo, você pode publicar uma nova versão de uma função para testes beta sem afetar os usuários da versão de produção estável. O Lambda cria uma nova versão da sua função sempre que você publicá-la. A nova versão é uma cópia da versão não publicada da função.

A versão da função inclui as seguintes informações:

- O código da função e todas as dependências associadas.
- O tempo de execução do Lambda que invoca a função.
- Todas as configurações das funções, incluindo as variáveis de ambiente.
- Um nome de recurso da Amazon (ARN) exclusivo para identificar essa versão específica da função.

Seções

- [Como criar versões de função \(p. 106\)](#)
- [Gerenciar versões com a API do Lambda \(p. 106\)](#)
- [Usar versões \(p. 107\)](#)
- [Conceder permissões \(p. 107\)](#)

Como criar versões de função

O código e as configurações das funções somente podem ser alterados na versão não publicada de uma função. Ao publicar uma versão, o código e a maioria das configurações são bloqueados, para manter uma experiência consistente para os usuários dessa versão. Para obter mais informações sobre a configuração de funções, consulte [Configurar as opções da função do Lambda \(p. 95\)](#).

Você pode criar uma versão de função usando o console do Lambda.

Para criar uma nova versão de função

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função e, em seguida, Versions (Versões).
3. Na página de configuração de versões, escolha Publish new version (Publicar nova versão).
4. (Opcional) Insira uma descrição de versão.
5. Escolha Publish.

Gerenciar versões com a API do Lambda

Para publicar uma versão de uma função, use a operação de API [PublishVersion \(p. 919\)](#).

O exemplo a seguir publica uma nova versão de uma função. A resposta retorna as informações de configuração sobre a versão da função, incluindo o número da versão e o ARN da função com o sufixo da versão.

```
aws lambda publish-version --function-name my-function
```

Você deve ver a saída a seguir:

```
{
```

```
"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": "nodejs12.x",
...
}
```

Usar versões

É possível fazer referência à sua função do Lambda usando um ARN qualificado ou um ARN não qualificado.

- ARN qualificado: o ARN da função com um sufixo da versão. O exemplo a seguir faz referência à versão 42 da função `helloworld`.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld:42
```

- ARN não qualificado: o ARN da função sem um sufixo da versão.

```
arn:aws:lambda:aws-region:acct-id:function:helloworld
```

É possível usar um ARN qualificado ou não qualificado em todas as operações de API relevantes. No entanto, não é possível usar um ARN não qualificado para criar um alias.

Se você decidir não publicar versões de funções, será possível invocar a função usando o ARN qualificado ou não qualificado no [mapeamento da origem do evento \(p. 170\)](#). Quando você invoca uma função usando um ARN não qualificado, o Lambda invoca implicitamente `$LATEST`.

O Lambda publica uma nova versão de função somente se o código nunca foi publicado ou se o código foi alterado a partir da última versão publicada. Se não houver nenhuma alteração, a versão da função permanecerá na última versão publicada.

O ARN qualificado para cada versão de função do Lambda é exclusivo. Depois de publicar uma versão, não é possível alterar o ARN ou o código da função.

Conceder permissões

É possível usar uma [política baseada em recurso \(p. 62\)](#) ou uma [política baseada em identidade \(p. 67\)](#) para conceder acesso à sua função. O escopo da permissão depende se você aplicar a política a uma função ou a uma versão de uma função. Para obter mais informações sobre nomes de recursos de função em políticas, consulte [Recursos e condições para ações do Lambda \(p. 72\)](#).

É possível simplificar o gerenciamento de fontes de eventos e políticas do AWS Identity and Access Management (IAM) utilizando aliases de funções. Para obter mais informações, consulte [Aliases de função do Lambda \(p. 108\)](#).

Aliases de função do Lambda

Você pode criar um ou mais aliases para sua função do Lambda. Um alias do Lambda é como um indicador para uma versão específica da função. Os usuários podem acessar a versão da função usando o nome de recurso da Amazon (ARN) do alias.

Seções

- [Como criar um alias da função \(console\) \(p. 108\)](#)
- [Gerenciar aliases com a API do Lambda \(p. 108\)](#)
- [Usar aliases \(p. 109\)](#)
- [Políticas de recursos \(p. 109\)](#)
- [Configuração de roteamento de alias \(p. 109\)](#)

Como criar um alias da função (console)

Você pode criar um alias da função usando o console do Lambda.

Para criar um alias

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Aliases e, em seguida, escolha Create alias (Criar alias).
4. Na página Create alias (Criar alias), faça o seguinte:
 - a. Insira um Name (Nome) para o alias.
 - b. (Opcional) Insira uma Description (Descrição) do alias.
 - c. Em Version (Versão), escolha uma versão da função para a qual você deseja que o alias indique.
 - d. (Opcional) Para configurar o roteamento no alias, expanda Weighted alias (Alias ponderado). Para obter mais informações, consulte [Configuração de roteamento de alias \(p. 109\)](#).
 - e. Escolha Save (Salvar).

Gerenciar aliases com a API do Lambda

Para criar um alias usando a AWS Command Line Interface (AWS CLI), use o comando `create-alias`.

```
aws lambda create-alias --function-name my-function --name alias-name --function-version version-number --description " "
```

Para alterar um alias para indicar uma nova versão da função, use o comando `update-alias`.

```
aws lambda update-alias --function-name my-function --name alias-name --function-version version-number
```

Para excluir um alias, use o comando `delete-alias`.

```
aws lambda delete-alias --function-name my-function --name alias-name
```

Os comandos da AWS CLI nas etapas anteriores correspondem às seguintes operações de API do Lambda:

- [CreateAlias \(p. 779\)](#)
- [UpdateAlias \(p. 949\)](#)
- [DeleteAlias \(p. 808\)](#)

Usar aliases

Cada alias tem um ARN exclusivo. Um alias pode apontar somente para uma versão de função, e não para outro alias. O alias pode ser atualizado para apontar para uma nova versão da função.

Fontes de eventos como Amazon Simple Storage Service (Amazon S3) invocam sua função do Lambda. Essas origens de evento mantêm um mapeamento que identifica a função a ser invocada quando ocorrem eventos. Se você especificar um alias da função do Lambda na configuração de mapeamento, não precisará atualizar o mapeamento quando a versão da função for alterada. Para obter mais informações, consulte [AWS Lambda Mapeamentos da fonte de eventos \(p. 170\)](#).

Em uma política de recursos, é possível conceder permissões para fontes de eventos para usar sua função do Lambda. Se um ARN de alias for especificado na política, não será necessário atualizar a política quando a versão da função for alterada.

Políticas de recursos

É possível usar uma [política baseada em recurso \(p. 62\)](#) para conceder acesso a um serviço, recurso ou conta à sua função. O escopo dessa permissão depende se você a aplica a um alias, uma versão ou a toda a função. Por exemplo, se você usar um nome de alias (como `helloworld:PROD`), a permissão permite invocar a função `helloworld` usando o ARN do alias (`helloworld:PROD`).

Se você tentar invocar a função sem um alias ou uma versão específica, receberá um erro de permissão. Esse erro de permissão ainda ocorrerá mesmo se você tentar invocar diretamente a versão da função associada ao alias.

Por exemplo, o comando da AWS CLI a seguir concede permissões do Amazon S3 para invocar o alias PROD da função `helloworld` quando o Amazon S3 está agindo em nome de `examplebucket`.

```
aws lambda add-permission --function-name helloworld \
--qualifier PROD --statement-id 1 --principal s3.amazonaws.com --action
lambda:InvokeFunction \
--source-arn arn:aws:s3:::examplebucket --source-account 123456789012
```

Para obter mais informações sobre como usar nomes de recursos em políticas, consulte [Recursos e condições para ações do Lambda \(p. 72\)](#).

Configuração de roteamento de alias

Use a configuração de roteamento em um alias para enviar uma parte do tráfego para uma segunda versão de função. Por exemplo, você pode reduzir o risco de implantar uma nova versão configurando o alias para enviar a maior parte do tráfego para a versão existente e apenas uma pequena porcentagem de tráfego para a nova versão.

Observe que o Lambda usa um modelo probabilístico simples para distribuir o tráfego entre as duas versões de função. Em níveis de tráfego baixos, você pode ver uma alta variação entre a porcentagem configurada e real de tráfego em cada versão. Se sua função usa simultaneidade provisionada, você pode evitar [Invocações de transbordamento \(p. 717\)](#) configurando um número maior de instâncias de simultaneidade provisionadas durante o tempo em que o roteamento de alias está ativo.

Você pode apontar um alias para um máximo de duas versões de função do Lambda. As versões devem atender aos seguintes critérios:

- As duas versões devem ter a mesma [função de execução \(p. 57\)](#).
- Ambas as versões devem ter a mesma configuração de [fila de mensagens mortas \(p. 167\)](#) ou configuração de nenhuma fila de mensagens mortas.
- Ambas as versões devem ser publicadas. O alias não pode apontar para `$LATEST`.

Para configurar o roteamento em um alias

Note

Verifique se a função tem pelo menos duas versões publicadas. Para criar versões adicionais, siga as instruções em [Versões da função do Lambda \(p. 106\)](#).

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Aliases e, em seguida, escolha Create alias (Criar alias).
4. Na página Create alias (Criar alias), faça o seguinte:
 - a. Insira um Name (Nome) para o alias.
 - b. (Opcional) Insira uma Description (Descrição) do alias.
 - c. Em Version (Versão), escolha a primeira versão da função para a qual você deseja que o alias aponte.
 - d. Expanda Weighted alias (Alias ponderado).
 - e. Em Additional version (Versão adicional), escolha a segunda versão da função para a qual você deseja que o alias aponte.
 - f. Em Weight (%) (Peso (%)), insira um valor de peso para a função. Peso é a porcentagem do tráfego atribuído a essa versão quando o alias é invocado. A primeira versão recebe o peso residual. Por exemplo, se você especificar 10% para Additional version (Versão adicional), a primeira versão receberá automaticamente a atribuição de 90 por cento.
 - g. Escolha Save (Salvar).

Configurar o roteamento de alias usando a CLI

Use os comandos `create-alias` e `update-alias` da AWS CLI para configurar os pesos de tráfego entre duas versões de uma função. Ao criar ou atualizar o alias, o peso do tráfego é especificado no parâmetro `routing-config`.

O exemplo a seguir cria um alias de função do Lambda chamado `routing-alias` que aponta para a versão 1 da função. A versão 2 da função recebe 3 por cento do tráfego. Os 97 por cento do tráfego restantes são roteados para a versão 1.

```
aws lambda create-alias --name routing-alias --function-name my-function --function-version 1 \
--routing-config AdditionalVersionWeights={"2":0.03}
```

Use o comando `update-alias` para aumentar a porcentagem de tráfego de entrada para a versão 2. No exemplo a seguir, o tráfego é aumentado para 5%.

```
aws lambda update-alias --name routing-alias --function-name my-function \
--routing-config AdditionalVersionWeights={"2":0.05}
```

Para rotear todo o tráfego para a versão 2, use o comando `update-alias` para alterar a propriedade `function-version` para apontar o alias para a versão 2. O comando também redefine a configuração de roteamento.

```
aws lambda update-alias --name routing-alias --function-name my-function \
--function-version 2 --routing-config AdditionalVersionWeights={}
```

Os comandos da AWS CLI nas etapas anteriores correspondem às seguintes operações de API do Lambda:

- [CreateAlias \(p. 779\)](#)
- [UpdateAlias \(p. 949\)](#)

Determinar qual versão foi invocada

Ao configurar pesos de tráfego entre duas versões da função, há duas maneiras de determinar a versão da função do Lambda que foi chamada:

- CloudWatch Logs: o Lambda emite automaticamente uma entrada de log START que contém o ID da versão invocada para Amazon CloudWatch Logs em cada invocação da função. Veja um exemplo a seguir:

```
19:44:37 START RequestId: request id Version: $version
```

Para invocações de alias, o Lambda usa a dimensão Executed Version para filtrar os dados de métrica pela versão invocada. Para obter mais informações, consulte [Trabalhar com métricas de função do AWS Lambda \(p. 717\)](#).

- Carga de resposta (invocações síncronas) – As respostas às invocações de função síncrona incluem um cabeçalho x-amz-executed-version para indicar qual versão de função foi invocada.

Gerenciar funções do AWS Lambda

[Configurar funções do](#) (p. 81) descreve como configurar as principais capacidades e opções para uma função. O Lambda também fornece recursos avançados, como controle de simultaneidade, acesso à rede, entrelaçamento de banco de dados, sistemas de arquivos e assinatura de código.

A [simultaneidade](#) (p. 113) é o número de instâncias de sua função que estão ativas. O Lambda oferece dois tipos de controle de simultaneidade: simultaneidade reservada e simultaneidade provisionada.

Para usar a função do Lambda com recursos da AWS em uma Amazon VPC, configure-a com grupos de segurança e sub-redes para [criar uma conexão da VPC](#) (p. 124). Conectar a função a uma VPC permite acessar recursos em uma sub-rede privada, como bancos de dados relacionais e caches. Também é possível [criar um proxy de banco de dados](#) (p. 134) para instâncias de banco de dados MySQL e Aurora. Um proxy de banco de dados permite que uma função atinja altos níveis de simultaneidade sem esgotar as conexões de banco de dados.

Para usar a [assinatura de código](#) (p. 144) com sua função do Lambda, configure-a com uma configuração de assinatura de código. Quando um usuário tenta implantar um pacote de código, o Lambda verifica se ele tem uma assinatura válida de um editor confiável. A configuração de assinatura de código inclui um conjunto de perfis de assinatura que define os editores confiáveis para essa função.

Gerenciar simultaneidade para uma função do Lambda

Simultaneidade é o número de solicitações que a função atende a cada momento. Quando a função é invocada, o Lambda aloca uma instância dela para processar o evento. Quando a execução do código da função terminar, ela poderá processar outra solicitação. Se a função for invocada novamente enquanto uma solicitação ainda estiver sendo processada, outra instância será alocada, o que aumenta a simultaneidade da função. A simultaneidade está sujeita a uma [cota \(p. 53\)](#) regional que é compartilhada por todas as funções em uma região.

Há dois tipos de controle de simultaneidade disponíveis:

- Simultaneidade reservada: garante o número máximo de instâncias simultâneas para a função. Quando uma função tem simultaneidade reservada, nenhuma outra função pode usar essa simultaneidade. Não há cobrança para configurar a simultaneidade reservada para uma função.
- Simultaneidade provisionada: inicializa um número solicitado de ambientes de execução para que eles estejam preparados para responder imediatamente às invocações da sua função. Observe que a configuração da simultaneidade provisionada incorre em cobranças na sua conta da AWS.

Este tópico detalha como gerenciar e configurar a simultaneidade reservada e provisionada. Para saber como a simultaneidade interage com a escalabilidade, consulte [Escalabilidade de função do Lambda](#).

Para garantir que uma função possa sempre atingir um determinado nível de simultaneidade, é possível configurar a função com [simultaneidade reservada \(p. 114\)](#). Quando uma função tem simultaneidade reservada, nenhuma outra função pode usar essa simultaneidade. A simultaneidade reservada também limita a simultaneidade máxima para a função e aplica-se à função como um todo, incluindo versões e aliases.

Quando o Lambda aloca uma instância da sua função, o [tempo de execução \(p. 214\)](#) carrega o código da sua função e executa o código de inicialização definido fora do manipulador. Se seu código e dependências forem grandes, ou você criar clientes de SDK durante a inicialização, esse processo poderá levar algum tempo. À medida que a função é [expandida \(p. 32\)](#), a parte das solicitações atendidas por novas instâncias passa a ter latência maior do que o restante.

Para permitir que sua função seja dimensionada sem flutuações na latência, use [simultaneidade provisionada \(p. 116\)](#). Ao alocar simultaneidade provisionada antes de um aumento nas invocações, é possível garantir que todas as solicitações sejam atendidas por instâncias inicializadas com latência muito baixa. Você pode configurar a simultaneidade provisionada em uma versão de uma função ou em um alias.

O Lambda também se integra ao [Application Auto Scaling](#). É possível configurar o Application Auto Scaling para gerenciar a simultaneidade provisionada em uma programação ou com base na utilização. Use o dimensionamento agendado para aumentar a simultaneidade provisionada em antecipação ao tráfego máximo. Para aumentar a simultaneidade provisionada automaticamente conforme necessário, use a [API do Application Auto Scaling \(p. 119\)](#) para registrar um destino e criar uma política de escalabilidade.

A simultaneidade provisionada é contabilizada para a simultaneidade reservada e as cotas regionais de uma função. Se a quantidade de simultaneidade provisionada nas versões e aliases de uma função se somar à simultaneidade reservada da função, todas as invocações serão executadas na simultaneidade provisionada. Essa configuração também tem o efeito de controlar a utilização da versão não publicada da função (`$LATEST`), o que impede que ela seja executada.

Note

Não é possível alocar mais simultaneidade provisionada do que a simultaneidade reservada para uma função.

Seções

- [Configurar a simultaneidade reservada \(p. 114\)](#)
- [Configurar a simultaneidade provisionada \(p. 116\)](#)
- [Configurar a simultaneidade com a API do Lambda \(p. 119\)](#)

Configurar a simultaneidade reservada

Para gerenciar configurações de simultaneidade reservada para uma função, use o console do Lambda.

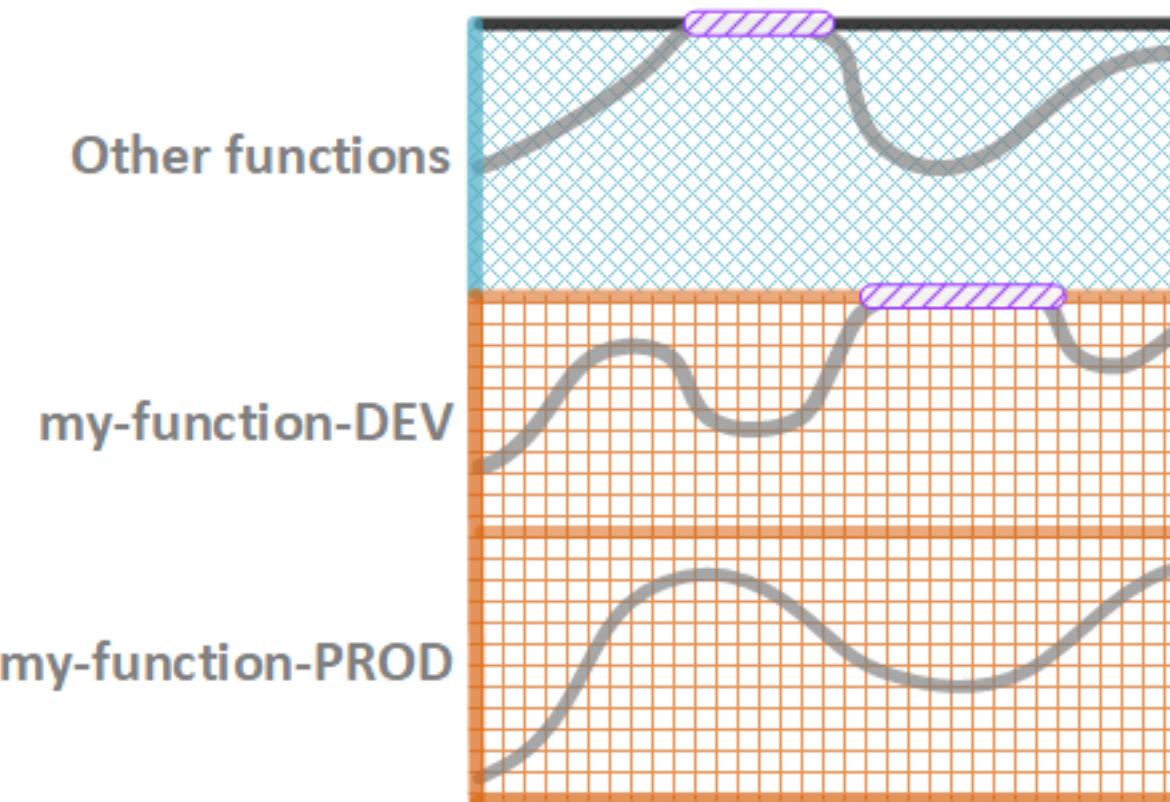
Para reservar simultaneidade para uma função

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e escolha (Concurrency (Simultaneidade)).
4. Em Concurrency (Simultaneidade), selecione Edit (Editar).
5. Selecione Reserve concurrency (Reservar simultaneidade). Insira a quantidade de simultaneidade para reservar para a função.
6. Escolha Save (Salvar).

É possível reservar até o valor mostrado de Unreserved account concurrency (Simultaneidade da conta não reservada), menos 100 para funções que não têm simultaneidade reservada. Para limitar uma função, defina a simultaneidade reservada como zero. Isso impede que todos os eventos sejam processados até que você remova o limite.

O exemplo a seguir mostra duas funções com grupos de simultaneidade reservada e o grupo de simultaneidade não reservada usado por outras funções. Erros de limitação ocorrem quando toda a simultaneidade em um grupo está em uso.

Reserved Concurrency



Legend

- Simultaneidade de funções
- Simultaneidade reservada
- Simultaneidade não reservada
- Limitação

Reservar simultaneidade tem os seguintes efeitos.

- Outras funções não podem impedir que sua função seja dimensionada: todas as funções da sua conta na mesma região sem simultaneidade reservada compartilham o grupo de simultaneidade não reservada. Sem simultaneidade reservada, outras funções podem usar toda a simultaneidade disponível. Isso evita que sua função seja ampliada quando necessário.
- Sua função não pode ter aumento de escala na horizontal a ponto de perder o controle: a simultaneidade reservada também impede que sua função use simultaneidade do grupo não reservado,

o que limita sua simultaneidade máxima. É possível reservar a simultaneidade para impedir que sua função use toda a simultaneidade disponível na região ou sobrecarregue os recursos downstream.

Configurar a simultaneidade por função pode afetar o grupo de simultaneidade disponível para outras funções. Para evitar problemas, limite o número de usuários que podem usar as operações de API `PutFunctionConcurrency` e `DeleteFunctionConcurrency`.

Configurar a simultaneidade provisionada

Para gerenciar configurações de simultaneidade provisionada para uma versão ou alias, use o console do Lambda.

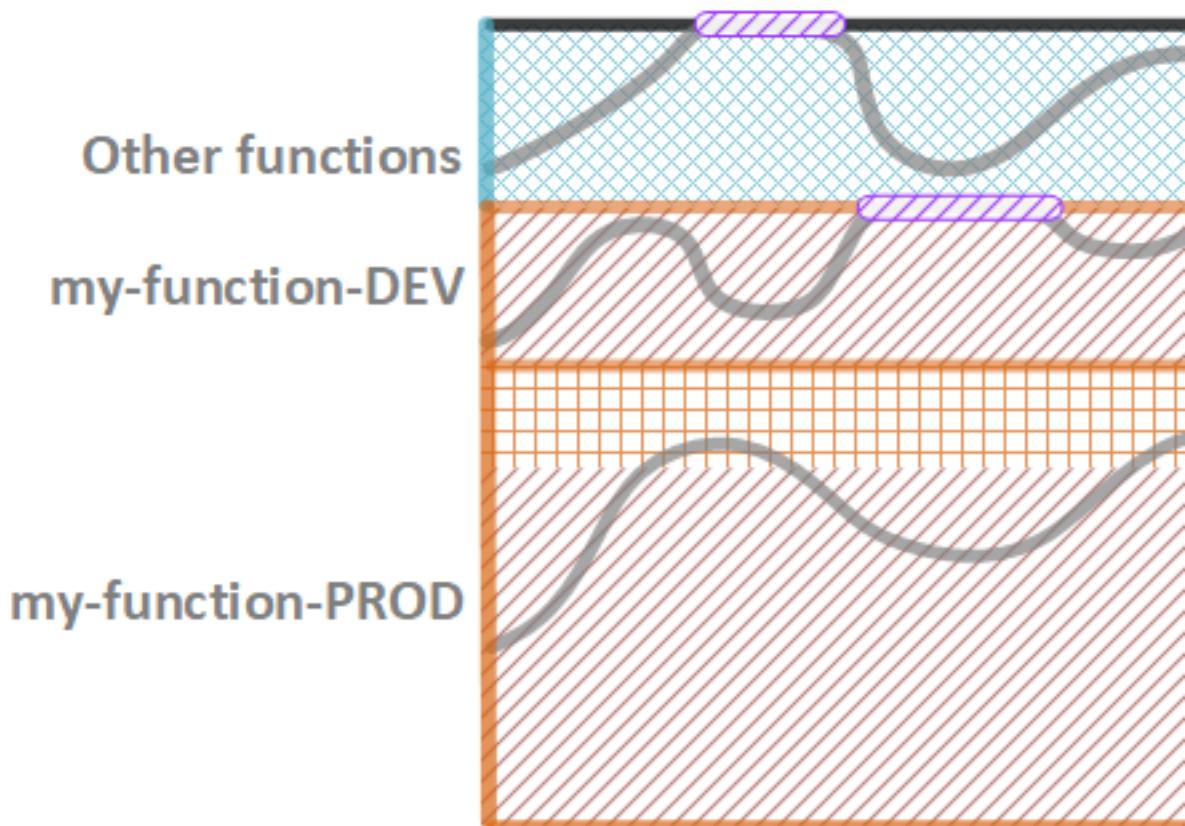
Para provisionar simultaneidade para um alias

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e escolha Concurrency (Simultaneidade).
4. Em Provisioned concurrency configurations (Configurações de simultaneidade provisionada), escolha Add configuration (Adicionar configuração).
5. Escolha um alias ou versão.
6. Informe a quantia de simultaneidade provisionada a ser alocada.
7. Escolha Save (Salvar).

É possível gerenciar a simultaneidade provisionada para todos os aliases e versões na página de configuração da função. A lista de configurações de simultaneidade provisionada mostra o andamento da alocação de cada configuração. As configurações de simultaneidade provisionada também estão disponíveis na página de configuração de cada versão e alias.

No exemplo a seguir, as funções `my-function-DEV` e `my-function-PROD` são configuradas com simultaneidade reservada e provisionada. Em `my-function-DEV`, o grupo completo de simultaneidade reservada também é a simultaneidade provisionada. Nesse caso, todas as invocações são executadas em simultaneidade provisionada ou são limitadas. Em `my-function-PROD`, uma parte do grupo de simultaneidade reservada é a simultaneidade padrão. Quando toda a simultaneidade provisionada está em uso, a função escalada para a simultaneidade padrão para atender a quaisquer solicitações adicionais.

Provisioned Concurrency with Reserved Concurrency

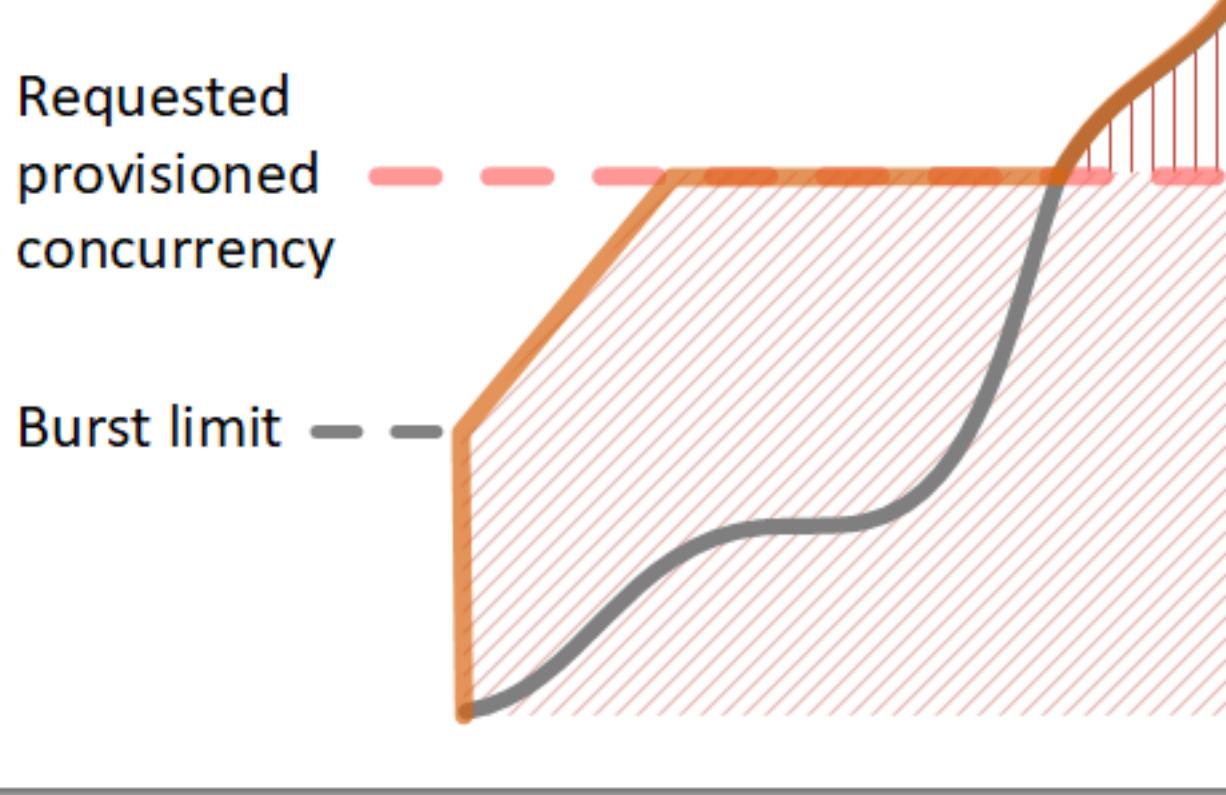


Legend

- Simultaneidade de funções
- Simultaneidade reservada
- Simultaneidade provisionada
- Simultaneidade não reservada
- Limitação

A simultaneidade provisionada não fica online imediatamente após a configuração. O Lambda começa a alocar a simultaneidade provisionada após um ou dois minutos de preparação. Semelhante à forma como as funções [escalam sob carga \(p. 32\)](#), é possível inicializar até 3000 instâncias da função de uma só vez, dependendo da região. Após a intermitência inicial, as instâncias são alocadas a uma taxa constante de 500 por minuto até que a solicitação seja atendida. Quando você solicita simultaneidade provisionada para várias funções ou versões de uma função na mesma região, as cotas de escalabilidade são aplicáveis a todas as solicitações.

Function Scaling with Provisioned Concurrency



Legend

- Instâncias de função
- Solicitações abertas
- Simultaneidade provisionada
- Simultaneidade padrão

Para otimizar a latência, você pode personalizar o comportamento de inicialização para funções que usam simultaneidade provisionada. Você pode executar o código de inicialização para instâncias de simultaneidade provisionadas sem afetar a latência, porque o código de inicialização é executado no momento da alocação. No entanto, o código de inicialização de uma instância sob demanda afeta diretamente a latência da primeira chamada. Para uma instância sob demanda, você pode optar por adiar a inicialização de um recurso específico até que a função precise desse recurso.

Para determinar o tipo de inicialização, verifique o valor de AWS_LAMBDA_INITIALIZATION_TYPE. O Lambda define essa variável de ambiente como provisioned-concurrency-on-demand. O valor de AWS_LAMBDA_INITIALIZATION_TYPE é imutável e não muda durante a vida útil do ambiente de execução.

Se você usar o tempo de execução do .NET 3.1, poderá configurar a variável de ambiente AWS_LAMBDA_DOTNET_PREJIT para melhorar a latência para funções que usam simultaneidade provisionada. O tempo de execução .NET compila e inicializa, sem otimização, cada biblioteca que seu código chama pela primeira vez. Como resultado, a primeira invocação de uma função do Lambda pode levar mais tempo do que as invocações subsequentes. Quando você define AWS_LAMBDA_DOTNET_PREJIT como ProvisionedConcurrencyO Lambda executa a compilação JIT antecipada para dependências comuns do sistema. O Lambda executa essa otimização de inicialização apenas para instâncias de simultaneidade provisionadas, o que resulta em performance mais rápida para a primeira invocação. Se você definir a variável de ambiente como Always, o Lambda executa a compilação JIT antecipada para cada inicialização. Se você definir a variável de ambiente como Never, a compilação JIT antecipada será desativada. O valor padrão para AWS_LAMBDA_DOTNET_PREJIT é ProvisionedConcurrency.

Para instâncias de simultaneidade provisionada, o [código de inicialização \(p. 30\)](#) da função é executado durante a alocação e de poucas em poucas horas, porque as instâncias em execução da sua função são recicladas. O tempo de inicialização pode ser visualizado em logs e [rastreamentos \(p. 477\)](#) depois que uma instância processar uma solicitação. No entanto, a inicialização é cobrada mesmo que a instância nunca processe uma solicitação. A simultaneidade provisionada é executada continuamente e faturada separadamente dos custos de inicialização e de chamada. Para obter detalhes, consulte [Definição de preço do AWS Lambda](#).

Cada versão de uma função só pode ter uma configuração de simultaneidade provisionada. Ela pode estar diretamente na própria versão ou em um alias que aponte para a versão. Dois aliases não podem alocar simultaneidade provisionada para a mesma versão. Além disso, não é possível alocar simultaneidade provisionada em um alias que aponte para a versão não publicada (\$LATEST).

Quando você altera a versão para a qual um alias aponta, a simultaneidade provisionada é desalocada da versão antiga e, depois, alocada para a nova versão. É possível adicionar uma configuração de roteamento a um alias que tenha simultaneidade provisionada. No entanto, você não poderá gerenciar configurações de simultaneidade provisionada no alias enquanto a configuração de roteamento estiver em vigor.

O Lambda emite as seguintes métricas para simultaneidade provisionada:

Métricas de simultaneidade provisionada

- ProvisionedConcurrentExecutions
- ProvisionedConcurrencyInvocations
- ProvisionedConcurrencySpilloverInvocations
- ProvisionedConcurrencyUtilization

Para obter mais detalhes, consulte [Trabalhar com métricas de função do AWS Lambda \(p. 717\)](#).

Configurar a simultaneidade com a API do Lambda

Para gerenciar as configurações de simultaneidade e escalabilidade automática com a AWS CLI ou com o AWS SDK, use as operações de API a seguir.

- PutFunctionConcurrency (p. 930)
- getFunctionConcurrency

- [DeleteFunctionConcurrency \(p. 822\)](#)
- [PutProvisionedConcurrencyConfig](#)
- [GetProvisionedConcurrencyConfig](#)
- [ListProvisionedConcurrencyConfigs](#)
- [DeleteProvisionedConcurrencyConfig](#)
- [getAccountSettings \(p. 830\)](#)
- [\(Application Auto Scaling\)RegisterScalableTarget](#)
- [\(Application Auto Scaling\)PutScalingPolicy](#)

Para configurar a simultaneidade reservada com a AWS CLI, use o comando `put-function-concurrency`. O comando a seguir reserva uma simultaneidade de 100 para uma função chamada `my-function`:

```
aws lambda put-function-concurrency --function-name my-function --reserved-concurrent-executions 100
```

Você deve ver a saída a seguir:

```
{  
    "ReservedConcurrentExecutions": 100  
}
```

Para alocar simultaneidade provisionada para uma função, use `put-provisioned-concurrency-config`. O comando a seguir aloca uma simultaneidade de 100 para o alias `BLUE` de uma função chamada `my-function`:

```
aws lambda put-provisioned-concurrency-config --function-name my-function \  
--qualifier BLUE --provisioned-concurrent-executions 100
```

Você deve ver a saída a seguir:

```
{  
    "Requested_ProvisionedConcurrentExecutions": 100,  
    "Allocated_ProvisionedConcurrentExecutions": 0,  
    "Status": "IN_PROGRESS",  
    "LastModified": "2019-11-21T19:32:12+0000"  
}
```

Para configurar o Application Auto Scaling para gerenciar a simultaneidade provisionada, use o Application Auto Scaling para configurar o[Escalabilidade de rastreamento de destino](#). Primeiro, registre o alias de uma função como um destino de escalabilidade. O exemplo a seguir registra o alias `BLUE` de uma função chamada `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
```

Depois, aplique uma política de escalabilidade ao destino. O exemplo a seguir configura o Application Auto Scaling para ajustar a configuração de simultaneidade provisionada para um alias manter a utilização próxima de 70%.

```
aws application-autoscaling put-scaling-policy --service-namespace lambda \  
--scalable-dimension lambda:function:ProvisionedConcurrency --target-id ...
```

```
--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" }}'
```

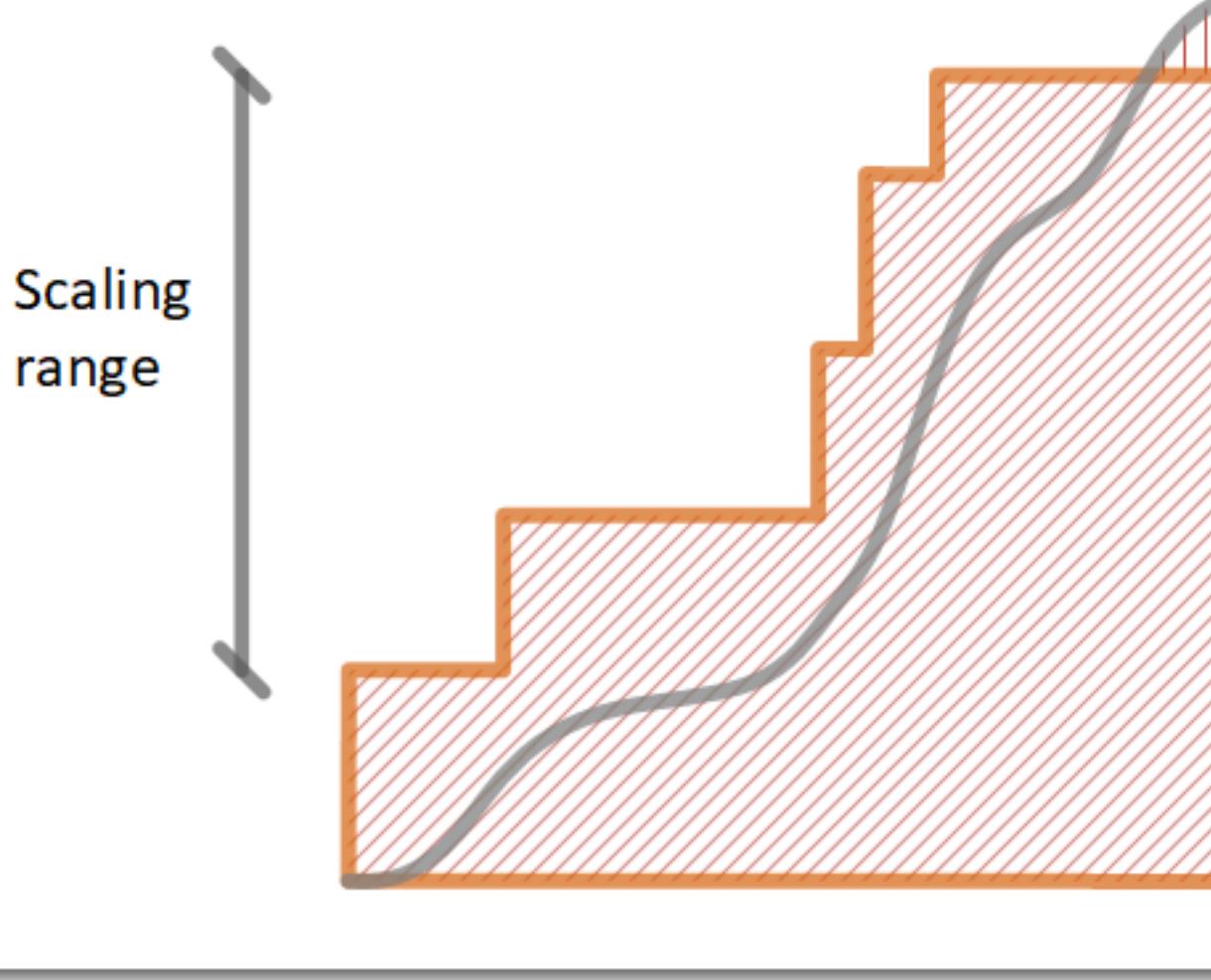
Você deve ver a saída a seguir:

```
{
    "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"
        }
    ]
}
```

O Application Auto Scaling cria dois alarmes no CloudWatch. O primeiro alarme será acionado quando a utilização da simultaneidade provisionada exceder consistentemente 70%. Quando isso acontecer, o Application Auto Scaling alocará mais simultaneidade provisionada para reduzir a utilização. O segundo alarme será acionado quando a utilização for consistentemente inferior a 63% (90% da meta de 70%). Quando isso acontecer, o Application Auto Scaling reduzirá a simultaneidade provisionada do alias.

No exemplo a seguir, uma função é dimensionada entre uma quantidade mínima e máxima de simultaneidade provisionada com base na utilização. Quando o número de solicitações abertas aumenta, o Application Auto Scaling aumenta a simultaneidade provisionada em grandes etapas até atingir o máximo configurado. A função continua a ser dimensionada em simultaneidade padrão até que a utilização comece a cair. Quando a utilização é consistentemente baixa, o Application Auto Scaling diminui a simultaneidade provisionada em etapas periódicas menores.

Autoscaling with Provisioned Concurrency



Legend

- Instâncias de função
- Solicitações abertas
- Simultaneidade provisionada
- Simultaneidade padrão

Para visualizar as cotas de simultaneidade da sua conta em uma região, use `get-account-settings`.

```
aws lambda get-account-settings
```

Você deve ver a saída a seguir:

```
{  
    "AccountLimit": {  
        "TotalCodeSize": 80530636800,  
        "CodeSizeUnzipped": 262144000,  
        "CodeSizeZipped": 52428800,  
        "ConcurrentExecutions": 1000,  
        "UnreservedConcurrentExecutions": 900  
    },  
    "AccountUsage": {  
        "TotalCodeSize": 174913095,  
        "FunctionCount": 52  
    }  
}
```

Configurar uma função do Lambda para acessar recursos em uma VPC

É possível configurar uma função do Lambda para se conectar a sub-redes privadas em uma nuvem privada virtual (VPC) na sua conta da AWS. Use a Amazon Virtual Private Cloud (Amazon VPC) para criar uma rede privada para recursos, como bancos de dados, instâncias de cache ou serviços internos. Conecte a função à VPC para acessar recursos privados enquanto a função está em execução.

Quando você conecta uma função a uma VPC, o Lambda cria uma [interface de rede elástica](#) para cada sub-rede na configuração da VPC da função. Esse processo pode levar alguns minutos.

Enquanto o Lambda cria uma interface de rede, não é possível executar operações adicionais direcionadas à função, como [criar versões \(p. 106\)](#) ou atualizar o código da função. Para novas funções, não é possível invocar a função até que seu estado seja alterado de Pending para Active. Para funções existentes, ainda é possível invocar uma versão anterior enquanto a atualização está em andamento. Para obter mais informações sobre estados de funções, consulte [Monitorar o estado de uma função com a API do Lambda \(p. 174\)](#).

Várias funções podem compartilhar uma interface de rede, se as funções compartilharem a mesma sub-rede e grupo de segurança. Conectar funções adicionais a uma sub-rede que tenha uma interface de rede existente gerenciada pelo Lambda é muito mais rápido do que fazer com que o Lambda crie interfaces de rede adicionais. No entanto, se você tiver muitas funções ou funções com alto uso de rede, o Lambda ainda poderá criar interfaces de rede adicionais.

Se as funções permanecerem inativas por um longo período, o Lambda recuperará suas interfaces de rede e as funções se tornarão `Idle`. Para reativar uma função ociosa, invoque-a. Essa invocação falha e a função entra em um estado `Pending` novamente até que uma interface de rede esteja disponível.

As funções do Lambda não podem se conectar diretamente a uma VPC com a [locação de instâncias dedicadas](#). Para se conectar a recursos em uma VPC dedicada, [emparelhe-a com uma segunda VPC com locação padrão](#).

Seções

- [Função de execução e permissões de usuário \(p. 124\)](#)
- [Como configurar o acesso à VPC \(console\) \(p. 125\)](#)
- [Como configurar o acesso à VPC \(API\) \(p. 126\)](#)
- [Usar chaves de condição do IAM para configurações de VPC \(p. 126\)](#)
- [Acesso aos serviços e à Internet para funções conectadas à VPC \(p. 129\)](#)
- [Tutoriais de VPC \(p. 130\)](#)
- [Configurações de VPC de exemplo \(p. 130\)](#)

Função de execução e permissões de usuário

O Lambda usa as permissões da sua função para criar e gerenciar interfaces de rede. Para se conectar a uma VPC, a [função de execução \(p. 57\)](#) da função precisa ter as seguintes permissões.

Permissões da função de execução

- `ec2:CreateNetworkInterface`
- `ec2:DescribeNetworkInterfaces`
- `ec2:DeleteNetworkInterface`

Essas permissões são incluídas na política gerenciada AWSLambdaVPCAccessExecutionRole da AWS.

Quando você configura a conectividade da VPC, o Lambda usa suas permissões para verificar os recursos de rede. Para configurar uma função para se conectar a uma VPC, o usuário do AWS Identity and Access Management (IAM) precisa das seguintes permissões:

Permissões de usuário

- ec2:DescribeSecurityGroups
- ec2:DescribeSubnets
- ec2:DescribeVpcs

Como configurar o acesso à VPC (console)

Se as [permissões do IAM \(p. 126\)](#) permitirem que você crie apenas funções do Lambda que se conectam à VPC, será necessário configurar a VPC ao criar a função. Se as permissões do IAM permitirem que você crie funções que não estão conectadas à VPC, será possível adicionar a configuração da VPC após criar a função.

Como configurar uma VPC ao criar uma função

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha Create function.
3. Em Basic information (Informações básicas), para Function name (Nome da função), insira um nome para a função.
4. Expanda Advanced settings (Configurações avançadas).
5. Em Network (Rede), escolha uma VPC para a função acessar.
6. Escolha sub-redes e grupos de segurança. Quando você escolhe um grupo de segurança, o console exibe as regras de entrada e saída dele.

Note

Para acessar recursos privados, conecte a função às sub-redes privadas. Se a função precisar de acesso à Internet, use a [conversão de endereços de rede \(NAT\) \(p. 129\)](#). Conectar uma função a uma sub-rede pública não dá a ela acesso à Internet nem a um endereço IP público.

7. Escolha Create function.

Como configurar uma VPC para uma função existente

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e, em seguida, escolha VPC.
4. Em VPC, selecione Edit (Editar).
5. Selecione uma VPC, as sub-redes e os grupos de segurança.

Note

Para acessar recursos privados, conecte a função às sub-redes privadas. Se a função precisar de acesso à Internet, use a [conversão de endereços de rede \(NAT\) \(p. 129\)](#). Conectar uma função a uma sub-rede pública não dá a ela acesso à Internet nem a um endereço IP público.

6. Escolha Save (Salvar).

Como configurar o acesso à VPC (API)

Para conectar uma função do Lambda a uma VPC, é possível usar as seguintes operações de API:

- [CreateFunction \(p. 796\)](#)
- [UpdateFunctionConfiguration \(p. 974\)](#)

Para criar uma função e conectá-la a uma VPC usando a AWS Command Line Interface (AWS CLI), é possível usar o comando `create-function` com a opção `vpc-config`. O exemplo a seguir cria uma função com uma conexão a uma VPC com duas sub-redes e um grupo de segurança.

```
aws lambda create-function --function-name my-function \
--runtime nodejs12.x --handler index.js --zip-file fileb://function.zip \
--role arn:aws:iam::123456789012:role/lambda-role \
--vpc-config
SubnetIds=subnet-071f712345678e7c8,subnet-07fd123456788a036,SecurityGroupIds=sg-085912345678492fb
```

Para conectar uma função existente a uma VPC, use o comando `update-function-configuration` com a opção `vpc-config`.

```
aws lambda update-function-configuration --function-name my-function \
--vpc-config
SubnetIds=subnet-071f712345678e7c8,subnet-07fd123456788a036,SecurityGroupIds=sg-085912345678492fb
```

Para desconectar a função de uma VPC, atualize a configuração da função com uma lista vazia de sub-redes e grupos de segurança.

```
aws lambda update-function-configuration --function-name my-function \
--vpc-config SubnetIds=[],SecurityGroupIds=[]
```

Usar chaves de condição do IAM para configurações de VPC

É possível usar chaves de condição específicas do Lambda para configurações de VPC a fim de fornecer controles de permissão adicionais para as funções do Lambda. Por exemplo, você pode exigir que todas as funções em sua organização estejam conectadas a uma VPC. Você também pode especificar as sub-redes e os grupos de segurança que os usuários da função podem e não podem usar.

O Lambda oferece suporte às seguintes chaves de condição em políticas do IAM:

- `lambda:VpcIds`: permitir ou negar uma ou mais VPCs.
- `lambda:SubnetIds`: permitir ou negar uma ou mais sub-redes.
- `lambda:SecurityGroupIds`: permitir ou negar um ou mais grupos de segurança.

As operações de API [CreateFunction \(p. 796\)](#) e [UpdateFunctionConfiguration \(p. 974\)](#) do Lambda oferecem suporte a essas chaves de condição. Para obter mais informações sobre o uso de chaves de condição em políticas do IAM, consulte [Elementos de política JSON do: condição no IAM User Guide](#).

Tip

Se a função já incluir uma configuração de VPC de uma solicitação de API anterior, você poderá enviar uma solicitação `UpdateFunctionConfiguration` sem a configuração da VPC.

Políticas de exemplo com chaves de condição para configurações de VPC

Os exemplos a seguir demonstram como usar chaves de condição para configurações de VPC. Depois de criar uma instrução de política com as restrições desejadas, acrescente a instrução de política para o usuário ou a função do IAM de destino.

Os usuários devem implantar somente funções conectadas à VPC

Para garantir que todos os usuários implantem somente funções conectadas à VPC, você pode negar operações de criação e de atualização de funções que não incluem um ID de VPC válido.

Observe que o ID da VPC não é um parâmetro de entrada para a solicitação `CreateFunction` ou `UpdateFunctionConfiguration`. O Lambda recupera o valor do ID da VPC com base nos parâmetros da sub-rede e do grupo de segurança.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "EnforceVPCFunction",  
            "Action": [  
                "lambda:CreateFunction",  
                "lambda:UpdateFunctionConfiguration"  
            ],  
            "Effect": "Deny",  
            "Resource": "*",  
            "Condition": {  
                "Null": {  
                    "lambda:VpcIds": "true"  
                }  
            }  
        }  
    ]  
}
```

Negar acesso de usuários a VPCs, sub-redes ou grupos de segurança específicos

Para negar acesso de usuários a VPCs específicas, use `StringEquals` para verificar o valor da condição `lambda:VpcIds`. O exemplo a seguir nega aos usuários acesso à `vpc-1` e à `vpc-2`.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "EnforceOutOfVPC",  
            "Action": [  
                "lambda:CreateFunction",  
                "lambda:UpdateFunctionConfiguration"  
            ],  
            "Effect": "Deny",  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "lambda:VpcIds": ["vpc-1", "vpc-2"]  
                }  
            }  
        }  
    ]  
}
```

```
}
```

Para negar acesso de usuários a sub-redes específicas, use `StringEquals` para verificar o valor da condição `lambda:SubnetIds`. O exemplo a seguir nega aos usuários acesso à `subnet-1` e à `subnet-2`.

```
{
    "Sid": "EnforceOutOfSubnet",
    "Action": [
        "lambda>CreateFunction",
        "lambda:UpdateFunctionConfiguration"
    ],
    "Effect": "Deny",
    "Resource": "*",
    "Condition": {
        "ForAnyValue:StringEquals": {
            "lambda:SubnetIds": ["subnet-1", "subnet-2"]
        }
    }
}
```

Para negar acesso de usuários a grupos de segurança específicos, use `StringEquals` para verificar o valor da condição `lambda:SecurityGroupIds`. O exemplo a seguir nega aos usuários acesso à `sg-1` e à `sg-2`.

```
{
    "Sid": "EnforceOutOfSecurityGroups",
    "Action": [
        "lambda>CreateFunction",
        "lambda:UpdateFunctionConfiguration"
    ],
    "Effect": "Deny",
    "Resource": "*",
    "Condition": {
        "ForAnyValue:StringEquals": {
            "lambda:SecurityGroupIds": ["sg-1", "sg-2"]
        }
    }
}
```

Permitir que os usuários criem e atualizem funções com configurações de VPC específicas

Para permitir que os usuários accessem VPCs específicas, use `StringEquals` para verificar o valor da condição `lambda:VpcIds`. O exemplo a seguir dá aos usuários permissão para acessar `vpc-1` e `vpc-2`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceStayInSpecificVpc",
            "Action": [
                "lambda>CreateFunction",
                "lambda:UpdateFunctionConfiguration"
            ],
            "Resource": "*"
        }
    ]
}
```

```
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "lambda:VpcIds": ["vpc-1", "vpc-2"]
        }
    }
}
```

Para permitir que os usuários accessem sub-redes específicas, use `StringEquals` para verificar o valor da condição `lambda:SubnetIds`. O exemplo a seguir dá aos usuários permissão para acessar `subnet-1` e `subnet-2`.

```
{
    "Sid": "EnforceStayInSpecificSubnets",
    "Action": [
        "lambda>CreateFunction",
        "lambda:UpdateFunctionConfiguration"
    ],
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
        "ForAllValues:StringEquals": {
            "lambda:SubnetIds": ["subnet-1", "subnet-2"]
        }
    }
}
```

Para permitir que os usuários accessem grupos de segurança específicos, use `StringEquals` para verificar o valor da condição `lambda:SecurityGroupIds`. O exemplo a seguir dá aos usuários permissão para acessar `sg-1` e `sg-2`.

```
{
    "Sid": "EnforceStayInSpecificSecurityGroup",
    "Action": [
        "lambda>CreateFunction",
        "lambda:UpdateFunctionConfiguration"
    ],
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
        "ForAllValues:StringEquals": {
            "lambda:SecurityGroupIds": ["sg-1", "sg-2"]
        }
    }
}
```

Acesso aos serviços e à Internet para funções conectadas à VPC

Por padrão, o Lambda executa as funções em uma VPC segura com acesso aos serviços da AWS e à Internet. O Lambda possui esta VPC, que não está conectada à [VPC padrão](#). Quando você conecta uma função a uma VPC em sua conta, a função não consegue acessar a Internet, a menos que a VPC conceda acesso.

Note

Vários serviços da AWS oferecem [VPC endpoints](#). É possível usar VPC endpoints para se conectar aos serviços da AWS de dentro de uma VPC sem acesso à Internet.

O acesso à Internet em uma sub-rede privada requer a conversão de endereços de rede (NAT). Para conceder acesso à Internet para a função, roteie o tráfego de saída para um [gateway NAT](#) em uma sub-rede pública. O gateway NAT tem um endereço IP público e pode se conectar à Internet pelo gateway da Internet da VPC. Para obter mais informações, consulte [Como faço para conceder acesso à Internet a uma função do Lambda em uma VPC?](#)

Tutoriais de VPC

Nos tutoriais a seguir, você conecta uma função do Lambda a recursos na VPC.

- [Tutorial: configurar uma função do Lambda para acessar o Amazon RDS em uma Amazon VPC \(p. 425\)](#)
- [Tutorial: configurar uma função do Lambda para acessar o Amazon ElastiCache em uma Amazon VPC \(p. 366\)](#)

Configurações de VPC de exemplo

É possível usar os modelos do AWS CloudFormation de exemplo a seguir a fim de criar configurações de VPC para usar com funções do Lambda. Há dois modelos disponíveis no repositório GitHub deste guia:

- [`vpc-private.yaml`](#)— uma VPC com duas sub-redes privadas e VPC endpoints para o Amazon Simple Storage Service (Amazon S3) e o Amazon DynamoDB. Use esse modelo para criar uma VPC para funções que não precisam de acesso à Internet. Essa configuração oferece suporte ao uso do Amazon S3 e do DynamoDB com os AWS SDK, e acesso a recursos do banco de dados na mesma VPC por meio de uma conexão de rede local.
- [`vpc-privatepublic.yaml`](#): uma VPC com duas sub-redes privadas, endpoints da VPC, uma sub-rede pública com um gateway NAT e um gateway da Internet. O tráfego direcionado à Internet de funções nas sub-redes privadas é roteado para o gateway NAT usando uma tabela de rotas.

Para criar uma VPC usando um modelo, na [página Pilhas](#) do console do AWS CloudFormation, selecione Create stack (Criar pilha) e siga as instruções no assistente Create stack (Criar pilha).

Configurar VPC endpoints de interface para o Lambda

Se você usar a Amazon Virtual Private Cloud (Amazon VPC) para hospedar os recursos da AWS, poderá estabelecer uma conexão entre a VPC e o Lambda. Você pode usar essa conexão para invocar a função do Lambda sem passar pela Internet pública.

Para estabelecer uma conexão privada entre a VPC e o Lambda, crie um [endpoint da VPC de interface](#). Os endpoints de interface são desenvolvidos pelo [AWS PrivateLink](#), o que permite que você acesse APIs do Lambda de forma privada, sem um gateway da Internet, um dispositivo NAT, uma conexão VPN ou uma conexão do AWS Direct Connect. As instâncias na VPC não precisam de endereços IP públicos para a comunicação com APIs do Lambda. O tráfego de rede entre a VPC e o Lambda não sai da rede da AWS.

Cada endpoint de interface é representado por uma ou mais [interfaces de rede elástica](#) nas sub-redes. Uma interface de rede fornece um endereço IP privado que serve como um ponto de entrada do tráfego para o Lambda.

Seções

- [Considerações para endpoints de interface do Lambda \(p. 131\)](#)
- [Criar um endpoint de interface para o Lambda \(p. 132\)](#)
- [Criar uma política de endpoint de interface para o Lambda \(p. 133\)](#)

Considerações para endpoints de interface do Lambda

Antes de configurar um endpoint de interface para o Lambda, analise o tópico [Interface endpoint properties and limitations](#) no Manual do usuário da Amazon VPC.

É possível chamar qualquer uma das operações de API do Lambda pela VPC. Por exemplo, você pode invocar a função do Lambda chamando a API `Invoke` de dentro da VPC. Para obter a lista completa de APIs do Lambda, consulte [Ações](#) na Referência da API do Lambda.

Keep-alive para conexões persistentes

O Lambda limpa conexões ociosas ao longo do tempo, portanto, é necessário usar uma diretiva de keep-alive para manter conexões persistentes. A tentativa de reutilizar uma conexão ociosa ao invocar uma função resultará em um erro de conexão. Para manter sua conexão persistente, use a diretiva `keep-alive` associada ao tempo de execução. Para obter um exemplo, consulte [Reutilizar Conexões com keep-alive em Node.js](#) no Guia do desenvolvedor do AWS SDK for JavaScript.

Considerações sobre faturamento

Não há custo adicional para acessar uma função do Lambda por meio de um endpoint de interface. Para obter mais informações sobre o preço do Lambda, consulte [Preço do AWS Lambda](#).

O preço padrão para o AWS PrivateLink aplica-se a pontos de extremidade de interface para o Lambda. Sua conta da AWS é cobrada por cada hora que um endpoint de interface é provisionado em cada zona de disponibilidade e pelos dados processados por meio do endpoint de interface. Para obter mais informações sobre preço do endpoint de interface, consulte [Preço do AWS PrivateLink](#).

Considerações sobre emparelhamento de VPC

É possível conectar outras VPCs à VPC endpoints de interface usando o [Emparelhamento de VPC](#). O emparelhamento de VPC é uma conexão de rede entre duas VPCs. É possível estabelecer uma conexão de emparelhamento entre suas próprias duas VPCs ou com uma VPC de outra conta da AWS. As VPCs também podem estar em duas regiões da AWS diferentes.

O tráfego entre VPCs emparelhadas permanece na rede da AWS e não passa pela Internet pública. Depois que as VPCs estiverem emparelhadas, recursos como instâncias do Amazon Elastic Compute Cloud (Amazon EC2), instâncias do Amazon Relational Database Service (Amazon RDS) ou funções do Lambda habilitadas para VPC em ambas as VPCs podem acessar a API do Lambda por meio de endpoints de interface criados em uma das VPCs.

Criar um endpoint de interface para o Lambda

Você pode criar um endpoint de interface para o Lambda usando o console da Amazon VPC ou a AWS Command Line Interface(AWS CLI). Para obter mais informações, consulte [Criar um endpoint de interface](#) no Manual do usuário da Amazon VPC.

Para criar um endpoint de interface para o Lambda (console)

1. Abra a página [Endpoints](#) no console da Amazon VPC.
2. Escolha Create Endpoint.
3. Para Service category (Categoria de serviço), verifique se AWS services (Serviços da AWS) está selecionado.
4. Em Service Name (Nome do serviço), selecione com.amazonaws.**região**.lambda. Verifique se o Type (Tipo) é Interface.
5. Escolher uma VPC e sub-redes
6. Para habilitar o DNS privado para o endpoint de interface, marque a caixa de seleção Enable DNS Name (Habilitar nome de DNS).
7. Em Security group (Grupo de segurança), selecione um ou mais grupos de segurança.
8. Escolha Create endpoint (Criar endpoint).

Para usar a opção de DNS privado, defina enableDnsHostnames e enableDnsSupportAttributes da VPC. Para obter mais informações, consulte [Viewing and updating DNS support for your VPC](#) no Manual do usuário da Amazon VPC. Se você habilitar o DNS privado para o endpoint de interface, poderá fazer solicitações de API para o Lambda usando seu nome DNS padrão para a região, por exemplo, `lambda.us-east-1.amazonaws.com`. Para obter mais endpoints de serviço, consulte [Endpoints e cotas](#) no AWS Referência geral.

Para obter mais informações, consulte [Acessar um serviço por um endpoint de interface](#) no Guia do usuário da Amazon VPC.

Para obter informações sobre como criar e configurar um endpoint usando o AWS CloudFormation, consulte o recurso [AWS::EC2::VPCEndpoint](#) no Manual do usuário do AWS CloudFormation .

Para criar um endpoint de interface para o Lambda (AWS CLI)

Use o comando `create-vpc-endpoint` e especifique o ID da VPC, o tipo do VPC endpoint (interface), o nome do serviço, as sub-redes que usarão o endpoint e os grupos de segurança associados às interfaces de rede do endpoint. Por exemplo:

```
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
```

Criar uma política de endpoint de interface para o Lambda

Para controlar quem pode usar o endpoint de interface e quais funções do Lambda o usuário pode acessar, é possível anexar uma política ao endpoint. Essa política especifica as seguintes informações:

- O principal que pode executar ações.
- As ações que o principal pode executar.
- Os recursos nos quais o principal pode executar ações.

Para obter mais informações, consulte [Controlar o acesso a serviços com VPC endpoints](#) no Guia do usuário da Amazon VPC.

Exemplo: política de endpoint de interface para ações do Lambda

Veja a seguir um exemplo de uma política de endpoint para o Lambda. Quando anexada a um endpoint, essa política permite que o usuário `MyUser` invoque a função `my-function`.

Note

Você precisa incluir o ARN das funções qualificada e não qualificada no recurso.

```
{
  "Statement": [
    {
      "Principal": [
        "AWS": "arn:aws:iam::123412341234: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:)"
      ]
    }
  ]
}
```

Configurar o acesso ao banco de dados de uma função do Lambda

Você pode criar um proxy de banco de dados do Proxy do Amazon RDS para a sua função. Um proxy de banco de dados gerencia um grupo de conexões de banco de dados e retransmite consultas de uma função. Isso permite que uma função atinja altos níveis de [simultaneidade](#) (p. 20) sem esgotar conexões de banco de dados.

Seções

- [Como criar um proxy de banco de dados \(console\)](#) (p. 134)
- [Usar as permissões da função para autenticação](#) (p. 135)
- [Aplicativo de amostra](#) (p. 135)

Como criar um proxy de banco de dados (console)

Você pode usar o console do Lambda para criar um proxy de banco de dados do Proxy do Amazon RDS.

Como criar um proxy de banco de dados

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e, em seguida, Database proxies (Proxies do banco de dados).
4. Escolha Add database proxy (Adicionar proxy de banco de dados).
5. Configure as opções a seguir.
 - Identificador de proxy: o nome do proxy.
 - Instância de banco de dados do RDS: uma instância ou cluster de banco de dados [MySQL](#) ou [PostgreSQL compatível](#).
 - Segredo: um Secrets Manager segredo com o nome de usuário e senha do banco de dados.

Example secret

```
{  
    "username": "admin",  
    "password": "e2abcecxmpldc897"  
}
```

6. Escolha Adicionar.
 - Função do IAM: uma função do IAM com permissão para usar o segredo e uma política de confiança que permite ao Amazon RDS assumir a função.
 - Autenticação: o método de autenticação e autorização para se conectar ao proxy do código da função.

Pricing

O Amazon RDS cobra um preço por hora para proxies que é determinado pelo tamanho da instância do banco de dados. Para obter detalhes, consulte [Definição de preço do proxy do RDS](#).

A criação do proxy leva alguns minutos. Quando o proxy estiver disponível, configure sua função para se conectar ao endpoint do proxy em vez do endpoint do banco de dados.

A definição de preço do Amazon RDS Proxy padrão é aplicada. Para obter mais informações, consulte [Managing connections with the Amazon RDS Proxy](#) no Manual do usuário do Amazon Aurora.

Usar as permissões da função para autenticação

Por padrão, é possível se conectar a um proxy com o mesmo nome de usuário e senha usado para se conectar ao banco de dados. A única diferença no código da função é o endpoint ao qual o cliente de banco de dados se conecta. A desvantagem desse método é que é necessário expor a senha ao código da função, configurando-a em uma variável de ambiente segura ou recuperando-a do Secrets Manager.

É possível criar um proxy de banco de dados que usa as credenciais do IAM da função para autenticação e autorização, em vez de uma senha. Para usar as permissões da função para se conectar ao proxy, defina a Autenticação como Função de execução.

O console do Lambda adiciona a permissão necessária (`rds-db:connect`) à função de execução. Depois disso, é possível usar o AWS SDK para gerar um token que permite que ele se conecte ao proxy. O exemplo a seguir mostra como configurar uma conexão de banco de dados com a biblioteca `mysql2` em Node.js.

Example [dbadmin/index-iam.js](#): assinador do AWS SDK

```
const signer = new AWS.RDS.Signer({
  region: region,
  hostname: host,
  port: sqlport,
  username: username
})

exports.handler = async (event) => {
  let connectionConfig = {
    host      : host,
    user     : username,
    database : database,
    ssl: 'Amazon RDS',
    authPlugins: { mysql_clear_password: () => () => signer.getAuthToken() }
  }
  var connection = mysql.createConnection(connectionConfig)
  var query = event.query
  var result
  connection.connect()
}
```

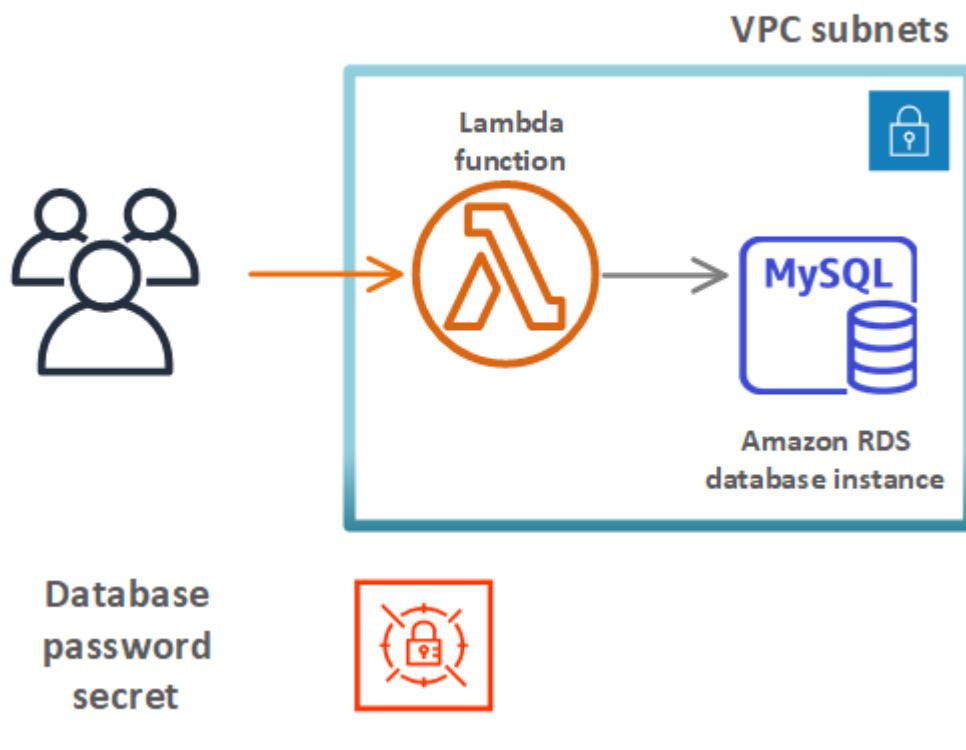
Para obter mais informações, consulte [IAM database authentication](#) no Manual do usuário do Amazon RDS.

Aplicativo de amostra

Aplicativos de exemplo que demonstram o uso do Lambda com um banco de dados do Amazon RDS estão disponíveis no repositório do GitHub deste guia. Existem dois aplicativos:

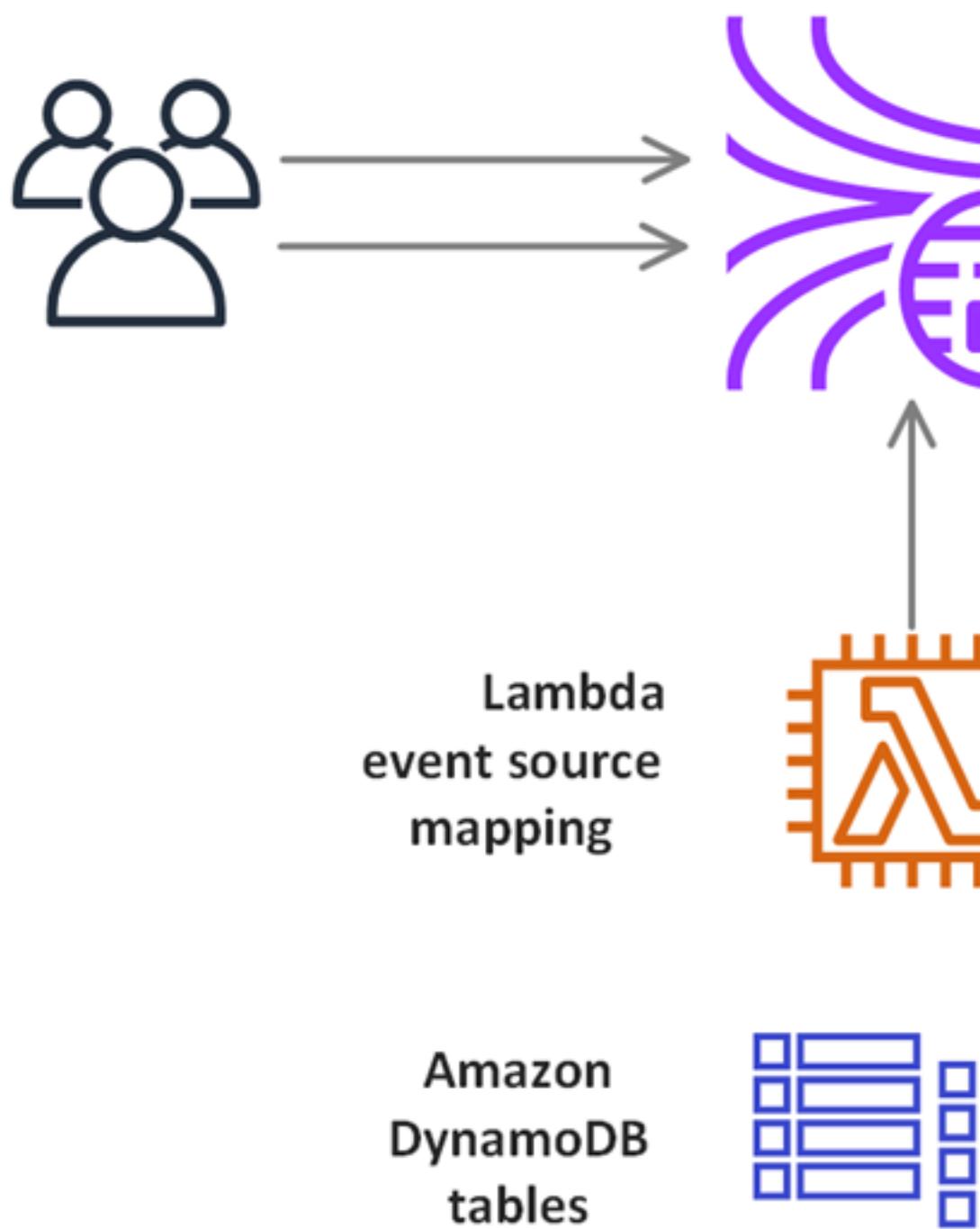
- [RDS MySQL](#): o modelo do AWS CloudFormation `template-vpcrds.yml` cria um banco de dados MySQL 5.7 em uma VPC privada. Na aplicação de exemplo, uma função do Lambda usa um proxy para consultas ao banco de dados. Os modelos de função e banco de dados usam o Secrets Manager para acessar credenciais de banco de dados.

RDS MySQL Application



- **Gerenciador de listas:** uma função de processador lê eventos de uma transmissão do Kinesis. Ele usa os dados dos eventos para atualizar tabelas do DynamoDB e armazena uma cópia do evento em um banco de dados MySQL.

List manager application



Para usar as amostras de aplicações, siga as instruções no repositório do GitHub: [RDS MySQL](#), [Gerenciador de listas](#).

Configurar o acesso ao sistema de arquivos para funções do Lambda

É possível configurar uma função para montar um sistema de arquivos Amazon Elastic File System (Amazon EFS) em um diretório local. Com o Amazon EFS, o código da função pode acessar e modificar os recursos compartilhados de forma segura e com alta simultaneidade.

Seções

- [Como conectar-se a um sistema de arquivos \(console\) \(p. 139\)](#)
- [Configurar um sistema de arquivos e ponto de acesso \(p. 140\)](#)
- [Função de execução e permissões de usuário \(p. 140\)](#)
- [Configurar o acesso ao sistema de arquivos com a API do Lambda \(p. 141\)](#)
- [AWS CloudFormation e AWS SAM \(p. 142\)](#)
- [Aplicativos de exemplo \(p. 143\)](#)

Como conectar-se a um sistema de arquivos (console)

Uma função se conecta a um sistema de arquivos pela rede local em uma VPC. As sub-redes às quais a função se conecta podem ser as mesmas sub-redes que contêm pontos de montagem para o sistema e arquivos, ou sub-redes na mesma zona de disponibilidade que pode rotear o tráfego de NFS (porta 2049) para o sistema de arquivos.

Note

Se a função ainda não estiver conectada a uma VPC, consulte [Configurar uma função do Lambda para acessar recursos em uma VPC \(p. 124\)](#).

Como configurar o acesso ao sistema de arquivos

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e Files systems (Sistemas de arquivos).
4. Em Sistema de arquivos, escolha Adicionar sistema de arquivos.
5. Configure as seguintes propriedades:
 - Sistema de arquivos do EFS: o ponto de acesso de um sistema de arquivos na mesma VPC.
 - Caminho de montagem local: o local onde o sistema de arquivos está montado na função do Lambda, começando com /mnt/.

Pricing

O Amazon EFS cobra pelo armazenamento e pela taxa de transferência, com taxas que variam de acordo com a classe de armazenamento. Para obter mais detalhes, consulte [Preço do Amazon EFS](#).

O Lambda cobra pela transferência de dados entre VPCs. Isso se aplica somente se a VPC da função for emparelhada com outra VPC em um sistema de arquivos. As taxas são as mesmas para a transferência de dados do Amazon EC2 entre VPCs na mesma região. Para obter detalhes, consulte [Preço do Lambda](#).

Para obter mais informações sobre integração do Lambda com o Amazon EFS, consulte [Uso do Amazon EFS com o Lambda \(p. 372\)](#).

Configurar um sistema de arquivos e ponto de acesso

Crie um sistema de arquivos no Amazon EFS com um destino de montagem em cada zona de disponibilidade à qual a função se conecta. Para performance e resiliência, use pelo menos duas zonas de disponibilidade. Por exemplo, em uma configuração simples, é possível ter uma VPC com duas sub-redes privadas em zonas de disponibilidade separadas. A função se conecta às duas sub-redes e um destino de montagem está disponível em cada uma delas. Verifique se o tráfego de NFS (porta 2049) é permitido pelos grupos de segurança usados pela função e pelos destinos de montagem.

Note

Ao criar um arquivo de sistemas, você escolhe um modo de performance que não pode ser alterado posteriormente. O modo Uso geral possui menor latência e o modo Máx. E/S oferece suporte a uma taxa de transferência máxima mais alta e IOPS. Para obter ajuda na escolha, consulte [Performance do Amazon EFS](#) no Manual do usuário do Amazon Elastic File System.

Um ponto de acesso conecta cada instância da função ao destino de montagem correto para a zona de disponibilidade à qual ele se conecta. Para obter a melhor performance, crie um ponto de acesso com um caminho não raiz e limite o número de arquivos criados em cada diretório. IDs de usuário e proprietário são obrigatórios, mas eles não precisam ter um valor específico. O exemplo a seguir cria um diretório chamado `my-function` no sistema de arquivos e define o ID do proprietário como 1001 com permissões de diretório padrão (755).

Example configuração do ponto de acesso

- Nome – `files`
- ID de usuário – 1001
- ID do grupo – 1001
- Caminho: `/my-function`
- Permissions – 755
- ID de usuário do proprietário: 1001
- ID de usuário do grupo: 1001

Quando uma função usa o ponto de acesso, ele recebe o ID de usuário 1001 e tem acesso total ao diretório.

Para obter mais informações, consulte os seguintes tópicos no Manual do usuário do Amazon Elastic File System:

- [Criação de recursos para o Amazon EFS](#)
- [Trabalhar com usuários, grupos e permissões](#)

Função de execução e permissões de usuário

O Lambda usa as permissões da função para montar sistemas de arquivos. Para conectar-se a um sistema de arquivos, a função de execução da sua função deve ter as seguintes permissões, além das [permissões necessárias para conectar-se à VPC do sistema de arquivos \(p. 124\)](#).

Permissões da função de execução

- `elasticfilesystem:ClientMount`
- `elasticfilesystem:ClientWrite` (não obrigatório para conexões somente leitura)

Essas permissões estão incluídas na política gerenciada
`AmazonElasticFileSystemClientReadWriteAccess`.

Quando você configura um sistema de arquivos, o Lambda usa as suas permissões para verificar os destinos de montagem. Para configurar uma função para se conectar a um sistema de arquivos, seu usuário do IAM precisa das permissões a seguir.

Permissões de usuário

- `elasticfilesystem:DescribeMountTargets`

Configurar o acesso ao sistema de arquivos com a API do Lambda

Use as seguintes operações de API para conectar a função do Lambda a um sistema de arquivos:

- [CreateFunction \(p. 796\)](#)
- [UpdateFunctionConfiguration \(p. 974\)](#)

Para conectar uma função a um sistema de arquivos, use o comando `update-function-configuration`. O exemplo a seguir conecta uma função chamada `my-function` a um sistema de arquivos com o ARN de um ponto de acesso.

```
ARN=arn:aws:elasticfilesystem:us-east-2:123456789012:access-point/fsap-015cxmplb72b405fd
aws lambda update-function-configuration --function-name my-function \
    --fs-config FileSystemArn=$ARN,LocalMountPath=/mnt/efs0
```

É possível obter o ARN de um ponto de acesso do sistema de arquivos com o comando `describe-access-points`.

```
aws efs describe-access-points
```

Você deve ver a saída a seguir:

```
{
    "AccessPoints": [
        {
            "ClientToken": "console-aa50c1fd-xmpl-48b5-91ce-57b27a3b1017",
            "Name": "lambda-ap",
            "Tags": [
                {
                    "Key": "Name",
                    "Value": "lambda-ap"
                }
            ],
            "AccessPointId": "fsap-015cxmplb72b405fd",
            "AccessPointArn": "arn:aws:elasticfilesystem:us-east-2:123456789012:access-point/fsap-015cxmplb72b405fd",
            "FileSystemId": "fs-aea3xmpl",
            "RootDirectory": {
                "Path": "/"
            },
            "OwnerId": "123456789012",
            "LifeCycleState": "available"
        }
    ]
}
```

}

AWS CloudFormation e AWS SAM

Você pode usar o AWS CloudFormation e o AWS Serverless Application Model (AWS SAM) para automatizar a criação de aplicações do Lambda. Para habilitar uma conexão do sistema de arquivos em um recurso AWS SAM do AWS::Serverless::Function, use a propriedade `FileSystemConfigs`.

Example template.yml: configuração do sistema de arquivos

```
Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16
  Subnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId:
        Ref: VPC
      CidrBlock: 10.0.1.0/24
      AvailabilityZone: "eu-central-1a"
  EfsSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      VpcId:
        Ref: VPC
      GroupDescription: "mnt target sg"
      SecurityGroupEgress:
        - IpProtocol: -1
          CidrIp: "0.0.0.0/0"
  FileSystem:
    Type: AWS::EFS::FileSystem
    Properties:
      PerformanceMode: generalPurpose
  MountTarget1:
    Type: AWS::EFS::MountTarget
    Properties:
      FileSystemId:
        Ref: FileSystem
      SubnetId:
        Ref: Subnet1
      SecurityGroups:
        - Ref: EfsSecurityGroup
  MyFunctionWithEfs:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: function/
      Description: Use a file system.
      FileSystemConfigs:
        - Arn: "arn:aws:elasticfilesystem:eu-central-1:123456789101:access-point/fsap-015cxmplb72b405fd"
          LocalMountPath: "/mnt/efs0"
      DependsOn: "MountTarget1"
```

Você deve adicionar `DependsOn` para garantir que os destinos de montagem sejam totalmente criados antes que o Lambda seja executado pela primeira vez.

Para o tipo AWS CloudFormation do AWS::Lambda::Function, o nome da propriedade e os campos são os mesmos. Para obter mais informações, consulte [Usar o AWS Lambda com o AWS CloudFormation \(p. 323\)](#).

Aplicativos de exemplo

O repositório do GitHub para este guia inclui aplicações de exemplo que demonstram o uso do Amazon EFS com uma função do Lambda.

- [efs-nodejs](#): uma função que usa um sistema de arquivos do Amazon EFS em uma Amazon VPC. Esse exemplo inclui uma VPC, um sistema de arquivos, destinos de montagem e ponto de acesso configurado para uso com o Lambda.

Configurar a assinatura de código para o AWS Lambda

A assinatura de código para AWS Lambda ajuda a garantir que apenas código confiável seja executado em suas funções do Lambda. Quando você habilita a assinatura de código para uma função, o Lambda verifica cada implantação de código e se o pacote de código é assinado por uma fonte confiável.

Note

Funções definidas como imagens de contêiner não são compatíveis com assinatura de código.

Para verificar a integridade do código, use o [AWS Signer](#) para criar pacotes de código assinados digitalmente para funções e camadas. Quando um usuário tenta implantar um pacote de código, o Lambda executa verificações de validação no pacote de código antes de aceitar a implantação. Como as verificações de validação de assinatura de código são executadas no momento da implantação, não há impacto na performance durante a execução da função.

Você também usa o AWS Signer para criar perfis de assinatura. Você usa um perfil de assinatura para criar o pacote de código assinado. Use o AWS Identity and Access Management (IAM) para controlar quem pode assinar pacotes de código e criar perfis de assinatura. Para obter mais informações, consulte [Controle de acesso e autenticação](#) no Guia do desenvolvedor do AWS Signer.

Para ativar a assinatura de código para uma função, crie uma configuração de assinatura de código e associe-a à função. Uma configuração de assinatura de código define uma lista de perfis de assinatura permitidos e a ação de política a ser executada se alguma das verificações de validação falhar.

As camadas do Lambda seguem o mesmo formato de pacote de código assinado que os pacotes de código de função. Quando você adiciona uma camada a uma função que tem a assinatura de código ativada, o Lambda verifica se a camada está assinada por um perfil de assinatura permitido. Quando você habilita a assinatura de código para uma função, todas as camadas adicionadas à função também devem ser assinadas por um dos perfis de assinatura permitidos.

Use o IAM para controlar quem pode criar configurações de assinatura de código. Normalmente, você permite que apenas usuários administrativos específicos tenham essa capacidade. Além disso, você pode configurar políticas do IAM para impor que os desenvolvedores criem apenas funções que tenham a assinatura de código ativada.

Você pode configurar a assinatura de código para registrar as alterações do AWS CloudTrail. Implantações bem-sucedidas e bloqueadas para funções são registradas no CloudTrail com informações sobre as verificações de assinatura e validação.

Você pode configurar a assinatura de código para suas funções usando o console do Lambda, a AWS Command Line Interface (AWS CLI), o AWS CloudFormation e o AWS Serverless Application Model (AWS SAM).

Não há custo adicional para usar o AWS Signer ou assinatura de código para o AWS Lambda.

Seções

- [Validação de assinatura \(p. 145\)](#)
- [Pré-requisitos de configuração \(p. 145\)](#)
- [Criar configurações de assinatura de código \(p. 145\)](#)
- [Atualizar uma configuração de assinatura de código \(p. 146\)](#)
- [Excluir uma configuração de assinatura de código \(p. 146\)](#)
- [Habilitar a assinatura de código para uma função \(p. 146\)](#)

- [Configurar políticas do IAM \(p. 147\)](#)
- [Configurar assinatura de código com a API do Lambda \(p. 148\)](#)

Validação de assinatura

O Lambda executa as seguintes verificações de validação ao implantar um pacote de código assinado na sua função:

1. Integridade — valida se o pacote de código não foi modificado desde que foi assinado. O Lambda compara o hash do pacote com o hash da assinatura.
2. Expiração valida se a assinatura do pacote de código não expirou.
3. Incompatibilidade: valida se o pacote de código está assinado com um dos perfis de assinatura permitidos para a função do Lambda. Uma incompatibilidade também ocorre se uma assinatura não estiver presente.
4. Revogação: valida se a assinatura do pacote de código não foi revogada.

A política de validação de assinatura definida na configuração da assinatura de código determina qual das seguintes ações serão executadas pelo Lambda se alguma das verificações de validação falhar:

- Avisar — O Lambda permite a implantação do pacote de código, mas emite um aviso. O Lambda emite uma nova métrica do Amazon CloudWatch e também armazena o aviso no log do CloudTrail.
- Impor: o Lambda emite um aviso (o mesmo que para a ação Avisar) e bloqueia a implantação do pacote de código.

Você pode configurar a política para as verificações de validação de expiração, incompatibilidade e revogação. Observe que não é possível configurar uma política para a verificação de integridade. Se a verificação de integridade falhar, o Lambda bloqueia a implantação.

Pré-requisitos de configuração

Antes de configurar a assinatura de código para uma função do Lambda, use o AWS Signer para fazer o seguinte:

- Criar um ou mais perfis de assinatura.
- Usar um perfil de assinatura para criar um pacote de código assinado para sua função.

Para obter mais informações, consulte [Criar perfis de assinatura \(Console\)](#) no Guia do desenvolvedor do AWS Signer.

Criar configurações de assinatura de código

Uma configuração de assinatura de código define uma lista dos perfis de assinatura permitidos e a política de validação de assinatura.

Para criar uma configuração de assinatura de código (console)

1. Abra a página [Code signing configurations](#) (Configurações de assinatura de código) do console do Lambda.
2. Selecione Create configuration (Criar configuração).
3. Em Description (Descrição), insira um nome descritivo para a configuração.
4. Em Signing profiles (Perfis de assinatura), adicione até 20 perfis de assinatura à configuração.

- a. Para Signing profile version ARN (ARN da versão do perfil de assinatura), escolha o nome de recurso da Amazon (ARN) de uma versão de perfil ou insira o ARN.
- b. Para adicionar um perfil de assinatura adicional, escolha Add signing profiles (Adicionar perfis de assinatura).
5. Em Signature validation policy (Política de validação de assinatura), escolha Warn (Avisar) ou Enforce (Impor).
6. Selecione Create configuration (Criar configuração).

Atualizar uma configuração de assinatura de código

Quando você atualiza uma configuração de assinatura de código, as alterações afetam as implantações futuras de funções que têm a configuração de assinatura de código anexada.

Para atualizar uma configuração de assinatura de código (console)

1. Abra a página [Code signing configurations](#) (Configurações de assinatura de código) do console do Lambda.
2. Selecione uma configuração de assinatura de código a ser atualizada e escolha Edit (Editar).
3. Em Description (Descrição), insira um nome descritivo para a configuração.
4. Em Signing profiles (Perfis de assinatura), adicione até 20 perfis de assinatura à configuração.
 - a. Para Signing profile version ARN (ARN da versão do perfil de assinatura), escolha o nome de recurso da Amazon (ARN) de uma versão de perfil ou insira o ARN.
 - b. Para adicionar um perfil de assinatura adicional, escolha Add signing profiles (Adicionar perfis de assinatura).
5. Em Signature validation policy (Política de validação de assinatura), escolha Warn (Avisar) ou Enforce (Impor).
6. Selecione Save changes.

Excluir uma configuração de assinatura de código

Uma configuração de assinatura de código pode ser excluída somente se nenhuma função estiver usando-a.

Para excluir uma configuração de assinatura de código (console)

1. Abra a página [Code signing configurations](#) (Configurações de assinatura de código) do console do Lambda.
2. Selecione uma configuração de assinatura de código a ser excluída e escolha Delete (Excluir).
3. Para confirmar, escolha Delete (Excluir) novamente.

Habilitar a assinatura de código para uma função

Para habilitar a assinatura de código para uma função, associe uma configuração de assinatura de código à função.

Para associar uma configuração de assinatura de código a uma função (console)

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.

2. Escolha a função para a qual deseja ativar a assinatura de código.
3. Em Code signing configuration (Configuração de assinatura de código), escolha Edit (Editar).
4. Em Edit code signing (Editar assinatura de código), escolha uma configuração de assinatura de código para esta função.
5. Escolha Save (Salvar).

Configurar políticas do IAM

Para conceder permissão a um usuário para acessar as [operações da API de assinatura de código \(p. 148\)](#), anexe uma ou mais instruções de política à política do usuário. Para obter mais informações sobre políticas de usuário, consulte [Políticas do IAM baseadas em identidade para o Lambda \(p. 67\)](#).

A instrução da política de exemplo a seguir concede permissão para criar, atualizar e recuperar configurações de assinatura de código.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "lambda:CreateCodeSigningConfig",  
                "lambda:UpdateCodeSigningConfig",  
                "lambda:GetCodeSigningConfig"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Os administradores podem usar a chave de condição `CodeSigningConfigArn` para especificar as configurações de assinatura de código que os desenvolvedores devem usar para criar ou atualizar suas funções.

O exemplo de declaração de política a seguir concede permissão para criar uma função. A declaração de política inclui um `lambda:CodeSigningConfigArn` para especificar a configuração de assinatura de código permitida. O Lambda bloqueia qualquer `CreateFunction` solicitação de API se `CodeSigningConfigArn` estiver ausente ou não corresponde ao valor na condição.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowReferencingCodeSigningConfig",  
            "Effect": "Allow",  
            "Action": [  
                "lambda:CreateFunction",  
            ],  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "lambda:CodeSigningConfigArn":  
                        "arn:aws:lambda:us-west-2:123456789012:code-signing-  
config:csc-0d4518bd353a0a7c6"  
                }  
            }  
        }  
    ]  
}
```

}

Configurar assinatura de código com a API do Lambda

Para gerenciar configurações de assinatura de código com a AWS CLI ou o AWS SDK ou, use as seguintes operações de API:

- [ListCoDesigningConfigs](#)
- [CreateCoDesigningConfig](#)
- [GetCoDesigningConfig](#)
- [UpdateCoDesigningConfig](#)
- [DeleteCoDesigningConfig](#)

Para gerenciar a configuração de assinatura de código de uma função, use as seguintes operações de API:

- [CreateFunction \(p. 796\)](#)
- [getFunctionCoDesigningConfig](#)
- [putFunctionCoDesigningConfig](#)
- [deleteFunctionCoDesigningConfig](#)
- [ListFunctionsByCoDesigningConfig](#)

Etiquetar funções do Lambda

Você pode etiquetar funções do Lambda para organizá-las por proprietário, projeto ou departamento. As tags são pares de chave-valor com formato livre, compatíveis com serviços da AWS, usados para filtrar recursos e adicionar detalhes aos relatórios de faturamento.

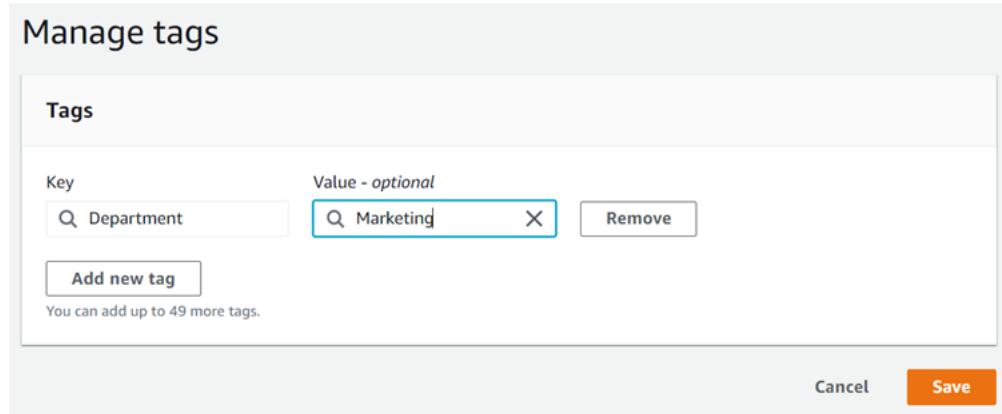
Seções

- [Como adicionar tags a uma função \(console\) \(p. 149\)](#)
- [Como usar tags para filtrar funções \(console\) \(p. 149\)](#)
- [Como usar tags com a AWS CLI \(p. 150\)](#)
- [Requisitos de chave de tag e valor \(p. 151\)](#)

Como adicionar tags a uma função (console)

Para adicionar tags a uma função

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e, em seguida, Tags.
4. Em Tags, selecione Manage tags (Gerenciar tags).
5. Insira um par de chave e valor. Para adicionar mais tags, selecione Add new tag (Adicionar nova tag).



6. Escolha Save (Salvar).

Como usar tags para filtrar funções (console)

Você pode filtrar funções com base na presença ou no valor de uma tag usando o console do Lambda ou a API do AWS Resource Groups. As tags se aplicam no nível da função, não em versões ou aliases. As tags não fazem parte da configuração específica da versão da qual é feita um snapshot quando você publica uma versão.

Como filtrar funções com tags

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Clique na barra de pesquisa para ver uma lista de atributos de função e chaves de tag.

The screenshot shows a table titled 'Functions (75)'. The columns are 'Function attributes' (Description, Function name, Master ARN, Runtime), 'Tags' (aws:cloudformation:logical-id, aws:cloudformation:stack-id, aws:cloudformation:stack-name, awscodestar:projectArn), and 'Department' (highlighted with a mouse cursor). A search bar at the top says 'Filter by tags and attributes or search by keyword'.

3. Selecione uma chave de tag para ver uma lista de valores que estão sendo usados na região atual.
4. Escolha um valor para ver funções correspondentes ou selecione (all values) (todos os valores) para ver todas as funções que têm uma tag com essa chave.

The screenshot shows a table titled 'Functions (75)' with a search bar containing 'tag:Department :'. The results show three items: '(all values)', '(empty)', and 'Customer support', 'Marketing', 'Sales' (highlighted with a mouse cursor). A dropdown arrow icon is next to the 'Marketing' entry.

A barra de pesquisa também oferece suporte para a pesquisa de chaves de tag. Digite tag para ver somente uma lista de chaves de tag ou comece a digitar o nome da chave para encontrá-la na lista.

Com o AWS Billing and Cost Management, você pode usar tags para personalizar relatórios de faturamento e criar relatórios de alocação de custos. Para obter mais informações, consulte o [Relatório mensal de alocação de custos](#) e [Usar etiquetas de alocação de custos](#) no Manual do usuário do AWS Billing and Cost Management.

Como usar tags com a AWS CLI

Ao criar uma função do Lambda, é possível incluir etiquetas com a opção `--tags`.

```
aws lambda create-function --function-name my-function
--handler index.js --runtime nodejs12.x \
--role arn:aws:iam::123456789012:role/lambda-role \
```

```
--tags Department=Marketing,CostCenter=1234ABCD
```

Para adicionar tags a uma função existente, use o comando `tag-resource`.

```
aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:function:my-function \
--tags Department=Marketing,CostCenter=1234ABCD
```

Para remover tags, use o comando `untag-resource`.

```
aws lambda untag-resource --resource function arn \
--tag-keys Department
```

Se você deseja visualizar as etiquetas que são aplicadas à uma função do Lambda específica, você pode usar os seguintes comandos de API do Lambda:

- [ListTags \(p. 909\)](#): você fornece o ARN (nome de recurso da Amazon) de sua função do Lambda para visualizar uma lista das etiquetas associadas a essa função:

```
aws lambda list-tags --resource function arn
```

- [GetFunction \(p. 842\)](#): você fornece o nome de sua função do Lambda para visualizar uma lista das etiquetas associadas a essa função:

```
aws lambda get-function --function-name my-function
```

Você também pode usar a API [GetResources](#) do serviço de atribuição de tags da AWS para filtrar seus recursos por tags. A API GetResources aceita até 10 filtros, cada filtro contendo uma chave de tags e até 10 valores de tag. Você fornece GetResources com um 'ResourceType' para filtrar por tipos de recursos específicos. Para obter mais informações sobre o serviço de atribuição de etiquetas da AWS, consulte [Trabalhar com Resource Groups](#).

Requisitos de chave de tag e valor

Os seguintes requisitos são aplicáveis às tags:

- Número máximo de tags por recurso: 50
- Comprimento máximo da chave: 128 caracteres Unicode em UTF-8
- Valor máximo da chave: 256 caracteres Unicode em UTF-8
- As chaves e os valores de tags diferenciam maiúsculas de minúsculas.
- Não use o prefixo `aws:` no nome nem no valor de suas tags, pois ele é reservado para uso da AWS. Você não pode editar nem excluir nomes ou valores de tag com esse prefixo. As tags com esse prefixo não contam para as tags por limite de recurso.
- Se seu esquema de tags for usado em vários serviços e recursos, lembre-se de que outros serviços podem ter restrições quanto a caracteres permitidos. No geral, os caracteres permitidos são letras, espaços e números representáveis em UTF-8, além dos seguintes caracteres especiais: + - = . _ : / @.

Usar camadas com sua função do Lambda

Uma camada do Lambda é um arquivo .zip que pode conter código adicional ou outro conteúdo. Uma camada pode conter bibliotecas, um tempo de execução personalizado, dados ou arquivos de configuração. Use camadas para reduzir o tamanho do pacote de implantação e promover o compartilhamento de código e a separação de responsabilidades para que você possa iterar mais rapidamente na escrita da lógica de negócios.

Você pode usar camadas somente com funções do Lambda [implantadas como um arquivo.zip \(p. 37\)](#). Para uma função [definida como uma imagem de contêiner \(p. 266\)](#), você pode empacotar seu tempo de execução preferido e todas as dependências de código ao criar a imagem de contêiner. Para obter mais informações, consulte [Working with Lambda layers and extensions in container images](#) no Blog de computação da AWS.

Seções

- [Configurar uma função para usar camadas \(p. 152\)](#)
- [Acessar o conteúdo de uma camada \(p. 153\)](#)
- [Localizar informações da camada \(p. 153\)](#)
- [Atualizar uma versão de camada utilizada por sua função \(p. 154\)](#)
- [Adicionar permissões de camada \(p. 155\)](#)
- [Usar o AWS SAM para adicionar uma camada a uma função \(p. 89\)](#)
- [Aplicativos de exemplo \(p. 155\)](#)

Configurar uma função para usar camadas

Uma função do Lambda pode usar até cinco camadas por vez. O tamanho total descompactado da função e de todas as camadas não pode exceder a cota de tamanho do pacote de implantação descompactado de 250 MB. Para obter mais informações, consulte [Cotas Lambda \(p. 53\)](#).

Se suas funções consomem uma camada que uma conta da AWS diferente publica, suas funções poderão continuar a usar a versão da camada depois que ela tiver sido excluída ou depois que sua permissão para acessar a camada for revogada. No entanto, você não pode criar uma nova função que use uma versão de camada excluída.

Para adicionar camadas à sua função, use o comando `update-function-configuration`. O exemplo a seguir adiciona duas camadas: um na mesma conta da AWS como a função e uma de outra conta.

```
aws lambda update-function-configuration --function-name my-function \
--layers arn:aws:lambda:us-east-2:123456789012:layer:my-layer:3
 \
arn:aws:lambda:us-east-2:210987654321:layer:their-layer:2
```

Você deve ver saída semelhante a:

```
{
  "FunctionName": "test-layers",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
  "Runtime": "nodejs12.x",
  "Role": "arn:aws:iam::123456789012:role/service-role/lambda-role",
  "Layers": [
    {
      "Arn": "arn:aws:lambda:us-east-2:123456789012:layer:my-layer:3",
      "CodeSize": 169
```

```
        },
        {
            "Arn": "arn:aws:lambda:us-east-2:210987654321:layer:their-layer:2",
            "CodeSize": 169
        }
    ],
    "RevisionId": "81cc64f5-5772-449a-b63e-12330476bcc4",
    ...
}
```

Para especificar as versões de camada a serem usadas, você deve fornecer o nome do recurso da Amazon (ARN) completo de cada versão da camada. Quando você adiciona camadas a uma função que já tem camadas, a nova lista substitui a anterior. Certifique-se de incluir todas as camadas sempre que atualizar a configuração da camada. Ou, para remover todas as camadas, especifique uma lista vazia.

```
aws lambda update-function-configuration --function-name my-function --layers []
```

O criador de uma camada pode excluir uma versão dela. Se você estiver usando essa versão de camada em uma função, sua função continuará a ser executada como se a versão da camada ainda existisse. No entanto, quando você atualizar a configuração da camada, precisará remover a referência à versão excluída.

Acessar o conteúdo de uma camada

Quando você inclui uma camada em uma função do Lambda, o Lambda extrai o conteúdo da camada para o diretório /opt no ambiente de execução da função. O Lambda extrai as camadas na ordem especificada, mesclando eventuais pastas com o mesmo nome. Se o mesmo arquivo aparecer em várias camadas, a função usará a versão na última camada extraída.

Cada [tempo de execução do Lambda \(p. 214\)](#) adiciona pastas de diretório /opt específicas à variável PATH. Seu código de função pode acessar o conteúdo da camada sem a necessidade de especificar o caminho. Para obter mais informações sobre as configurações de caminhos no ambiente de execução do Lambda, consulte [Variáveis de ambiente com tempo de execução definido \(p. 102\)](#).

Localizar informações da camada

Para encontrar camadas em sua conta da AWS que sejam compatíveis com o tempo de execução de sua função do Lambda, use o comando list-layers.

```
aws lambda list-layers --compatible-runtime python3.8
```

Você deve ver saída semelhante a:

```
{
    "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": "2018-11-15T00:37:46.592+0000",
                "CompatibleRuntimes": [
                    "python3.6",
                    "python3.7",

```

```
        "python3.8",
    ]
}
]
```

Para listar todas as camadas da sua conta, você pode omitir a opção `--compatible-runtime`. Os detalhes na resposta refletem a versão mais recente da camada.

Você também pode obter a versão mais recente de uma camada usando o comando `list-layer-versions`.

```
aws lambda list-layer-versions --layer-name my-layer --query
'LayerVersions[0].LayerVersionArn'
```

Atualizar uma versão de camada utilizada por sua função

As camadas são versionadas e o conteúdo de cada versão de camada é imutável. O proprietário da camada pode liberar uma nova versão dela para fornecer conteúdo atualizado.

Use o comando `update-function-configuration` para adicionar uma versão de camada atualizada à sua função. Use a opção `--layers` com esse comando para listar todas as versões de camada que você deseja adicionar. Se a função já tiver camadas, a nova lista substitui a anterior.

Para atualizar apenas uma das versões de camada, inclua os ARNs das versões de camada existentes com a opção `--layers`.

O procedimento a seguir pressupõe que você tenha compactado o código da camada atualizado em um arquivo local com o nome `layer.zip`.

Adicione uma versão de camada atualizada à sua função

1. (Opcional) Se a nova versão da camada ainda não tiver sido publicada, publique-a.

```
aws lambda publish-layer-version --layer-name my-layer --description "My layer" --
license-info "MIT" \
--zip-file "fileb://layer.zip" --compatible-runtimes python3.6 python3.7
```

2. (Opcional) Se a função tiver mais de uma camada, obtenha as versões atuais da camada associadas à função.

```
aws lambda get-function-config --function-name my-function --query 'Layers[*].Arn' --
output yaml
```

3. Adicione a nova versão da camada à função. No comando de exemplo a seguir, a função também tem uma versão de camada com o nome `other-layer:5`:

```
aws lambda update-function-configuration --function-name my-function \
--layers arn:aws:lambda:us-east-2:123456789012:layer:my-layer:2 \
arn:aws:lambda:us-east-2:123456789012:layer:other-layer:5
```

Adicionar permissões de camada

Para usar uma função do Lambda com uma camada, você precisa de permissão para chamar a operação da API [GetLayerVersion \(p. 861\)](#) na versão da camada. Para funções em sua conta da AWS, você pode adicionar essa permissão de sua [política de usuário \(p. 67\)](#).

Para usar uma camada em outra conta, o proprietário dessa conta deve conceder permissão à sua conta em uma [política baseada em recursos \(p. 62\)](#).

Para ver exemplos, consulte [Conceder acesso de camada a outras contas \(p. 65\)](#).

Usar o AWS SAM para adicionar uma camada a uma função

Para automatizar a criação e o mapeamento de camadas em sua aplicação, use o AWS Serverless Application Model (AWS SAM). O tipo de recurso `AWS::Serverless::LayerVersion` cria uma versão de camada à qual você pode fazer referência na configuração da função do Lambda.

Example [blank-nodejs/template.yml](#): recursos sem servidor

```
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:
      Handler: index.handler
      Runtime: nodejs12.x
      CodeUri: function/
      Description: Call the AWS Lambda API
      Timeout: 10
      # Function's execution role
      Policies:
        - AWSLambdaBasicExecutionRole
        - AWSLambda_ReadOnlyAccess
        - AWSXrayWriteOnlyAccess
      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:
        - nodejs12.x
```

Ao atualizar suas dependências e implantar, o AWS SAM cria uma nova versão da camada e atualiza o mapeamento.

Aplicativos de exemplo

O repositório do GitHub para este guia fornece exemplos de aplicações em branco que demonstram o uso de camadas para o gerenciamento de dependências.

- Node.js – [blank-nodejs](#)

- Python – [blank-python](#)
- Ruby – [blank-ruby](#)
- Java – [blank-java](#)

Para obter mais informações sobre a aplicação amostral em branco, consulte [Aplicativo de exemplo de função em branco para o AWS Lambda \(p. 493\)](#). Para outros exemplos, consulte [Amostras \(p. 491\)](#).

Chamada de funções do AWS Lambda

Você pode invocar as funções do Lambda diretamente [com o console do Lambda \(p. 9\)](#), a API do Lambda, o AWS SDK, a AWS CLI e os toolkits da AWS. Você também pode configurar outros serviços da AWS para invocar a função ou configurar o Lambda para ler de uma transmissão ou uma fila e chamar sua função.

Quando você invocar uma função, poderá optar por invocá-la de forma síncrona ou assíncrona. Com a [invocação síncrona \(p. 158\)](#), você aguarda a função processar o evento e retornar uma resposta. Com a [invocação assíncrona \(p. 161\)](#), o Lambda coloca o evento na fila para processamento e retorna uma resposta imediatamente. Para chamada assíncrona, o Lambda manipula novas tentativas e pode enviar registros de chamada para um [destino \(p. 164\)](#).

Para usar a função a fim de processar dados automaticamente, adicione um ou mais triggers. Um acionador é um recurso do Lambda ou um recurso em outro serviço que você configura para invocar a função em resposta a eventos de ciclo de vida, solicitações externas ou em uma programação. A função pode ter vários triggers. Cada trigger atua como um cliente invocando a função de modo independente. Cada evento que o Lambda transmite à função só tem dados de um cliente ou acionador.

Para processar itens de um stream ou fila, você pode criar um [mapeamento de fonte de eventos \(p. 170\)](#). O mapeamento de uma fonte de eventos é um recurso no Lambda que lê os itens de uma fila do Amazon SQS, de um Amazon Kinesis Stream ou de um Amazon DynamoDB Streams, e os envia para sua função em lotes. Cada evento que seus processos de função pode conter centenas ou milhares de itens.

Outros serviços e recursos da AWS invocam a função diretamente. Por exemplo, você pode configurar o CloudWatch Events para invocar sua função em um temporizador ou configurar o Amazon S3 para chamar sua função quando um objeto é criado. Cada serviço varia no método usado para invocar sua função, a estrutura do evento e como configurá-lo. Para obter mais informações, consulte [Usar o AWS Lambda com outros serviços \(p. 277\)](#).

Dependendo de quem chama a função e de como ela é chamada, o comportamento da dimensionamento e os tipos de erros que podem ocorrer variam. Quando você invocar uma função de forma síncrona, receberá erros na resposta e poderá tentar novamente. Quando você invoca de forma assíncrona, usa o mapeamento de uma origem de evento ou configura outro serviço para invocar a função, os requisitos de nova tentativa e a maneira como sua função pode ser dimensionada para lidar com um grande número de eventos variam. Para obter mais detalhes, consulte [Escalabilidade da função do Lambda \(p. 32\)](#) e [Lidar com erros e novas tentativas automáticas no AWS Lambda \(p. 176\)](#).

Tópicos

- [Invocação síncrona \(p. 158\)](#)
- [Invocação assíncrona \(p. 161\)](#)
- [AWS LambdaMapeamentos da fonte de eventos do \(p. 170\)](#)
- [Monitorar o estado de uma função com a API do Lambda \(p. 174\)](#)
- [Lidar com erros e novas tentativas automáticas no AWS Lambda \(p. 176\)](#)
- [Uso de extensões do Lambda \(p. 178\)](#)
- [Invocar funções definidas como imagens de contêiner \(p. 182\)](#)
- [Chamada de funções do Lambda com oAWS Mobile SDK for Android \(p. 183\)](#)

Invocação síncrona

Quando você invoca uma função de forma síncrona, o Lambda executa a função e aguarda uma resposta. Quando a função é concluída, o Lambda retorna a resposta do código da função com dados adicionais, como a versão da função que foi invocada. Para invocar uma função de forma síncrona com o AWS CLI, use o comando `invoke`.

```
aws lambda invoke --function-name my-function --payload '{ "key": "value" }' response.json
```

Você deve ver a saída a seguir:

```
{  
    "ExecutedVersion": "$LATEST",  
    "StatusCode": 200  
}
```

O diagrama a seguir mostra clientes invocando uma função do Lambda de forma síncrona. O Lambda envia os eventos diretamente para a função e envia a resposta da função de volta para o invocador.

Synchronous Invocation



O `payload` é uma string que contém um evento no formato JSON. O nome do arquivo em que o AWS CLI grava a resposta da função é `response.json`. Se a função retorna um objeto ou erro, a resposta é o objeto ou erro no formato JSON. Se a função é encerrada sem erros, a resposta é `null`.

A saída do comando, que é exibida no terminal, inclui informações de cabeçalhos na resposta do Lambda. Isso inclui a versão que processou o evento (útil quando você usa [aliases \(p. 108\)](#)) e o código de status retornado pelo Lambda. Se o Lambda foi capaz de executar a função, o código de status é 200, mesmo que a função tenha retornado um erro.

Note

Para funções com um longo tempo limite, o cliente pode ser desconectado durante a invocação síncrona enquanto aguarda por uma resposta. Configure seu cliente HTTP, SDK, firewall, proxy ou sistema operacional para permitir conexões longas com tempo limite ou configurações de ativação.

Se o Lambda não for capaz de executar a função, o erro será exibido na saída.

```
aws lambda invoke --function-name my-function --payload value response.json
```

Você deve ver a saída a seguir:

```
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]
```

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Você pode usar a [AWS CLI](#) para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Você deve ver a saída a seguir:

```
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZimjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
```

Você deve ver a saída a seguir:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e [Ubuntu no Windows](#). Os usuários do macOS precisam usar `base64 -D`.

Para obter mais informações sobre a API `Invoke`, incluindo uma lista completa de parâmetros, cabeçalhos e erros, consulte [Invoke \(p. 875\)](#).

Quando invocar uma função diretamente, você poderá verificar a resposta quanto a erros e tentar novamente. A AWS CLI e o AWS SDK também realizam automaticamente novas tentativas em erros de serviço, controle de utilização e tempo limite do cliente. Para obter mais informações, consulte [Lidar com erros e novas tentativas automáticas no AWS Lambda \(p. 176\)](#).

Invocação assíncrona

Vários AWS OS serviços, como o Amazon Simple Storage Service (Amazon S3) e o Amazon Simple Notification Service (Amazon SNS), chamam funções de forma assíncrona para processar eventos. Ao invocar uma função de forma assíncrona, não aguarde uma resposta do código da função. Você entrega o evento para o Lambda e ele cuida do resto. É possível configurar como o Lambda processa os erros e enviar registros de invocação para um recurso de downstream a fim de encadear componentes de sua aplicação.

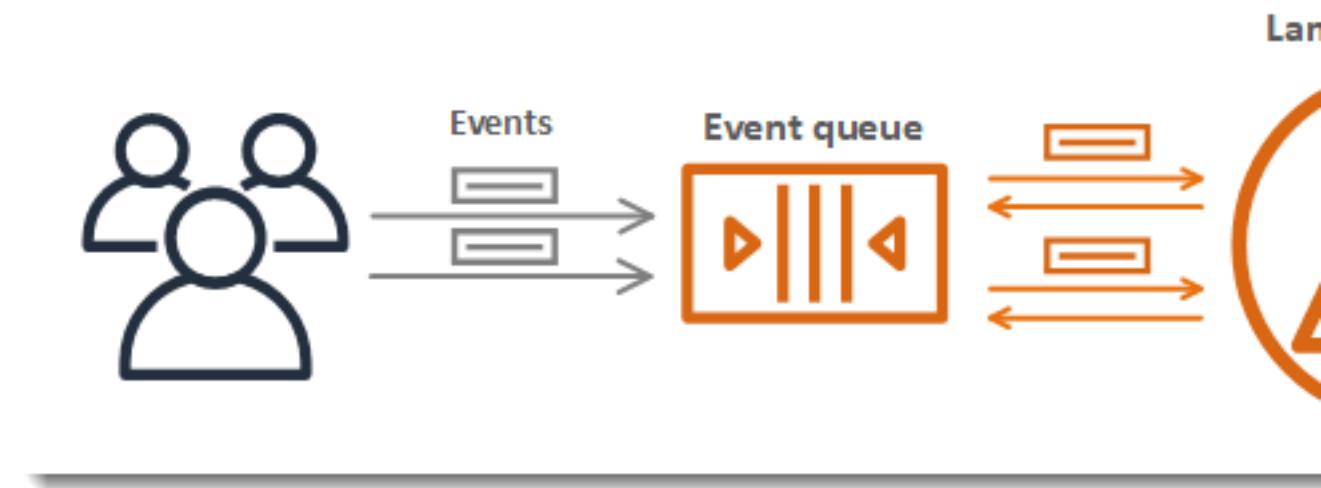
Seções

- [Como o Lambda trabalha com invocações assíncronas \(p. 161\)](#)
- [Configurar o tratamento de erros para invocação assíncrona \(p. 163\)](#)
- [Configurar destinos para invocação assíncrona \(p. 164\)](#)
- [API de configuração de invocação assíncrona \(p. 166\)](#)
- [Filas de mensagens mortas \(p. 167\)](#)

Como o Lambda trabalha com invocações assíncronas

O diagrama a seguir mostra clientes invocando uma função do Lambda de forma assíncrona. O Lambda coloca os eventos em fila antes de enviá-los para a função.

Asynchronous Invocation



Para invocação de forma assíncrona, o Lambda coloca o evento em uma fila e retorna uma resposta bem-sucedida sem informações adicionais. Um processo separado lê os eventos na fila e os envia para sua função. Para invocar uma função de maneira assíncrona, defina o parâmetro do tipo de invocação como `Event`.

```
aws lambda invoke \
--function-name my-function \
```

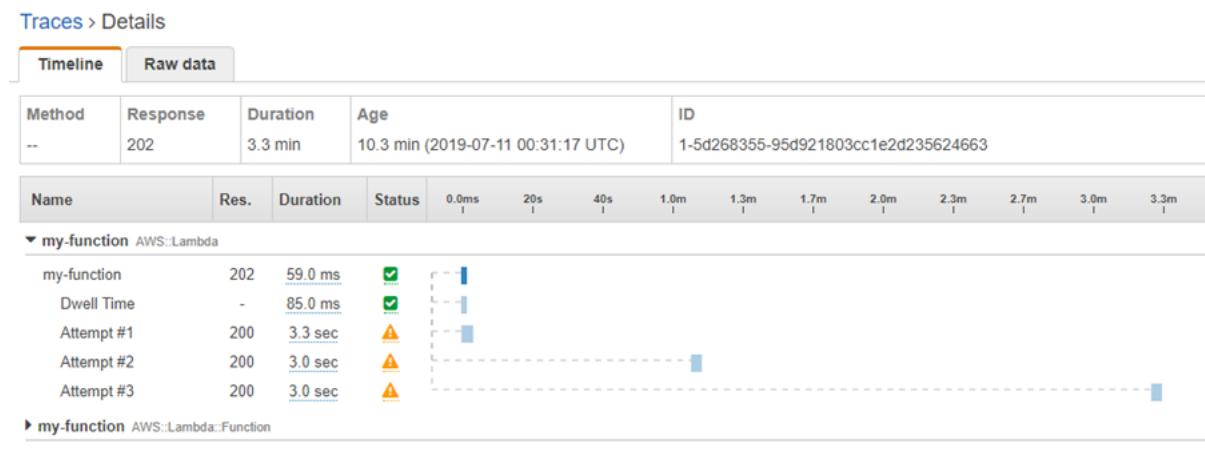
```
--invocation-type Event \
--cli-binary-format raw-in-base64-out \
--payload '{ "key": "value" }' response.json
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

```
{
    "StatusCode": 202
}
```

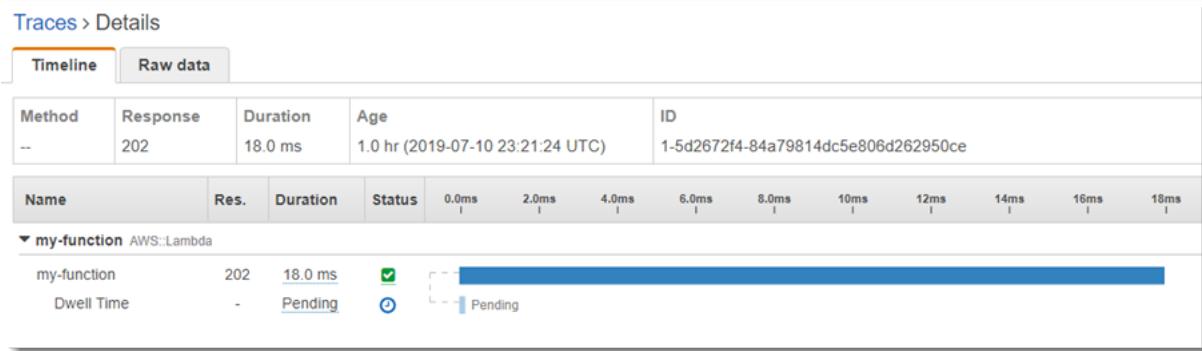
O arquivo de saída (`response.json`) não contém informações, mas continuará sendo criado quando esse comando for executado. Se o Lambda não conseguir adicionar o evento a uma fila, a mensagem de erro será exibida na saída do comando.

O Lambda gerencia a fila de eventos assíncronos da função e realiza novas tentativas em casos de erro. Se a função retornar um erro, o Lambda tentará executá-la mais duas vezes, com um intervalo de um minuto entre a primeira e a segunda tentativa, e um intervalo de dois minutos entre a segunda e a terceira. Os erros de função incluem erros retornados pelo código e pelo tempo de execução da função, como o tempo limite ser atingido.



Se a função não tiver simultaneidade suficiente disponível para processar todos os eventos, as solicitações adicionais serão limitadas. Para erros de controle de utilização (429) e de sistema (série 500), o Lambda retorna o evento para a fila e tenta executar a função novamente por até 6 horas. O intervalo de repetição aumenta exponencialmente de 1 segundo após a primeira tentativa para no máximo 5 minutos. Se a fila contém muitas entradas, o Lambda aumenta o intervalo de repetição e reduz a taxa em que lê eventos da fila.

O exemplo a seguir mostra um evento que foi adicionado com êxito à fila, mas ainda está pendente uma hora depois devido à limitação.



Mesmo que a função não retorne um erro, é possível que ela receba o mesmo evento do Lambda várias vezes, porque a própria fila se tornará consistente. Se a função não conseguir acompanhar os eventos recebidos, é possível que eventos sejam excluídos da fila sem serem enviados para a função. Certifique-se de que seu código de função lide corretamente com eventos duplicados, e de que você tenha simultaneidade suficiente disponível para lidar com todas as invocações.

Quando a fila é muito longa, novos eventos podem expirar antes de o Lambda ter a chance de enviá-los para sua função. Quando há falha em todas as tentativas de processamento de um evento ou ele expira, o Lambda o descarta. É possível [configurar o tratamento de erros \(p. 163\)](#) para uma função a fim de reduzir o número de tentativas que o Lambda executa ou descartar eventos não processados mais rapidamente.

Também é possível configurar o Lambda para enviar um registro de invocação para outro serviço. O Lambda suporta o seguinte[Destinos \(p. 164\)](#)para invocação assíncrona.

- Amazon SQS— Uma fila SQS padrão.
- Amazon SNS— Um tópico do SNS.
- AWS Lambda: uma função do Lambda.
- Amazon EventBridge: u barramento de eventos do EventBridge.

O registro de invocação contém detalhes sobre a solicitação e a resposta no formato JSON. É possível configurar destinos separados para eventos que são processados com êxito e eventos em que há falha no processamento de todas as tentativas. Como alternativa, é possível configurar uma fila do Amazon SQS ou um tópico do Amazon SNS como uma [fila de mensagens mortas \(p. 167\)](#) para os eventos descartados. Para filas de mensagens mortas, o Lambda envia somente o conteúdo do evento, sem detalhes sobre a resposta.

Note

Para evitar que uma função seja acionada, você pode definir a simultaneidade reservada da função como zero. Quando você define a simultaneidade reservada como zero para uma função invocada de forma assíncrona, o Lambda envia imediatamente todos os eventos para a [fila de mensagens mortas \(p. 167\)](#) configurada ou para o [destino do evento \(p. 164\)](#) em falha, sem novas tentativas. Para processar eventos que foram enviados enquanto a simultaneidade reservada foi definida como zero, você precisa consumir os eventos da fila de mensagens mortas ou do destino do evento em falha.

Configurar o tratamento de erros para invocação assíncrona

Use o console do Lambda para definir as configurações de tratamento de erros em uma função, uma versão ou um alias.

Como configurar o tratamento de erros

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e, em seguida, Asynchronous invocation (Invocação assíncrona).
4. Em Asynchronous invocation (Invocação assíncrona), escolha Edit (Editar).
5. Configure as definições a seguir.
 - Maximum age of event (Idade máxima do evento): a quantidade máxima de tempo que o Lambda retém um evento na fila de eventos assíncronos, até 6 horas.
 - Retry attempts (Tentativas de repetição): o número de vezes que o Lambda tenta novamente quando a função retorna um erro, entre 0 e 2.
6. Escolha Save (Salvar).

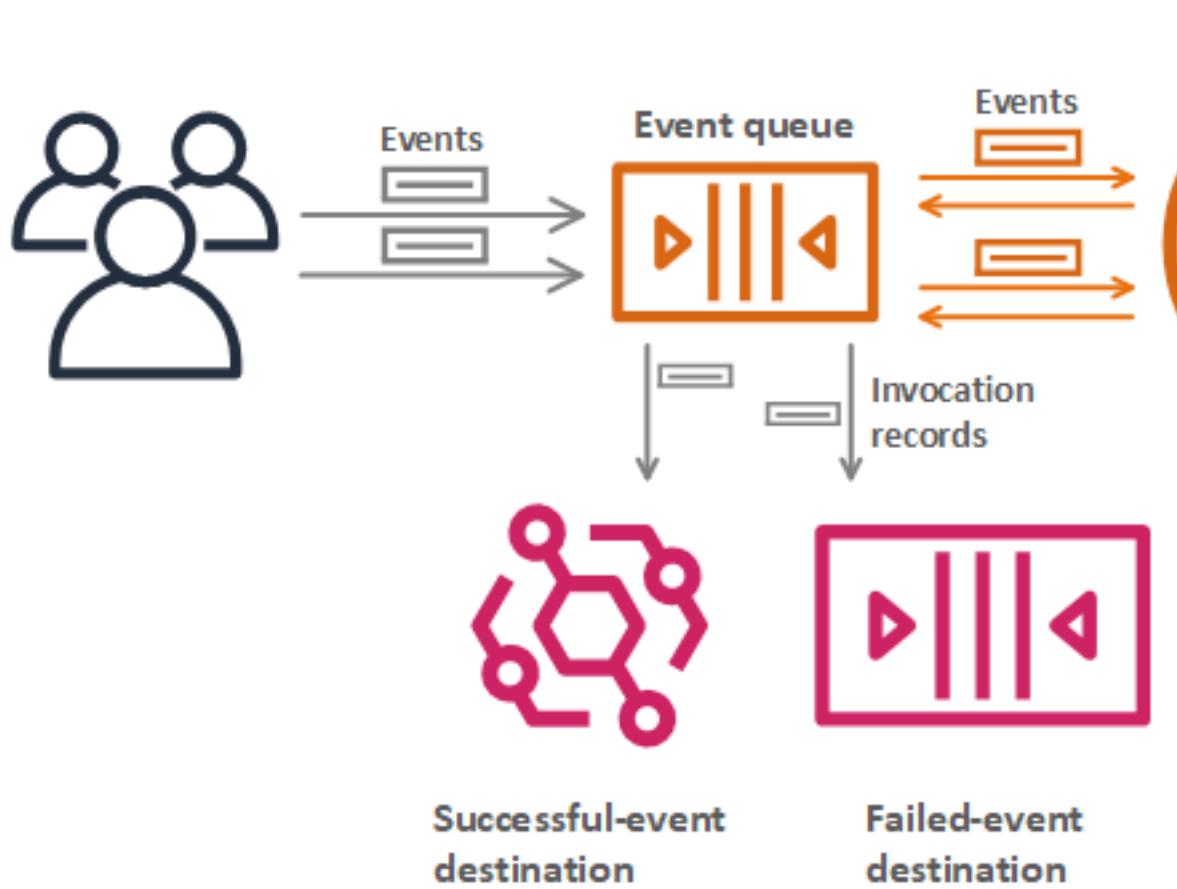
Quando um evento de invocação exceder a idade máxima ou falhar em todas as tentativas de repetição, o Lambda o descartará. Para reter uma cópia de eventos descartados, configure um destino de evento com falha.

Configurar destinos para invocação assíncrona

Para enviar registros de invocações assíncronas para outro serviço, adicione um destino à sua função. É possível configurar destinos separados para eventos que falham no processamento e eventos que são processados com êxito. Como configurações de tratamento de erros, é possível configurar destinos em uma função, em uma versão ou em um alias.

O exemplo a seguir mostra uma função que está processando chamadas assíncronas. Quando a função retorna uma resposta bem-sucedida ou sai sem lançar um erro, o Lambda envia um registro da invocação para um barramento de eventos do EventBridge. Quando um evento falha em todas as tentativas de processamento, o Lambda envia um registro de chamada para uma fila do Amazon SQS.

Destinations for Asynchronous Invocation



Para enviar eventos a um destino, sua função precisa de permissões adicionais. Adicione uma política com as permissões necessárias à [função de execução \(p. 57\)](#) de sua função. Cada serviço de destino requer uma permissão diferente, como se segue:

- Amazon SQS – `sqs:SendMessage`
- Amazon SNS – `sns:Publish`
- Lambda – [InvokeFunction \(p. 875\)](#)
- EventBridge–`events:PutEvents`

Adicione destinos à sua função na visualização de funções do console do Lambda.

Como configurar um destino para registros de invocação assíncrona

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Em Function overview (Visão geral da função), escolha Add destination (Adicionar destino).

4. Em Source (Origem), escolha Asynchronous invocation (Invocação assíncrona).
5. Em Condition (Condição), escolha uma das seguintes opções:
 - On failure (Em caso de falha): envie um registro quando o evento falhar em todas as tentativas de processamento ou exceder a idade máxima.
 - On success (Em caso de êxito): envie um registro quando a função processa com êxito uma invocação assíncrona.
6. Em Destination type (Tipo de destino), escolha o tipo de recurso que recebe o registro da invocação.
7. Em Destination (Destino), escolha um recurso.
8. Escolha Save (Salvar).

Quando uma invocação corresponde à condição, o Lambda envia um documento JSON com detalhes sobre a invocação para o destino. O exemplo a seguir mostra um registro de invocação para um evento que teve três falhas de tentativa de processamento devido a um erro de função.

Example registro de invocação

```
{  
    "version": "1.0",  
    "timestamp": "2019-11-14T18:16:05.568Z",  
    "requestContext": {  
        "requestId": "e4b46cbf-b738-xmpl-8880-a18cdf61200e",  
        "functionArn": "arn:aws:lambda:us-east-2:123456789012: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"  
    }  
}
```

O registro de invocação contém detalhes sobre o evento, a resposta e o motivo pelo qual o registro foi enviado.

API de configuração de invocação assíncrona

Para gerenciar configurações de invocação assíncrona com a AWS CLI ou o AWS SDK, use as operações de API a seguir.

- [PutFunctionEventInvoke](#)
- [GetFunctionEventInvokeConfig](#)
- [UpdateFunctionEventInvokeConfig](#)
- [ListFunctionEventInvokeConfigs](#)

- [DeleteFunctionEventInvoke](#)

Para configurar a invocação assíncrona com a AWS CLI, use o comando `put-function-event-invoke-config`. O exemplo a seguir configura uma função com uma idade máxima de evento de 1 hora e nenhuma repetição.

```
aws lambda put-function-event-invoke-config --function-name error \
--maximum-event-age-in-seconds 3600 --maximum-retry-attempts 0
```

Você deve ver a saída a seguir:

```
{  
    "LastModified": 1573686021.479,  
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:error:$LATEST",  
    "MaximumRetryAttempts": 0,  
    "MaximumEventAgeInSeconds": 3600,  
    "DestinationConfig": {  
        "OnSuccess": {},  
        "OnFailure": {}  
    }  
}
```

O comando `put-function-event-invoke-config` substitui qualquer configuração existente na função, versão ou alias. Para configurar uma opção sem redefinir outras, use `update-function-event-invoke-config`. O exemplo a seguir configura o Lambda para enviar um registro a uma fila do SQS denominada `destination` quando um evento não puder ser processado.

```
aws lambda update-function-event-invoke-config --function-name error \
--destination-config '{"OnFailure":{"Destination": "arn:aws:sqs:us-
east-2:123456789012:destination"}}'
```

Você deve ver a saída a seguir:

```
{  
    "LastModified": 1573687896.493,  
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:error:$LATEST",  
    "MaximumRetryAttempts": 0,  
    "MaximumEventAgeInSeconds": 3600,  
    "DestinationConfig": {  
        "OnSuccess": {},  
        "OnFailure": {  
            "Destination": "arn:aws:sqs:us-east-2:123456789012:destination"  
        }  
    }  
}
```

Filas de mensagens mortas

Como alternativa a um destino em caso de falha (p. 164), é possível configurar a função com uma fila de mensagens mortas para salvar eventos descartados para processamento adicional. Uma fila de mensagens mortas age da mesma forma que um destino em caso de falha na medida em que é usado quando há falha em todas as tentativas de processamento de um evento ou ele expira sem ser processado. No entanto, uma fila de mensagens mortas faz parte da configuração específica da versão de uma função, portanto ela é bloqueada quando você publica uma versão. Destinos em caso de falha também oferecem suporte a destinos adicionais e incluem detalhes sobre a resposta da função no registro de invocação.

Para reprocessar eventos em uma fila de mensagens mortas, você pode defini-la como uma fonte do evento para a função do Lambda. Você também pode recuperar os eventos manualmente.

É possível escolher uma fila do Amazon SQS ou um tópico do Amazon SNS para a fila de mensagens mortas. Se você não tiver uma fila ou um tópico, crie um. Escolha o tipo de destino que corresponde ao seu caso de uso.

- **Fila do Amazon SQS:** uma fila suspende os eventos falhos até serem recuperados. Escolha uma fila do Amazon SQS se você espera que uma única entidade, como uma função do Lambda ou um alarme do CloudWatch, processe o evento em falha. Para obter mais informações, consulte [Usar AWS Lambda com o Amazon SQS \(p. 465\)](#) e invocar uma função.

Crie uma fila no [console do Amazon SQS](#).

- **Tópico do Amazon SNS:** um tópico retransmite os eventos com falha para um ou mais destinos. Escolha um tópico do Amazon SNS se você espera que várias entidades atuem em um evento com falha. Por exemplo, você pode configurar um tópico para enviar eventos a um endereço de email, uma função do Lambda e/ou um endpoint HTTP. Para obter mais informações, consulte [Usar o Lambda com o Amazon SNS \(p. 465\)](#).

Crie um tópico do SNS usando o console do [Amazon SNS](#).

Sua função precisa de permissões adicionais para enviar eventos a uma fila ou um tópico. Adicione uma política com as permissões necessárias à [função de execução \(p. 57\)](#) de sua função.

- Amazon SQS – [sns:SendMessage](#)
- Amazon SNS – [sns:Publish](#)

Se a fila ou o tópico de destino for criptografado com uma chave gerenciada pelo cliente, a função de execução também deverá ser um usuário na [política baseada em recursos](#) da chave.

Depois de criar o destino e atualizar a função de execução de sua função, adicione a fila de mensagens mortas à função. Você pode configurar várias funções para enviarem eventos ao mesmo destino.

Como configurar uma fila de mensagens mortas

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Configuration (Configuração) e, em seguida, Asynchronous invocation (Invocação assíncrona).
4. Em Asynchronous invocation (Invocação assíncrona), escolha Edit (Editar).
5. Definarecurso DLQ para Amazon SQS ou Amazon SNS.
6. Escolha a fila ou o tópico de destino.
7. Escolha Save (Salvar).

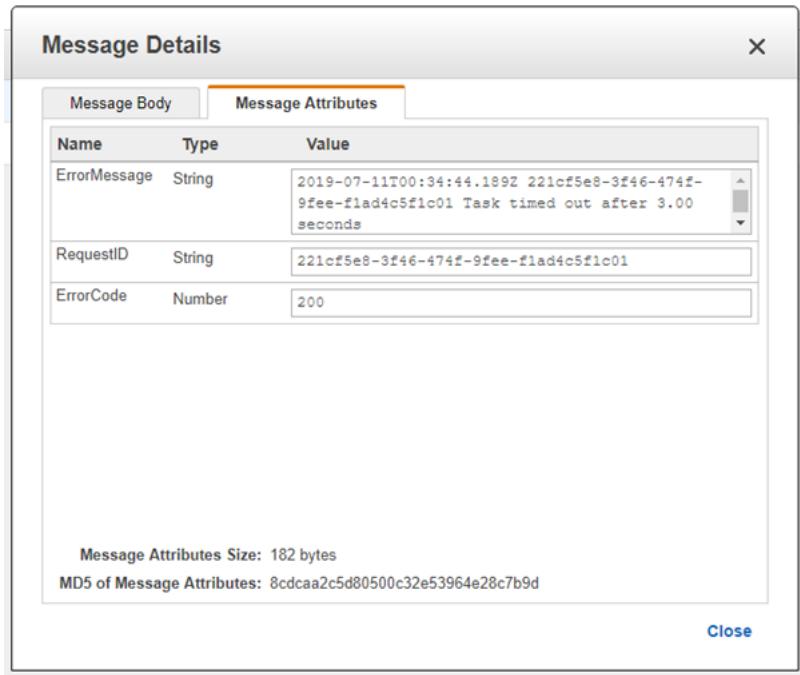
Para configurar uma fila de mensagens mortas com o AWS CLI, use o comando `update-function-configuration`.

```
aws lambda update-function-configuration --function-name my-function \
--dead-letter-config TargetArn=arn:aws:sns:us-east-2:123456789012:my-topic
```

O Lambda envia o evento para a fila de mensagens mortas no mesmo estado, mas informações adicionais nos atributos. Você pode usar essas informações para identificar o erro retornado pela função, ou correlacionar o evento com logs ou um rastreamento do AWS X-Ray.

Atributos de mensagens da fila de mensagens mortas

- RequestID (String): o ID de invocação da solicitação. Os IDs de solicitação aparecem nos logs de função. O X-Ray SDK também pode ser usado para registrar o ID de solicitação em um atributo no rastreamento. Em seguida, os rastreamentos podem ser procurados por ID de solicitação no console do X-Ray. Para ver um exemplo, consulte a [amostra de processador de erros \(p. 500\)](#).
- ErrorCode (Número): o código de status HTTP.
- ErrorMessage (String): o primeiro 1 KB da mensagem de erro.



Se o Lambda não puder enviar uma mensagem para a fila de mensagens mortas, ele excluirá o evento e emitirá a métrica [DeadLetterErrors \(p. 717\)](#). Isso pode acontecer por causa de falta de permissões, ou se o tamanho total da mensagem exceder o limite da fila ou do tópico de destino. Por exemplo, se uma notificação do Amazon SNS com um corpo próximo a 256 KB aciona uma função que resulta em erro, os dados do evento adicional incluídos pelo Amazon SNS, combinados aos atributos adicionados pelo Lambda podem fazer com que a mensagem exceda o tamanho máximo permitido na fila de mensagens mortas.

Se você está usando o Amazon SQS como uma fonte de eventos, configure uma fila de mensagens mortas na própria fila do Amazon SQS e não na função do Lambda. Para obter mais informações, consulte [O uso do AWS LambdaCom o Amazon SQS \(p. 465\)](#).

AWS Lambda Mapeamentos da fonte de eventos do

Um mapeamento da origem do evento é um recurso do AWS Lambda que lê a partir de uma origem de evento e invoca uma função do Lambda. Você pode usar os mapeamentos de origem do evento para processar itens de um stream ou fila em serviços que não invocam funções do Lambda diretamente. O Lambda fornece mapeamentos de origem de eventos para os seguintes serviços.

Serviços dos quais o Lambda lê eventos

- [Amazon DynamoDB \(p. 333\)](#)
- [Amazon Kinesis \(p. 387\)](#)
- [Amazon MQ \(p. 411\)](#)
- [Amazon Managed Streaming for Apache Kafka \(p. 419\)](#)
- [Apache Kafka autogerenciado \(p. 377\)](#)
- [Amazon Simple Queue Service \(p. 465\)](#)

Um mapeamento da origem do evento usa permissões na [função de execução \(p. 57\)](#) para ler e gerenciar itens na origem do evento. Permissões, estrutura do evento, configurações e comportamento de sondagem variam de acordo com a origem do evento. Para obter mais informações, consulte o tópico vinculado para o serviço que você usa como origem de evento.

Para gerenciar uma origem de evento com a [AWS CLI](#) ou o [AWS SDK](#), você pode usar as seguintes operações da API:

- [CreateEventSourceMapping \(p. 786\)](#)
- [ListEventSourceMappings \(p. 888\)](#)
- [GetEventSourceMapping \(p. 837\)](#)
- [UpdateEventSourceMapping \(p. 956\)](#)
- [DeleteEventSourceMapping \(p. 812\)](#)

O exemplo a seguir usa a AWS CLI para mapear uma função chamada `my-function` para uma transmissão do DynamoDB especificada pelo nome do recurso da Amazon (ARN), com um tamanho de lote de 500.

```
aws lambda create-event-source-mapping --function-name my-function --batch-size 500 --starting-position LATEST \
--event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2019-06-10T19:26:16.525
```

Você deve ver a saída a seguir:

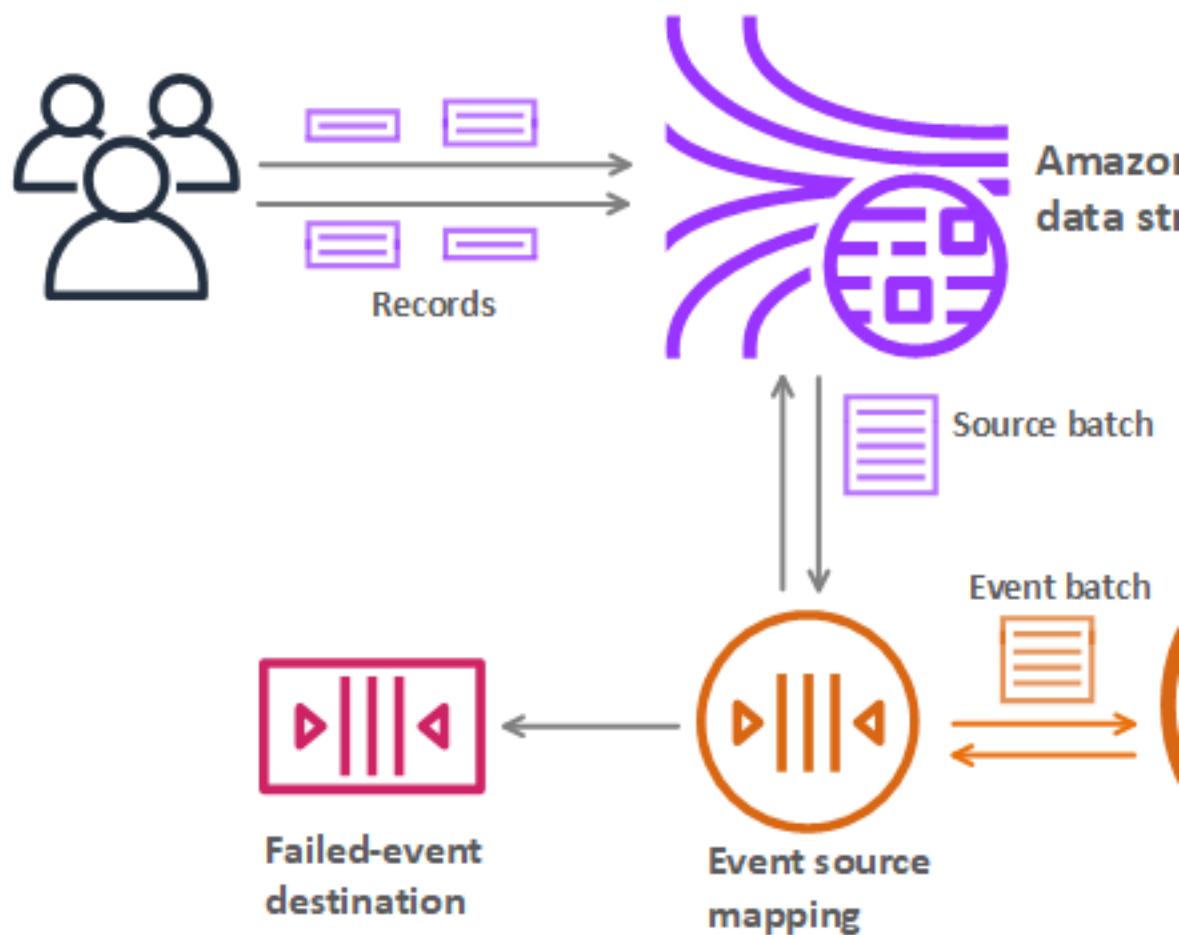
```
{  
    "UUID": "14e0db71-5d35-4eb5-b481-8945cf9d10c2",  
    "BatchSize": 500,  
    "MaximumBatchingWindowInSeconds": 0,  
    "ParallelizationFactor": 1,  
    "EventSourceArn": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2019-06-10T19:26:16.525",  
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
    "LastModified": 1560209851.963,  
    "LastProcessingResult": "No records processed",  
    "State": "Creating",
```

```
"StateTransitionReason": "User action",
"DestinationConfig": {},
"MaximumRecordAgeInSeconds": 604800,
"BisectBatchOnFunctionError": false,
"MaximumRetryAttempts": 10000
}
```

Mapeamentos de origem de evento leem itens de um fluxo ou fila em lotes. Eles incluem vários itens no evento recebido por sua função. Você pode configurar o tamanho do lote que o mapeamento da origem do evento envia para a sua função, até um valor máximo que varia de acordo com o serviço. O número de itens no evento poderá ser menor que o tamanho do lote se não houver itens suficientes disponíveis, ou se o lote for muito grande para enviar em um evento e deve ser dividido.

O exemplo a seguir mostra um mapeamento de origem de evento que lê de um fluxo do Kinesis. Se houver falha em todas as tentativas de processamento de um lote de eventos, o mapeamento de origem do evento enviará detalhes sobre o lote para uma fila do SQS.

Event Source Mapping with Kinesis Stream



O lote de eventos é o evento que o Lambda envia para a função. Ele é um lote de registros ou mensagens compiladas dos itens que o mapeamento de origem de evento lê de um fluxo ou de uma fila. O tamanho do lote e outras configurações se aplicam somente ao lote de eventos.

Para streams, um mapeamento da origem do evento cria um iterador para cada fragmento no stream e processa itens em cada fragmento na ordem. Você pode configurar o mapeamento da origem do evento para ler apenas novos itens que aparecem no stream ou para iniciar com os itens mais antigos. Os itens processados não são removidos do fluxo e podem ser processados por outras funções ou consumidores.

Por padrão, se a função retornar um erro, o lote inteiro será reprocessado até que a função seja bem-sucedida ou os itens no lote expirem. Para garantir o processamento na ordem, o processamento do fragmento afetado é pausado até o erro ser resolvido. É possível configurar o mapeamento de origem do evento para descartar eventos antigos, restringir o número de tentativas ou processar vários lotes em paralelo. Se você processar vários lotes em paralelo, o processamento em ordem ainda será garantido para cada chave de partição, mas várias chaves de partição no mesmo estilhaço serão processadas simultaneamente.

Também é possível configurar o mapeamento de origem do evento para enviar um registro de invocação para outro serviço quando descartar um lote de eventos. O Lambda suporta o seguinteDestinos (p. 164)Para mapeamentos da fonte de eventos do.

- Amazon SQS— Uma fila do SQS.
- Amazon SNS— Um tópico do SNS.

O registro de invocação contém detalhes sobre o lote de eventos com falha no formato JSON.

O exemplo a seguir mostra um registro de invocação de uma transmissão do Kinesis.

Example Registro de invocação

```
{  
    "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"  
    }  
}
```

O Lambda também oferece suporte ao processamento em ordem para [filas FIFO \(primeiro a entrar, primeiro a sair\) \(p. 465\)](#), aumentando a escala na vertical até o número de grupos de mensagens ativos. Para filas padrão, os itens não são necessariamente processados na ordem. O Lambda escala para processar uma fila padrão o mais rápido possível. Quando ocorre uma falha, os lotes são retornados para a fila como itens individuais e talvez sejam processados em um agrupamento diferente do que o lote

original. Ocasionalmente, o mapeamento da origem do evento pode receber o mesmo item da fila duas vezes, mesmo que não tenha ocorrido nenhum erro de função. O Lambda exclui itens da fila depois de serem processados com êxito. É possível configurar a fila de origem para enviar itens para uma fila de mensagens mortas se eles não puderem ser processados.

Para obter informações sobre serviços que invocam funções do Lambda diretamente, consulte [Usar o AWS Lambda com outros serviços \(p. 277\)](#).

Monitorar o estado de uma função com a API do Lambda

Quando você cria ou atualiza uma função, o Lambda provisiona os recursos de computação e rede que permitem que ela seja executada. Na maioria dos casos, esse processo é muito rápido, e sua função está pronta para ser invocada ou modificada imediatamente.

Se você configurar sua função para se conectar a uma nuvem privada virtual (VPC), o processo poderá demorar mais tempo. Quando você conecta uma função pela primeira vez a uma VPC, o Lambda provisiona interfaces de rede, o que leva cerca de um minuto. Para comunicar o estado atual da sua função, o Lambda inclui campos adicionais no documento de configuração da função que é retornado por várias ações de API do Lambda.

Quando você cria uma função, ela está inicialmente no estado `Pending`. Quando a função está pronta para ser invocada, o estado muda de `Pending` para `Active`. Enquanto o estado é `Pending`, invocações e outras ações de API que operam na função retornam um erro. Se você criar automação em torno da criação e atualização de funções, aguarde a função se tornar ativa antes de executar ações adicionais operadas na função.

É possível usar a API do Lambda para obter informações sobre o estado de uma função. Informações de estado são incluídas no documento [FunctionConfiguration \(p. 1015\)](#) retornado por várias ações de API. Para visualizar o estado da função com a AWS CLI, use o comando `get-function-configuration`.

```
aws lambda get-function-configuration --function-name my-function
```

Você deve ver a saída a seguir:

```
{  
    "FunctionName": "my-function",  
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",  
    "Runtime": "nodejs12.x",  
    "Role": "arn:aws:iam::123456789012:role/lambda-role",  
    "TracingConfig": {  
        "Mode": "Active"  
    },  
    "State": "Pending",  
    "StateReason": "The function is being created.",  
    "StateReasonCode": "Creating",  
    ...  
}
```

O `StateReason` e o `StateReasonCode` contêm informações adicionais sobre o estado quando não é `Active`. Ocorre falha nas seguintes operações enquanto a criação da função está pendente:

- [Invoke \(p. 875\)](#)
- [UpdateFunctionCode \(p. 965\)](#)
- [UpdateFunctionConfiguration \(p. 974\)](#)
- [PublishVersion \(p. 919\)](#)

Quando você atualiza a configuração de uma função, a atualização pode acionar uma operação assíncrona para provisionar recursos. Enquanto esse processo estiver em andamento, você poderá invocar a função, mas ocorrerá uma falha em outras operações na função. As invocações ocorridas enquanto a atualização está em andamento são executadas na configuração anterior. O estado da função é `Active`, mas `LastUpdateStatus` é `InProgress`.

Example Configuração de funções: conectar a uma VPC

```
{  
    "FunctionName": "my-function",  
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",  
    "Runtime": "nodejs12.x",  
    "VpcConfig": {  
        "SubnetIds": [  
            "subnet-071f712345678e7c8",  
            "subnet-07fd123456788a036",  
            "subnet-0804f77612345cacf"  
        ],  
        "SecurityGroupIds": [  
            "sg-085912345678492fb"  
        ],  
        "VpcId": "vpc-08e1234569e011e83"  
    },  
    "State": "Active",  
    "LastUpdateStatus": "InProgress",  
    ...  
}
```

Ocorre falha nas seguintes operações enquanto uma atualização assíncrona está em andamento:

- [UpdateFunctionCode \(p. 965\)](#)
- [UpdateFunctionConfiguration \(p. 974\)](#)
- [PublishVersion \(p. 919\)](#)

Outras operações, incluindo invocação, funcionam enquanto as atualizações estão em andamento.

Por exemplo, quando você conecta sua função a uma nuvem privada virtual (VPC), o Lambda provisiona uma interface de rede elástica para cada sub-rede. Esse processo pode deixar sua função em um estado pendente por um minuto ou mais. O Lambda também recupera interfaces de rede que não estão em uso, colocando sua função em um `Inactive` estado. Quando a função está inativa, uma invocação faz com que ela entre no estado `Pending` enquanto o acesso à rede é restaurado. Uma falha ocorrerá na invocação que aciona a restauração e outras invocações enquanto a operação estiver pendent `ResourceNotReadyException`.

Se o Lambda encontrar um erro ao restaurar a interface de rede de uma função, a função voltará ao estado `Inactive`. A próxima invocação pode acionar outra tentativa. Para alguns erros de configuração, o Lambda aguarda pelo menos 5 minutos antes de tentar criar outra interface de rede. Esses erros têm os seguintes valores de `LastUpdateStatusReasonCode`:

- `InsufficientRolePermission`: a função não existe ou há permissões ausentes.
- `SubnetOutOfRangeAddresses`: todos os endereços IP em uma sub-rede estão em uso.

Para obter mais informações sobre como os estados funcionam com a conectividade da VPC, consulte [Configurar uma função do Lambda para acessar recursos em uma VPC \(p. 124\)](#).

Lidar com erros e novas tentativas automáticas no AWS Lambda

Dois tipos de erro podem ocorrer quando você invoca uma função. Os erros de invocação ocorrem quando a solicitação de invocação for rejeitada antes da função recebê-la. Erros de função ocorrem quando o código ou o [tempo de execução \(p. 214\)](#) de sua função retornarem um erro. O comportamento de novas tentativas e a estratégia de gerenciamento de erros varia de acordo com o tipo de erro, o tipo de chamada e o cliente ou serviço que chama a função.

Problemas com a solicitação, o chamador e a conta podem causar erros de invocação. Os erros de invocação incluem um tipo de erro e um código de status na resposta que indicam a causa do erro.

Erros de invocação comuns

- Request (Solicitação): o evento de solicitação é muito grande ou não é um JSON válido, a função não existe ou um valor de parâmetro é do tipo errado.
- Caller (Chamador): o usuário ou o serviço não tem permissão para invocar a função.
- Account (Conta): o número máximo de instâncias de função já está em execução, ou as solicitações estão sendo feitas muito rápido.

Clientes como a AWS CLI e o AWS SDK fazem novas tentativas em casos de tempo limite atingido, erros de controle de utilização (429) e outros erros não ocasionados por solicitações inválidas. Para obter uma lista completa de erros de invocação, consulte [Invoke \(p. 875\)](#).

Erros de função ocorrem quando seu código de função ou o tempo de execução que ele usa retornarem um erro.

Erros de função comuns

- Function (Função): o código de sua função abre uma exceção ou retorna um erro.
- Runtime (Tempo de execução): o tempo de execução encerrou sua função porque detectou um erro de sintaxe, falhou em ordenar o objeto de resposta no JSON ou o limite de tempo foi atingido. A função foi encerrada com um código de erro.

Ao contrário dos erros de chamada, erros de função não fazem o Lambda retornar erros de status das séries 400 ou 500. Se a função retorna um erro, o Lambda indica isso ao incluir um cabeçalho chamado `x-Amz-Function-Error` e uma resposta em JSON com a mensagem de erro e outros detalhes. Para ver exemplos de erros de função em cada linguagem, consulte os tópicos a seguir.

- [AWS LambdaErros da função do em Node.js \(p. 529\)](#)
- [AWS LambdaErros da função do em Python \(p. 558\)](#)
- [AWS LambdaErros da função do em Ruby \(p. 583\)](#)
- [AWS LambdaErros da função do em Java \(p. 618\)](#)
- [AWS LambdaErros da função do em Go \(p. 652\)](#)
- [AWS LambdaErros da função do em C# \(p. 682\)](#)
- [AWS LambdaErros de função do no PowerShell \(p. 703\)](#)

Ao invocar um função diretamente, você determinará a estratégia para lidar com erros. Você pode tentar novamente, enviar o evento para uma fila de depuração ou ignorá-lo. O código de sua função pode ter sido executado completa ou parcialmente, ou nem ter sido executado. Se você tentar novamente, certifique-se de que o código de sua função pode lidar com o mesmo evento várias vezes sem duplicar transações nem ter efeitos colaterais indesejados.

Ao invocar um função indiretamente, você deve estar ciente do comportamento de novas tentativas do invocador e dos serviços que podem interagir com a solicitação no processo. Isso inclui as seguintes situações.

- Asynchronous Invocation (Chamada assíncrona): o Lambda faz duas novas tentativas de erros de função. Se a função não tiver capacidade suficiente para lidar com todas as solicitações em andamento, os eventos poderão ter de aguardar na fila por horas ou dias até serem enviados para a função. Você pode configurar uma fila de mensagens mortas na função para registrar eventos que não foram processadas com êxito. Para obter mais informações, consulte [Invocação assíncrona \(p. 161\)](#).
- Event source mappings (Mapeamento da fonte do evento): mapeamentos da fonte do evento que leem das transmissões repetem todo o lote de itens. Os erros repetidos bloqueiam o processamento do estilhaço afetado até o erro ser resolvido ou os itens expirarem. Para detectar estilhaços interrompidos, é possível monitorar a métrica [Iterator Age \(Idade do iterador\) \(p. 717\)](#).

Para mapeamentos da fonte do evento que leem de uma fila, você mesmo determina o intervalo de tempo entre repetições e o destino de eventos falhos. Para fazer isso, configure o tempo limite de visibilidade e a política de redirecionamento na fila de origem. Para obter mais informações, consulte [AWS Lambda Mapeamentos da fonte de eventos \(p. 170\)](#) e os tópicos específicos do serviço em [Usar o AWS Lambda com outros serviços \(p. 277\)](#).

- Serviços da AWS: os serviços da AWS podem chamar uma função [de maneira síncrona \(p. 158\)](#) ou assíncrona. Para chamada síncrona, o serviço decide se deve tentar novamente. Por exemplo, as operações em lote do Amazon S3 tentam novamente a operação se a função Lambda retornar um código de resposta `TemporaryFailure`. Os serviços que usam proxy em solicitações de um usuário ou cliente de upstream podem ter uma estratégia de repetição ou retransmitir a resposta de erro de volta para o solicitante. Por exemplo, o API Gateway sempre retransmite a resposta de erro de volta para o solicitante.

Para invocação assíncrona, o comportamento é o mesmo de quando você invoca a função de maneira síncrona. Para obter mais informações, consulte os tópicos específicos do serviço em [Usar o AWS Lambda com outros serviços \(p. 277\)](#) e a documentação do serviço de invocação.

- Other Accounts and Clients (Outras contas e clientes): ao conceder acesso a outras contas, você pode usar as [políticas baseadas em recursos \(p. 62\)](#) para restringir os serviços ou recursos que eles podem configurar para chamar sua função. Para evitar que a função fique sobrecarregada, considere colocar uma camada de API na frente da função com o [Amazon API Gateway \(p. 281\)](#).

Para ajudar você a lidar com erros em aplicações do Lambda, o Lambda se integra com serviços como o Amazon CloudWatch e o AWS X-Ray. Você pode usar um conjunto de logs, métricas, alarmes e rastreamento para detectar e identificar rapidamente os problemas no código da função, na API ou em outros recursos compatíveis com seu aplicativo. Para obter mais informações, consulte [Monitoramento e solução de problemas de aplicações do Lambda \(p. 707\)](#).

Para ver uma aplicação de exemplo que usa uma assinatura do CloudWatch Logs, rastreamento do X-Ray e uma função do Lambda para detectar e processar erros, consulte [Aplicativo de exemplo de processador de erros para o AWS Lambda \(p. 500\)](#).

Uso de extensões do Lambda

É possível usar extensões do Lambda para aumentar as funções do Lambda. Por exemplo, use extensões do Lambda para integrar funções com suas ferramentas preferidas de monitoramento, observação, segurança e governança. Você pode escolher entre várias ferramentas que os [parceiros do AWS Lambda fornecem](#), ou pode [criar suas próprias extensões do Lambda \(p. 228\)](#).

O Lambda é compatível com extensões internas e externas. Uma extensão externa é executada como um processo independente no ambiente de execução e continua a ser executada após a invocação da função ser totalmente processada. Como as extensões são executadas como processos separados, elas podem ser escritas em uma linguagem diferente da função.

Uma extensão interna é executada como parte do processo de tempo de execução. A função acessa extensões internas usando scripts wrapper ou mecanismos no processo, como `JAVA_TOOL_OPTIONS`. Para obter mais informações, consulte [Modificar o ambiente de execução \(p. 250\)](#).

É possível adicionar extensões a uma função usando o console do Lambda, a AWS Command Line Interface (AWS CLI) ou serviços e ferramentas da infraestrutura como código (IaC), como o AWS CloudFormation, o AWS Serverless Application Model (AWS SAM) e o Terraform.

Os seguintes [Tempos de execução do Lambda \(p. 214\)](#) são compatíveis com extensões:

- .NET Core 3.1 (C#/PowerShell) (`dotnetcore3.1`)
- Tempo de execução personalizado (`provided`)
- Tempo de execução personalizado no Amazon Linux 2 (`provided.al2`)
- Java 11 (Corretto) (`java11`)
- Java 8 (Corretto) (`java8.al2`)
- Node.js 14.x (`nodejs14.x`)
- Node.js 12.x (`nodejs12.x`)
- Node.js 10.x (`nodejs10.x`)
- Python 3.9 (`python3.9`)
- Python 3.8 (`python3.8`)
- Python 3.7 (`python3.7`)
- Ruby 2.7 (`ruby2.7`)
- Ruby 2.5 (`ruby2.5`)

Observe que o tempo de execução Go 1.x não suporta extensões. Para suportar extensões, você pode criar funções Go `noprovided.al2`Tempo de execução. Para obter mais informações, consulte[Migração de funções do Lambda para o Amazon Linux 2](#).

Você é cobrado pelo tempo de execução consumido pela extensão (em incrementos de 1 ms). Para obter mais informações sobre a definição de preço para extensões, consulte [Definição de preço do AWS Lambda](#). Para obter informações de definição de preço sobre extensões de parceiros, consulte os sites desses parceiros. Não há custo para instalar suas próprias extensões.

Tópicos

- [Ambiente de execução \(p. 179\)](#)
- [Impacto na performance e nos recursos \(p. 179\)](#)
- [Permissions \(p. 179\)](#)
- [Configurando extensões \(arquivamento de arquivo.zip\) \(p. 180\)](#)

- [Uso de extensões em imagens de contêiner \(p. 180\)](#)
- [Próximas etapas \(p. 181\)](#)

Ambiente de execução

O Lambda invoca a função em um [ambiente de execução \(p. 219\)](#), que fornece um ambiente do tempo de execução seguro e isolado. O ambiente de execução gerencia os recursos necessários para executar sua função e fornece suporte ao ciclo de vida para o tempo de execução e extensões da função.

O ciclo de vida do ambiente de execução inclui as seguintes fases:

- **Init:** nessa fase, o Lambda cria ou descongela um ambiente de execução com os recursos configurados, faz download do código da função e de todas as camadas, inicializa todas as extensões e o tempo de execução, em seguida executa o código de inicialização da função (o código fora do handler principal). O `Init` ocorre durante a primeira invocação, ou antes de invocações de função se você tiver habilitado [Simultaneidade provisionada \(p. 116\)](#).

O `Init` é dividido em três subfases: `Extension init`, `Runtime init`, e `Function init`. Essas subfases garantem que todas as extensões e o tempo de execução concluam suas tarefas de configuração antes que o código da função seja executado.

- **Invoke:** Nesta fase, o Lambda chama o manipulador de função. Depois que a função é executada até a conclusão, o Lambda se prepara para manipular outra invocação de função.
- **Shutdown:** Esta fase é acionada se a função Lambda não receber quaisquer invocações por um período de tempo. No `Shutdown`, o Lambda encerra o tempo de execução, alerta as extensões para permitir que elas parem de forma limpa e, em seguida, remove o ambiente. Lambda envia um `Shutdown` para cada extensão, que informa a extensão que o ambiente está prestes a ser encerrado.

Durante o `Init`, o Lambda extrai camadas contendo extensões para o /opt no ambiente de execução. O Lambda procura extensões no /opt/extensions/, interpreta cada arquivo como um bootstrap executável para iniciar a extensão e inicia todas as extensões em paralelo.

Impacto na performance e nos recursos

O tamanho das extensões da função conta para o limite de tamanho do pacote de implantação. Para um arquivo .zip, o tamanho total descompactado da função e de todas as extensões não pode exceder o limite de tamanho do pacote de implantação descompactado de 250 MB.

As extensões podem afetar a performance da função porque compartilham recursos de função, como CPU, memória e armazenamento. Por exemplo, se uma extensão executa operações de computação intensiva, você pode ver o aumento da duração da execução da função.

Cada extensão deve concluir sua inicialização antes de o Lambda invocar a função. Portanto, uma extensão que consome tempo de inicialização significativo pode aumentar a latência da invocação da função.

Para medir o tempo adicional que a extensão leva após a execução da função, você pode usar a [métrica da função \(p. 717\)](#) `PostRuntimeExtensionsDuration`. Para medir o aumento da memória usada, você pode usar a métrica `MaxMemoryUsed`. Para entender o impacto de uma extensão específica, você pode executar diferentes versões das funções lado a lado.

Permissions

As extensões têm acesso aos mesmos recursos que as funções. Como as extensões são executadas no mesmo ambiente que a função, as permissões são compartilhadas entre a função e a extensão.

Para um arquivo .zip, é possível criar um modelo do AWS CloudFormation para simplificar a tarefa de anexar a mesma configuração de extensão, incluindo permissões do AWS Identity and Access Management (IAM), a várias funções.

Configurando extensões (arquivamento de arquivo.zip)

Você pode adicionar uma extensão à sua função como uma [camada do Lambda \(p. 85\)](#). O uso de camadas permite compartilhar extensões em toda a organização ou com toda a comunidade de desenvolvedores do Lambda. É possível adicionar uma ou mais extensões a uma camada. É possível registrar até 10 extensões para uma função.

Adicione a extensão à função usando o mesmo método que faria para qualquer camada. Para obter mais informações, consulte [Usar camadas com sua função do Lambda \(p. 152\)](#).

Adicionar uma extensão à função (console)

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha a guia Código se ainda não estiver selecionada.
4. Em Camadas, escolha Editar.
5. Em Choose a layer (Selecionar uma camada), selecione Specify an ARN (Especificiar um ARN).
6. Em Specify an ARN (Especificiar um ARN), insira o nome de recurso da Amazon (ARN) de uma camada de extensão.
7. Escolha Adicionar.

Uso de extensões em imagens de contêiner

Você pode adicionar extensões à [imagem do contêiner \(p. 266\)](#). A configuração de imagem de contêiner ENTRYPPOINT especifica o processo principal para a função. Configure a configuração ENTRYPPOINT no Dockerfile ou como uma substituição na configuração da função.

Você pode executar vários processos dentro de um contêiner. Lambda gerencia o ciclo de vida do processo principal e quaisquer processos adicionais. O Lambda usa a [API Extensions do \(p. 228\)](#) para gerenciar o ciclo de vida da extensão.

Exemplo: Adicionando uma extensão externa

Uma extensão externa é executada em um processo separado da função Lambda. O Lambda inicia um processo para cada extensão no /opt/extensions/Diretório. O Lambda usa a API de extensões para gerenciar o ciclo de vida da extensão. Depois que a função for executada até a conclusão, o Lambda envia um evento Shutdown para cada extensão externa.

Example de adicionar uma extensão externa a uma imagem base Python

```
FROM public.ecr.aws/lambda/python:3.8

# 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
ADD my-extension.zip /opt
```

```
CMD python ./index.py
```

Próximas etapas

Para saber mais sobre extensões, recomendamos os seguintes recursos:

- Para obter um exemplo básico de trabalho, consulte [Building Extensions for AWS Lambda](#) no AWS Compute Blog.
- Para obter informações sobre extensões fornecidas pelos parceiros da AWS Lambda, consulte [Introducing AWS Lambda Extensions](#) no AWS Compute Blog.
- Para exibir extensões de exemplo disponíveis e scripts de wrapper, consulte [AWS Lambda Extensões](#) no repositório AWS Samples GitHub.

Invocar funções definidas como imagens de contêiner

O comportamento de uma função do Lambda definida como uma imagem de contêiner durante a invocação é muito semelhante a uma função definida como um arquivo .zip. As seções a seguir destacam as semelhanças e diferenças.

Tópicos

- [Ciclo de vida da função \(p. 182\)](#)
- [Invocar a função \(p. 182\)](#)
- [Segurança de imagem \(p. 182\)](#)

Ciclo de vida da função

O Lambda otimiza uma imagem de contêiner nova ou atualizada em preparação para que a função receba invocações. O processo de otimização pode levar alguns segundos. A função permanece no estado `Pending` até que o processo seja concluído. A função então faz a transição para o estado `Active`. Enquanto o estado for `Pending`, você poderá invocar a função, mas ocorrerá uma falha em outras operações na função. As invocações que ocorrem enquanto uma atualização de imagem está em andamento executam o código da imagem anterior.

Se uma função não é invocada por várias semanas, o Lambda recupera a versão otimizada e a função muda para o estado `Inactive`. Para reativar a função, você deve chamá-la. O Lambda rejeita a primeira invocação e a função entra no `Pending` até que o Lambda reotimize a imagem. A função então retorna ao estado `Active`.

O Lambda busca periodicamente a imagem de contêiner associada a partir do repositório do Amazon Elastic Container Registry (Amazon ECR). Se a imagem correspondente do contêiner não existir mais no Amazon ECR, a função entrará no estado `Failed` e o Lambda retornará uma falha para qualquer chamada de função.

É possível usar a API do Lambda para obter informações sobre o estado de uma função. Para obter mais informações, consulte [Monitorar o estado de uma função com a API do Lambda \(p. 174\)](#).

Invocar a função

Quando você invoca a função, o Lambda implanta a imagem do contêiner em um ambiente de execução. O Lambda inicializa qualquer [extensões \(p. 180\)](#), em seguida, executa o código de inicialização da função (o código fora do manipulador principal). Observe que a duração da inicialização da função está incluída no tempo de execução cobrado.

O Lambda então executa a função chamando o ponto de entrada de código especificado na configuração da função (as [configurações de imagem de contêiner \(p. 268\)](#) do ENTRYPOINT e do CMD).

Segurança de imagem

Quando o Lambda faz download pela primeira vez da imagem do contêiner da fonte original (Amazon ECR), a imagem do contêiner é otimizada, criptografada e armazenada usando métodos de criptografia convergentes autenticados. Todas as chaves necessárias para descriptografar os dados do cliente são protegidas usando chaves mestras do cliente (CMKs) do AWS Key Management Service (AWS KMS) gerenciado. Para acompanhar e auditar o uso de CMKs do Lambda, você pode visualizar os [logs do AWS CloudTrail \(p. 305\)](#).

Chamada de funções do Lambda com oAWS Mobile SDK for Android

É possível chamar uma função do Lambda a partir de uma aplicação para dispositivos móveis. Coloque a lógica de negócios em funções para separar seu ciclo de vida de desenvolvimento daquele de clientes front-end, tornando aplicativos para dispositivos móveis menos complexos para desenvolver e manter. Com o Mobile SDK for Android, você [Use o Amazon Cognito para autenticar usuários e autorizar solicitações \(p. 183\)](#).

Ao invocar uma função de um aplicativo para dispositivos móveis, você escolherá a estrutura do evento, o [tipo de invocação \(p. 157\)](#) e o modelo de permissões. Você pode usar [aliases \(p. 108\)](#) para habilitar atualizações contínuas para seu código de função, caso contrário, a função e o aplicativo estão totalmente vinculados. À medida que você adiciona mais funções, é possível criar uma camada de API para desacoplar o código da função de clientes frontend e melhorar a performance.

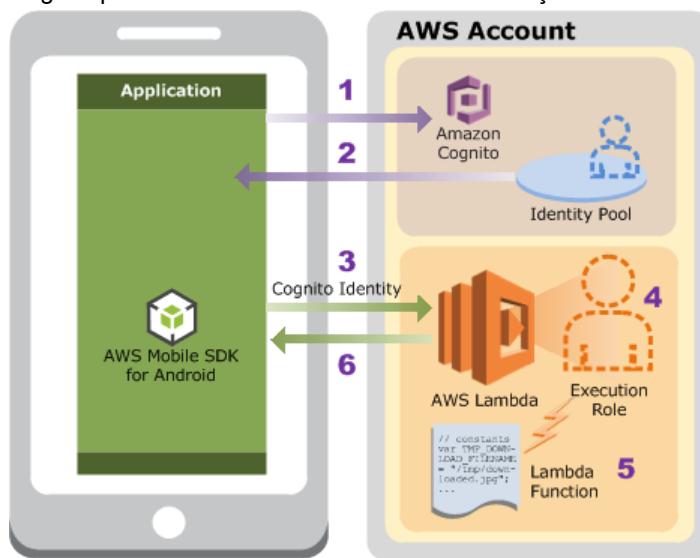
Para criar uma API da web com todos os recursos para suas aplicações Web e para dispositivos móveis, use o Amazon API Gateway. Com o API Gateway, você pode adicionar autorizadores personalizados, limitar solicitações e armazenar resultados em cache para todas as suas funções. Para obter mais informações, consulte [Usar o AWS Lambda com o Amazon API Gateway \(p. 281\)](#).

Tópicos

- [Como usar oAWS LambdaCom o Mobile SDK for Android \(p. 183\)](#)
- [Código de exemplo da função do \(p. 189\)](#)

Como usar oAWS LambdaCom o Mobile SDK for Android

Neste tutorial, você vai criar um aplicação simples para dispositivos móveis Android que usa o Amazon Cognito para obter credenciais e invoca uma função do Lambda.



A aplicação para dispositivos móveis recupera credenciais da AWS a partir de um grupo de identidades do Amazon Cognito e as utiliza para invocar uma função do Lambda com um evento que contém dados de solicitação. A função processa a solicitação e retorna uma resposta para o front-end.

Prerequisites

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em [Conceitos básicos do Lambda \(p. 9\)](#) para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, você precisa de um terminal de linha de comando ou de um shell para executar os comandos. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

Você deve ver a saída a seguir:

```
aws-cli/2.0.57 Python/3.7.4 Darwin/19.6.0 exe/x86_64
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

Criar a função de execução

Crie a [função de execução \(p. 57\)](#) que dá à sua função permissão para acessar recursos do AWS.

Para criar uma função de execução

1. Abra a [página Roles](#) (Funções) no console do IAM.
2. Selecione Create role.
3. Crie uma função com as seguintes propriedades.
 - Trusted entity (Entidade confiável – AWS Lambda).
 - Permissions (Permissões): AWSLambdaBasicExecutionRole.
 - Nome da função – **lambda-android-role**.

A política AWSLambdaBasicExecutionRole tem as permissões necessárias para a função gravar logs no CloudWatch Logs.

Criar a função

O exemplo a seguir usa dados para gerar uma resposta de string.

Note

Para o código de amostra em outras linguagens, consulte [Código de exemplo da função do \(p. 189\)](#).

Example index.js

```
exports.handler = function(event, context, callback) {
  console.log("Received event: ", event);
  var data = {
    "greetings": "Hello, " + event.firstName + " " + event.lastName + "."
  };
  callback(null, data);
```

}

Para criar a função

1. Copie o código de amostra em um arquivo chamado `index.js`.
2. Crie um pacote de implantação.

```
zip function.zip index.js
```

3. Crie uma função do Lambda com o comando `create-function`.

```
aws lambda create-function --function-name AndroidBackendLambdaFunction \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs12.x \
--role arn:aws:iam::123456789012:role/lambda-android-role
```

Testar a função do Lambda

Invoque a função manualmente usando os dados de evento de exemplo.

Para testar a função do Lambda (AWS CLI)

1. Salve o seguinte JSON de evento de exemplo em um arquivo, `input.txt`.

```
{ "firstName": "first-name", "lastName": "last-name" }
```

2. Execute o seguinte comando `invoke`:

```
aws lambda invoke --function-name AndroidBackendLambdaFunction \
--payload file://file-path/input.txt outputfile.txt
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

Criar um grupo de identidades do Amazon Cognito

Nesta seção, você cria um grupo de identidades do Amazon Cognito. O grupo de identidades tem duas funções do IAM. Você atualiza a função do IAM para usuários não autenticados e concede permissões para executar a função `AndroidBackendLambdaFunction` do Lambda.

Para obter mais informações sobre funções do IAM, consulte [Funções do IAM](#) no Manual do usuário do IAM. Para obter mais informações sobre os serviços do Amazon Cognito, consulte a [página de detalhes do produto](#).

Para criar um grupo de identidades

1. Abra o [console do Amazon Cognito](#).
2. Crie um novo grupo de identidades chamado `JavaFunctionAndroidEventHandlerPool`. Antes de seguir o procedimento para criar um grupo de identidades, observe o seguinte:
 - O grupo de identidades que você está criando deve permitir o acesso a identidades não autenticadas, pois nosso aplicativo móvel de exemplo não exige que um usuário faça login. Portanto, certifique-se de selecionar a opção `Enable access to unauthenticated identities` (`Habilitar o acesso a identidades não autenticadas`).
 - Adicione a seguinte instrução à política de permissão associada às identidades não autenticadas.

```
{  
    "Effect": "Allow",  
    "Action": [  
        "lambda:InvokeFunction"  
    ],  
    "Resource": [  
        "arn:aws:lambda:us-  
east-1:123456789012:function:AndroidBackendLambdaFunction"  
    ]  
}
```

A política resultante será a seguinte:

```
{  
    "Version":"2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "mobileanalytics:PutEvents",  
                "cognito-sync:*"  
            ],  
            "Resource": [  
                "*"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "lambda:invokefunction"  
            ],  
            "Resource": [  
                "arn:aws:lambda:us-east-1:account-  
id:function:AndroidBackendLambdaFunction"  
            ]  
        }  
    ]  
}
```

Para obter instruções sobre como criar um grupo de identidades, faça login no [Console do Amazon Cognito](#) e siga o assistente New Identity Pool (Novo grupo de identidades).

3. Observe o ID do grupo de identidades. Você especificará esse ID no aplicativo móvel que será criado na próxima seção. A aplicação usa esse ID quando envia a solicitação ao Amazon Cognito para solicitar credenciais de segurança temporárias.

Criar um aplicativo Android

Crie uma aplicação móvel simples para Android que gera eventos e invoca funções do Lambda ao passar os dados de eventos como parâmetros.

As instruções a seguir foram verificadas usando o Android Studio.

1. Crie um novo projeto Android chamado `AndroidEventGenerator` usando a seguinte configuração:
 - Selecione a plataforma Phone and Tablet.
 - Escolha Blank Activity.
2. No arquivo `build.gradle` (`Module:app`), adicione o seguinte na seção `dependencies`:

```
compile 'com.amazonaws:aws-android-sdk-core:2.2.+'  
compile 'com.amazonaws:aws-android-sdk-lambda:2.2.+'
```

3. Crie o projeto para que as dependências necessárias sejam obtidas por download, conforme necessário.
4. No manifesto do aplicativo Android (`AndroidManifest.xml`), adicione as seguintes permissões para que seu aplicativo possa se conectar à Internet. Você pode adicioná-las antes da tag `</manifest>` final.

```
<uses-permission android:name="android.permission.INTERNET" />  
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

5. Em `MainActivity`, adicione as seguintes importações:

```
import com.amazonaws.mobileconnectors.lambdainvoker.*;  
import com.amazonaws.auth.CognitoCachingCredentialsProvider;  
import com.amazonaws.regions.Regions;
```

6. Na seção `package`, adicione as seguintes duas classes (`RequestClass` e `ResponseClass`). Observe que o POJO é igual ao POJO que você criou na sua função do Lambda na seção anterior.

- `RequestClass`. As instâncias dessa classe atuam como o POJO (objeto Java antigo simples) para dados de eventos que consiste em nome e sobrenome. Se você estiver usando Java para a função do Lambda que você criou na seção anterior, este POJO é o mesmo que o POJO que você criou no código de sua função do Lambda.

```
package com.example....lambdaeventgenerator;  
public class RequestClass {  
    String firstName;  
    String lastName;  
  
    public String getFirstName() {  
        return firstName;  
    }  
  
    public void setFirstName(String firstName) {  
        this.firstName = firstName;  
    }  
  
    public String getLastName() {  
        return lastName;  
    }  
  
    public void setLastName(String lastName) {  
        this.lastName = lastName;  
    }  
  
    public RequestClass(String firstName, String lastName) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
    }  
  
    public RequestClass() {  
    }  
}
```

- `ResponseClass`

```
package com.example....lambdaeventgenerator;  
public class ResponseClass {
```

```
String greetings;

public String getGreetings() {
    return greetings;
}

public void setGreetings(String greetings) {
    this.greetings = greetings;
}

public ResponseClass(String greetings) {
    this.greetings = greetings;
}

public ResponseClass() {
}
}
```

7. No mesmo pacote, crie uma interface chamada MyInterface para invocar a função AndroidBackendLambdaFunction do Lambda.

```
package com.example....lambdaeventgenerator;
import com.amazonaws.mobileconnectors.lambdainvoker.LambdaFunction;
public interface MyInterface {

    /**
     * Invoke the Lambda function "AndroidBackendLambdaFunction".
     * The function name is the method name.
     */
    @LambdaFunction
    ResponseClass AndroidBackendLambdaFunction(RequestClass request);

}
```

A anotação `@LambdaFunction` no código mapeia o método de cliente específico à função do Lambda do mesmo nome.

8. Para manter o aplicação simples, vamos adicionar um código para invocar a função do Lambda no manipulador de eventos `onCreate()`. Em `MainActivity`, adicione o seguinte código ao final do código `onCreate()`.

```
// Create an instance of CognitoCachingCredentialsProvider
CognitoCachingCredentialsProvider cognitoProvider = new
    CognitoCachingCredentialsProvider(
        this.getApplicationContext(), "identity-pool-id", Regions.US_WEST_2);

// Create LambdaInvokerFactory, to be used to instantiate the Lambda proxy.
LambdaInvokerFactory factory = new LambdaInvokerFactory(this.getApplicationContext(),
    Regions.US_WEST_2, cognitoProvider);

// Create the Lambda proxy object with a default Json data binder.
// You can provide your own data binder by implementing
// LambdaDataBinder.
final MyInterface myInterface = factory.build(MyInterface.class);

RequestClass request = new RequestClass("John", "Doe");
// The Lambda function invocation results in a network call.
// Make sure it is not called from the main thread.
new AsyncTask<RequestClass, Void, ResponseClass>() {
    @Override
    protected ResponseClass doInBackground(RequestClass... params) {
        // invoke "echo" method. In case it fails, it will throw a
        // LambdaFunctionException.
        try {
```

```
        return myInterface.AndroidBackendLambdaFunction(params[0]);
    } catch (LambdaFunctionException lfe) {
        Log.e("Tag", "Failed to invoke echo", lfe);
        return null;
    }
}

@Override
protected void onPostExecute(ResponseClass result) {
    if (result == null) {
        return;
    }

    // Do a toast
    Toast.makeText(MainActivity.this, result.getGreetings(),
    Toast.LENGTH_LONG).show();
}
}.execute(request);
```

9. Execute o código e verifique-o como se segue:

- O `Toast.makeText()` exibe a resposta retornada.
- Verifique se o CloudWatch Logs exibe o log criado pela função do Lambda. Ele deve mostrar os dados do evento (nome e sobrenome). Você também pode verificar isso no console do AWS Lambda.

Código de exemplo da função do

O código de amostra está disponível para as seguintes linguagens.

Tópicos

- [Node.js \(p. 189\)](#)
- [Java \(p. 189\)](#)

Node.js

O exemplo a seguir usa dados para gerar uma resposta de string.

Example index.js

```
exports.handler = function(event, context, callback) {
    console.log("Received event: ", event);
    var data = {
        "greetings": "Hello, " + event.firstName + " " + event.lastName + "."
    };
    callback(null, data);
}
```

Compreenda o código do exemplo para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Node.js com arquivos .zip \(p. 517\)](#).

Java

O exemplo a seguir usa dados para gerar uma resposta de string.

No código, o `handler` (`myHandler`) usa os tipos `RequestClass` e `ResponseClass` para a entrada e saída. O código fornece implementação para esses tipos.

Example HelloPojo.java

```
package example;

import com.amazonaws.services.lambda.runtime.Context;

public class HelloPojo {

    // Define two classes/POJOs for use with Lambda function.
    public static class RequestClass {
        String firstName;
        String lastName;

        public String getFirstName() {
            return firstName;
        }

        public void setFirstName(String firstName) {
            this.firstName = firstName;
        }

        public String getLastName() {
            return lastName;
        }

        public void setLastName(String lastName) {
            this.lastName = lastName;
        }

        public RequestClass(String firstName, String lastName) {
            this.firstName = firstName;
            this.lastName = lastName;
        }

        public RequestClass() {
        }
    }

    public static class ResponseClass {
        String greetings;

        public String getGreetings() {
            return greetings;
        }

        public void setGreetings(String greetings) {
            this.greetings = greetings;
        }

        public ResponseClass(String greetings) {
            this.greetings = greetings;
        }

        public ResponseClass() {
        }
    }

    public static ResponseClass myHandler(RequestClass request, Context context){
        String greetingString = String.format("Hello %s, %s.", request.firstName,
request.lastName);
        context.getLogger().log(greetingString);
        return new ResponseClass(greetingString);
    }
}
```

```
}
```

Dependencies

- `aws-lambda-java-core`

Crie o código com as dependências da biblioteca do Lambda para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Java com arquivos .zip ou JAR \(p. 599\)](#).

AWS Lambda Aplicativos do

Uma aplicação do AWS Lambda é uma combinação de funções do Lambda, fontes de eventos e outros recursos que trabalham juntos para realizar tarefas. Você pode usar o AWS CloudFormation e outras ferramentas para coletar os componentes do aplicativo em um único pacote que pode ser implantado e gerenciado como um recurso. As aplicações tornam seus projetos do Lambda portáteis e permitem a integração com ferramentas adicionais do desenvolvedor, como o AWS CodePipeline, o AWS CodeBuild e a interface de linha de comando do AWS Serverless Application Model (SAM CLI).

O [AWS Serverless Application Repository](#) fornece uma coleção de aplicações do Lambda que você pode implantar na sua conta com alguns cliques. O repositório inclui aplicativos e amostras prontas para uso que você pode usar como ponto de partida para seus próprios projetos. Você também pode enviar seus próprios projetos para inclusão.

O [AWS CloudFormation](#) permite criar um modelo que define os recursos do aplicativo e permite gerenciar o aplicativo como uma pilha. Você pode adicionar ou modificar recursos com mais segurança na sua pilha de aplicativos. Se qualquer parte de uma atualização falhar, o AWS CloudFormation reverterá automaticamente para a configuração anterior. Com os parâmetros do AWS CloudFormation, é possível criar vários ambientes para as aplicações usando o mesmo modelo. O [AWS SAM \(p. 5\)](#) estende o AWS CloudFormation com uma sintaxe simplificada com foco no desenvolvimento de aplicações do Lambda.

O [AWS CLI \(p. 5\)](#) e a [SAM CLI \(p. 6\)](#) são ferramentas de linha de comando para gerenciar pilhas de aplicações do Lambda. Além de comandos para gerenciar pilhas de aplicativos com a API do AWS CloudFormation, o suporte do AWS CLI a comandos de nível superior que simplificam tarefas como o upload de pacotes de implantação e a atualização de modelos. A CLI do AWS SAM fornece funcionalidade adicional, incluindo a validação de modelos e testes localmente.

Ao criar um aplicativo, você pode criar seu repositório Git usando o CodeCommit ou uma conexão do AWS CodeStar com o GitHub. O CodeCommit permite que você use o console do IAM para gerenciar chaves SSH e credenciais HTTP para seus usuários. As conexões do AWS CodeStar permitem que você se conecte à sua conta do GitHub. Para obter mais informações sobre conexões, consulte [What are connections?](#) no Developer Tools console User Guide (Guia do usuário do console de ferramentas do desenvolvedor).

Para obter mais informações sobre como projetar aplicações do Lambda, consulte [Design de aplicações](#) no guia do operador do Lambda.

Tópicos

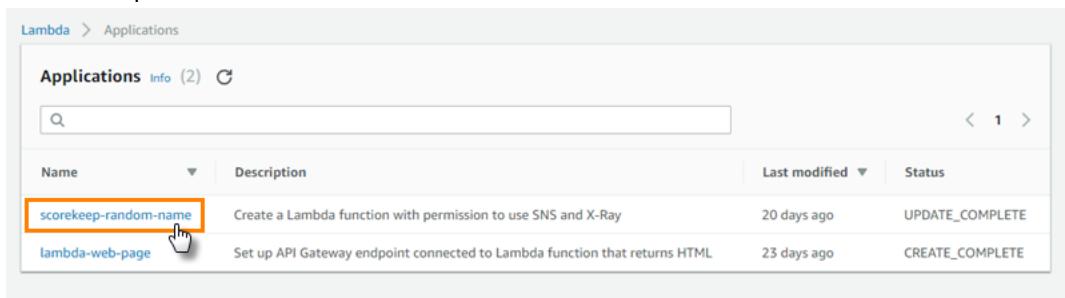
- [Gerenciar aplicativos no console do AWS Lambda \(p. 193\)](#)
- [Criar uma aplicação com entrega contínua no console do Lambda \(p. 196\)](#)
- [Implantações contínuas para funções do Lambda \(p. 206\)](#)
- [Tipos de aplicações e casos de uso comuns do Lambda \(p. 208\)](#)
- [Melhores práticas para trabalhar com funções do AWS Lambda \(p. 211\)](#)

Gerenciar aplicativos no console do AWS Lambda

O console do AWS Lambda ajuda a monitorar e gerenciar suas [aplicações do Lambda](#) (p. 192). O menu Applications (Aplicações) lista pilhas do AWS CloudFormation com funções do Lambda. O menu inclui pilhas que você inicia no AWS CloudFormation usando o console do AWS CloudFormation, o AWS Serverless Application Repository, o AWS CLI ou a CLI do AWS SAM.

Para visualizar uma aplicação do Lambda

1. Abra a página [Applications](#) (Aplicações) do console do Lambda.
2. Escolha o aplicativo.



A visão geral mostra as seguintes informações sobre o seu aplicativo.

- AWS CloudFormation template (Modelo do CloudFormation) ou SAM template (Modelo do SAM): o modelo que define sua aplicação.
- Resources (Recursos): os recursos da AWS definidos no modelo de sua aplicação. Para gerenciar as funções do Lambda de sua aplicação, escolha um nome de função na lista.

Monitorar aplicativos

Por fim, a guia Monitoring (Monitoramento) mostra um painel do Amazon CloudWatch com métricas agregadas para os recursos da aplicação.

Para monitorar uma aplicação do Lambda

1. Abra a página [Applications](#) (Aplicações) do console do Lambda.
2. Escolha Monitoring.

Por padrão, o console do Lambda mostra um painel básico. Você pode personalizar essa página definindo painéis personalizados em seu template de aplicativo. Quando seu template inclui um ou mais painéis, a página mostra seus painéis em vez do painel padrão. Você pode alternar entre painéis com o menu suspenso no canto superior direito da página.

Painéis de monitoramento personalizados

Personalize sua página de monitoramento de aplicações adicionando um ou mais painéis do Amazon CloudWatch ao seu modelo de aplicação com o tipo de recurso [AWS::CloudWatch::Dashboard](#). O exemplo a seguir cria um painel com um único widget que representa graficamente o número de invocações de uma função denominada `my-function`.

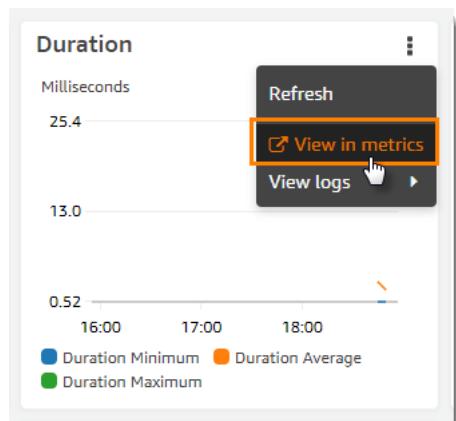
Example modelo de painel de função

```
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
              }
            ]
          ]
        }
```

Você pode obter a definição para qualquer um dos widgets no painel de monitoramento padrão no console do CloudWatch.

Para visualizar uma definição de widget

1. Abra a [página Applications](#) (Aplicações) do console do Lambda.
2. Escolha um aplicativo que tenha o painel padrão.
3. Escolha Monitoring.
4. Em qualquer widget, escolha View in metrics (Visualizar nas métricas) no menu suspenso.



5. Selecione Source.

Para obter mais informações sobre como criar painéis e widgets do CloudWatch, consulte [Estrutura e sintaxe do corpo do painel](#) na Referência de API do Amazon CloudWatch.

Criar uma aplicação com entrega contínua no console do Lambda

Você pode usar o console do Lambda para criar uma aplicação com um pipeline de entrega contínua integrado. Com a entrega contínua, cada alteração que você envia para o repositório de controle de origem aciona um pipeline que cria e implanta o aplicativo automaticamente. O console do Lambda fornece projetos iniciais para tipos de aplicações comuns com código de exemplo Node.js e modelos que criam recursos de suporte.

Neste tutorial, você vai criar os recursos a seguir.

- Aplicação: a função Node.js do Lambda, especificação de compilação e o modelo do AWS Serverless Application Model (AWS SAM).
- Pipeline: um pipeline do AWS CodePipeline que conecta os outros recursos para permitir a entrega contínua.
- Repositório: um repositório Git no AWS CodeCommit. Quando você envia uma alteração, o pipeline copia o código-fonte em um bucket do Amazon S3 e o transmite para o projeto de compilação.
- Acionador: uma regra do Amazon CloudWatch Events que observa a ramificação mestre do repositório e aciona o pipeline.
- Projeto de compilação: uma compilação do AWS CodeBuild obtém o código-fonte do pipeline e empacota a aplicação. A origem inclui uma especificação de compilação com comandos que instalam dependências e preparam um modelo do aplicativo para a implantação.
- Configuração de implantação: o estágio de implantação do pipeline define um conjunto de ações que tiram o modelo do AWS SAM processado da saída de compilação e implantam a nova versão com o AWS CloudFormation.
- BucketUm bucket do Amazon Simple Storage Service (Amazon S3) para armazenamento de artefato de implantação.
- Funções: os estágios de origem, criação e implantação do pipeline têm funções do IAM que permitem gerenciar recursos da AWS. A função do aplicativo tem uma [função de execução \(p. 57\)](#) que permite que ele carregue logs e pode ser estendida para acessar outros serviços.

Os recursos de aplicativos e pipeline são definidos em modelos do AWS CloudFormation que você pode personalizar e estender. O repositório de aplicações inclui um modelo que você pode modificar para adicionar tabelas do Amazon DynamoDB, uma API do Amazon API Gateway e outros recursos de aplicações. O pipeline de entrega contínua é definido em um modelo separado fora do controle de origem e tem sua própria pilha.

O pipeline mapeia uma única ramificação em um repositório para uma única pilha de aplicativo. Você pode criar pipelines adicionais para adicionar ambientes para outras ramificações no mesmo repositório. Você também pode adicionar estágios ao pipeline para testes, preparação e aprovações manuais. Para obter mais informações sobre o AWS CodePipeline, consulte [O que é o AWS CodePipeline?](#).

Seções

- [Prerequisites \(p. 197\)](#)
- [Criar uma aplicação \(p. 197\)](#)
- [Invocar a função do \(p. 198\)](#)
- [Adicionar um recurso da AWS \(p. 199\)](#)
- [Atualizar o limite de permissões \(p. 201\)](#)
- [Atualizar o código da função \(p. 201\)](#)
- [Próximas etapas \(p. 203\)](#)

- [Troubleshooting \(p. 203\)](#)
- [Limpar \(p. 204\)](#)

Prerequisites

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em [Conceitos básicos do Lambda \(p. 9\)](#) para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, você precisa de um terminal de linha de comando ou de um shell para executar os comandos. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

Você deve ver a saída a seguir:

```
aws-cli/2.0.57 Python/3.7.4 Darwin/19.6.0 exe/x86_64
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

Este tutorial usa CodeCommit para controle de origem. Para configurar a máquina local para acessar e atualizar o código da aplicação, consulte [Setting up \(Configurar\)](#) no Manual do usuário do AWS CodeCommit.

Criar uma aplicação

Crie uma aplicação no console do Lambda. No Lambda, uma aplicação é uma pilha do AWS CloudFormation com uma função do Lambda e qualquer número de recursos de suporte. Neste tutorial, crie um aplicativo que tenha uma função e sua função de execução.

Para criar um aplicativo.

1. Abra a [página Applications](#) (Aplicações) do console do Lambda.
2. Selecione Criar aplicativo.
3. Escolha Author from scratch (Criar do zero).
4. Defina as configurações do aplicativo.
 - Application name (Nome do aplicativo – **my-app**).
 - Runtime (Tempo de execução): Node.js 14.x.
 - Serviço de controle de origem–CodeCommit.
 - Repository name (Nome do repositório – **my-app-repo**).
 - Permissions (Permissões): Create roles and permissions boundary (Criar limite de funções e permissões).
5. Escolha Create (Criar).

O Lambda cria o pipeline e os recursos relacionados e confirma o código de aplicação de exemplo para o repositório Git. À medida que os recursos são criados, eles aparecem na página de visão geral.

Logical ID	Physical ID	Type	Last modified
helloFromLambdaFunction	b4976f9b-9399-45f1-bd21-e7a9a315981d	CodeCommit Repository	9 seconds ago

Logical ID	Physical ID	Type	Last modified
CodeCommitRepo	b4976f9b-9399-45f1-bd21-e7a9a315981d	CodeCommit Repository	9 seconds ago
PermissionsBoundaryPolicy	arn:aws:iam::123456789012:policy/my-app-us-east-2-PermissionsBoundary	IAM ManagedPolicy	13 seconds ago
S3Bucket	aws-us-east-2-123456789012-my-app-pipe	S3 Bucket	13 seconds ago

A pilha Infrastructure (Infraestrutura) contém o repositório, o projeto de compilação e outros recursos que se combinam para formar um pipeline de entrega contínua. Quando essa pilha termina a implantação, ela, por sua vez, implanta a pilha de aplicativo que contém a função e a função de execução. Estes são os recursos do aplicativo que aparecem em Resources (Recursos).

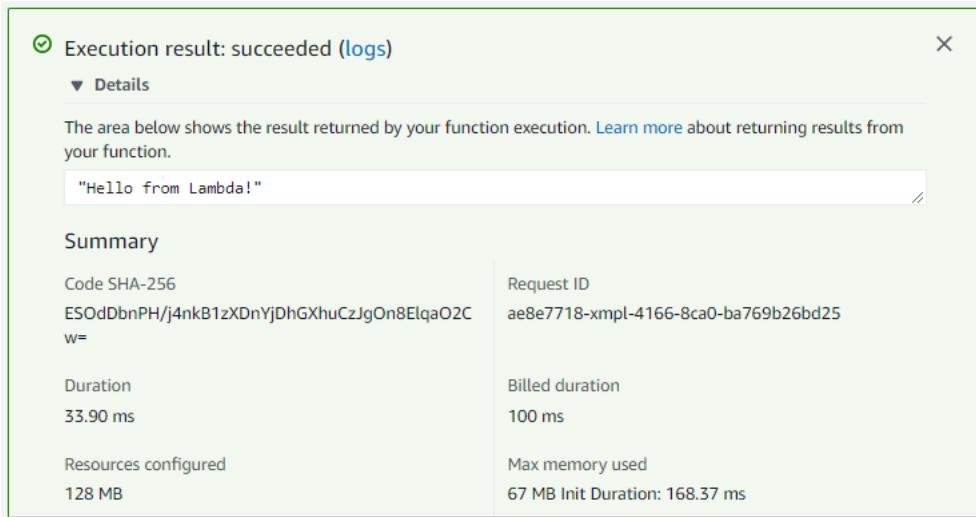
Invocar a função do

Quando o processo de implantação for concluído, invoque a função pelo console do Lambda.

Para invocar a função do aplicativo

1. Abra a página [Applications](#) (Aplicações) do console do Lambda.
2. Escolha my-app.
3. Em Resources (Recursos), escolha helloFromLambdaFunction.
4. Escolha Test (Testar).
5. Configure um evento de teste.
 - Event name (Nome do evento): **event**
 - Corpo – **{ }**
6. Escolha Create (Criar).
7. Escolha Test (Testar).

O console do Lambda executa a função e exibe o resultado. Expanda a seção Details (Detalhes) sob o resultado para ver os detalhes de saída e execução.



Adicionar um recurso da AWS

Na etapa anterior, o console do Lambda criou um repositório do Git que contém o código da função, um modelo e uma especificação de compilação. É possível adicionar recursos ao aplicativo modificando o modelo e enviando alterações ao repositório. Para obter uma cópia do aplicativo na máquina local, clone o repositório.

Para clonar o repositório do projeto

1. Abra a página [Applications](#) (Aplicações) do console do Lambda.
2. Escolha my-app.
3. Escolha Code (Código).
4. Em Repository details (Detalhes do repositório), copie o URI do repositório HTTP ou SSH, dependendo do modo de autenticação configurado durante a [configuração](#) (p. 197).
5. Para clonar o repositório, use o comando `git clone`.

```
git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/my-app-repo
```

Para adicionar uma tabela do DynamoDB à aplicação, defina um recurso `AWS::Serverless::SimpleTable` no modelo.

Para adicionar uma tabela do DynamoDB

1. Em um editor de texto, abra `template.yml`.
2. Adicione um recurso de tabela, uma variável de ambiente que passa o nome da tabela para a função e uma política de permissões que permite à função gerenciá-la.

Example `template.yml` - recursos

```
...
Resources:
  ddbTable:
    Type: AWS::Serverless::SimpleTable
    Properties:
```

```
PrimaryKey:  
  Name: id  
  Type: String  
ProvisionedThroughput:  
  ReadCapacityUnits: 1  
  WriteCapacityUnits: 1  
helloFromLambdaFunction:  
  Type: AWS::Serverless::Function  
  Properties:  
    CodeUri: ./  
    Handler: src/handlers/hello-from-lambda.helloFromLambdaHandler  
    Runtime: nodejs14.x  
    MemorySize: 128  
    Timeout: 60  
    Description: A Lambda function that returns a static string.  
  Environment:  
    Variables:  
      DDB_TABLE: !RefddbTable  
  Policies:  
    - DynamoDBCrudPolicy:  
        TableName: !RefddbTable  
    - AWSLambdaBasicExecutionRole
```

3. Confirme e envie a alteração.

```
git commit -am "Add DynamoDB table"  
git push
```

Quando você envia uma alteração, ela aciona o pipeline do aplicativo. Use a guia Deployments (Implantações) da tela do aplicativo para rastrear a alteração à medida que ela flui pelo pipeline. Quando a implantação estiver concluída, avance para a próxima etapa.

The screenshot shows the AWS Lambda Application console interface. At the top, there's a navigation bar with tabs: Overview, Code, Deployments (which is highlighted in orange), and Monitoring. Below the navigation bar, there's a section titled 'Application pipeline' which lists a single pipeline named 'my-app-Pipeline'. The pipeline status is 'InProgress' and it has a recent action 'Build: PackageExport - 8 seconds ago'. To the right of the pipeline details, there's a link 'View in CodePipeline' with a magnifying glass icon. Below the pipeline section, there's a 'SAM template' section with a 'CloudFormation stack' link and a magnifying glass icon. The main focus of the screenshot is the 'Deployment history' section, which contains a table with two rows of deployment data. The columns are 'Deployment', 'Resource type', 'Last updated time', and 'Status'. The first row shows a deployment '6 minutes ago' for a 'Lambda application' at '5 minutes ago' with a status of 'Update complete' and a green checkmark icon. The second row shows a deployment '2 hours ago' for a 'Lambda application' at '2 hours ago' with a status of 'Create complete' and a green checkmark icon. There are also buttons for 'View stack events' and navigation arrows (< 1 >) at the bottom of the deployment history table.

Deployment	Resource type	Last updated time	Status
6 minutes ago	Lambda application	5 minutes ago	✓ Update complete
2 hours ago	Lambda application	2 hours ago	✓ Create complete

Atualizar o limite de permissões

O aplicativo de exemplo aplica um limite de permissões à função de execução de sua função. O limite de permissões limita as permissões que você pode adicionar à função da função. Sem o limite, os usuários com acesso de gravação ao repositório do projeto poderiam modificar o modelo do projeto para dar à função permissão para acessar recursos e serviços fora do escopo do aplicativo de amostra.

Para que a função use a permissão do DynamoDB que você adicionou à sua função de execução na etapa anterior, você deve estender o limite para permissões adicionais. O console do Lambda detecta recursos que não estão no limite de permissões e fornece uma política atualizada que você pode usar para atualizá-lo.

Para atualizar o limite de permissões do aplicativo

1. Abra a página [Applications](#) (Aplicações) do console do Lambda.
2. Escolha o aplicativo.
3. Em Resources (Recursos), escolha Edit permissions boundary (Editar limite de permissões).
4. Siga as instruções mostradas para atualizar o limite para permitir o acesso à nova tabela.

Para obter mais informações sobre esses limites de permissões, consulte [Usar limites de permissões para aplicativos do AWS Lambda \(p. 79\)](#).

Atualizar o código da função

Atualize o código de função para usar a tabela. O código a seguir usa a tabela do DynamoDB para rastrear o número de invocações processadas por cada instância da função. Ele usa o ID de stream de log como um identificador exclusivo para a instância da função.

Para atualizar o código da função

1. Adicione um novo manipulador denominado `index.js` à pasta `src/handlers` com o conteúdo a seguir.

Example `src/handlers/index.js`

```
const dynamodb = require('aws-sdk/clients/dynamodb');
const docClient = new dynamodb.DocumentClient();

exports.handler = async (event, context) => {
    const message = 'Hello from Lambda!';
    const tableName = process.env.DDB_TABLE;
    const logStreamName = context.logStreamName;
    var params = {
        TableName : tableName,
        Key: { id : logStreamName },
        UpdateExpression: 'set invocations = if_not_exists(invocations, :start)
+ :inc',
        ExpressionAttributeValues: {
            ':start': 0,
            ':inc': 1
        },
        ReturnValues: 'ALL_NEW'
    };
    await docClient.update(params).promise();

    const response = {
```

```
        body: JSON.stringify(message)
    };
    console.log(`body: ${response.body}`);
    return response;
}
```

2. Abra o modelo de aplicativo e altere o valor do manipulador para `src/handlers/index.handler`.

Example template.yml

```
...
helloFromLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: ./
    Handler: src/handlers/index.handler
    Runtime: nodejs14.x
```

3. Confirme e envie a alteração.

```
git add . && git commit -m "Use DynamoDB table"
git push
```

Depois que a alteração de código é implantada, chame a função algumas vezes para atualizar a tabela do DynamoDB.

Para exibir a tabela do DynamoDB

1. Abra a [página Tables \(Tabelas\)](#) no console do DynamoDB.
2. Escolha a tabela que começa com my-app.
3. Escolha Items (Itens).
4. Selecione Start search (Iniciar pesquisa).

The screenshot shows the AWS DynamoDB console interface. At the top, there's a navigation bar with tabs for Overview, Items (which is selected), Metrics, Alarms, Capacity, Indexes, Global Tables, Backups, and More. Below the navigation bar, there's a toolbar with Create item, Actions, and settings icons. The main area is titled 'Scan: [Table] my-app-ddbTable-6WYEXMPL8K55'. It displays a single item with the following details:

id	invocations
2020/05/08[\$LATEST]e2fbe7b2ce58490486c4c57da0f2aa7b	3

O Lambda cria instâncias adicionais da função para lidar com várias invocações simultâneas. Cada stream de log no grupo de logs do CloudWatch Logs corresponde a uma instância de função. Uma instância de função também é criada quando você altera o código ou a configuração da função. Para obter mais informações sobre a escalabilidade, consulte [Escalabilidade da função do Lambda \(p. 32\)](#).

Próximas etapas

O modelo do AWS CloudFormation que define os recursos do aplicativo usa a transformação do AWS Serverless Application Model para simplificar a sintaxe das definições de recursos e automatizar o carregamento do pacote de implantação e de outros artefatos. O AWS SAM também fornece uma interface de linha de comando (a CLI do AWS SAM), que tem a mesma funcionalidade de empacotamento e implantação que a AWS CLI, com recursos adicionais específicos para aplicativos Lambda. Use a CLI do AWS SAM para testar a aplicação localmente em um contêiner do Docker que emula o ambiente de execução do Lambda.

- [Como instalar a CLI do AWS SAM](#)
- [Testar e depurar aplicativos sem servidor](#)

O AWS Cloud9 fornece um ambiente de desenvolvimento online que inclui Node.js, a CLI do AWS SAM e o Docker. Com o AWS Cloud9, você pode começar a desenvolver rapidamente e acessar seu ambiente de desenvolvimento a partir de qualquer computador. Para obter instruções, consulte [Getting started \(Introdução\)](#) no Manual do usuário do AWS Cloud9.

Para o desenvolvimento local, os toolkits da AWS para ambientes de desenvolvimento integrado (IDEs) permitem testar e depurar funções antes de as enviar para o repositório.

- [AWS Toolkit for JetBrains](#): plugin para IDEs PyCharm (Python) e IntelliJ (Java).
- [AWS Toolkit for Eclipse](#): plugin para IDE Eclipse (várias linguagens).
- [AWS Toolkit for Visual Studio Code](#): plugin para IDE Visual Studio Code (várias linguagens).
- [AWS Toolkit for Visual Studio](#): plugin para IDE Visual Studio (várias linguagens).

Troubleshooting

À medida que desenvolver seu aplicativo, você provavelmente encontrará os seguintes tipos de erros.

- Erros de compilação: problemas que ocorrem durante a fase de compilação, incluindo erros de compilação, teste e empacotamento.
- Erros de implantação: problemas que ocorrem quando o AWS CloudFormation não é capaz de atualizar a pilha de aplicações. Estes incluem erros de permissões, cotas de conta, problemas de serviço ou erros de modelo.
- Erros de chamada: erros que são retornados pelo código de uma função ou tempo de execução.

Para erros de compilação e implantação, você pode identificar a causa de um erro no console do Lambda.

Para solucionar erros de aplicativos

1. Abra a [página Applications](#) (Aplicações) do console do Lambda.
2. Escolha o aplicativo.
3. Escolha Deployments (Implantações).
4. Para exibir o pipeline do aplicativo, escolha Deployment pipeline (Pipeline de implantação).
5. Identifique a ação que encontrou um erro.
6. Para exibir o erro no contexto, escolha Details (Detalhes).

Para erros de implantação que ocorrem durante a ação ExecuteChangeSet, o pipeline vincula uma lista de eventos de pilha no console do AWS CloudFormation. Procure um evento com o status UPDATE_FAILED. Como o AWS CloudFormation reverte após um erro, o evento relevante está em vários outros eventos na

lista. Se o AWS CloudFormation não puder criar um conjunto de alterações, o erro será exibido em Change sets (Conjuntos de alterações) em vez de em Events (Eventos).

Uma causa comum de erros de implantação e invocação é a falta de permissões em uma ou mais funções. O pipeline tem uma função para implantações (`CloudFormationRole`) equivalente às [permissões de usuário \(p. 67\)](#) que você usaria para atualizar uma pilha do AWS CloudFormation diretamente. Se você adicionar recursos à aplicação ou habilitar recursos do Lambda que exigem permissões de usuário, a função de implantação será usada. Você pode encontrar um link para a função de implantação em Infrastructure (Infraestrutura) na visão geral do aplicativo.

Se a função acessar outros produtos ou recursos da AWS, ou se você habilitar recursos que exigem que a função tenha permissões adicionais, a [função de execução \(p. 57\)](#) dessa função será usada. Todas as funções de execução criadas no modelo de aplicativo também estão sujeitas ao limite de permissões do aplicativo. Esse limite exige que você conceda explicitamente acesso a serviços e recursos adicionais no IAM depois de adicionar permissões à função de execução no modelo.

Por exemplo, para [conectar uma função a uma nuvem privada virtual \(p. 124\)](#) (VPC), você precisa de permissões de usuário para descrever os recursos da VPC. A função de execução precisa de permissão para gerenciar interfaces de rede. Isso requer as etapas a seguir.

1. Adicione as permissões de usuário necessárias à função de implantação no IAM.
2. Adicione as permissões de função de execução ao limite de permissões no IAM.
3. Adicione as permissões de função de execução à função de execução no modelo de aplicativo.
4. Confirme e envie para implantar a função de execução atualizada.

Depois de resolver erros de permissões, escolha Release change (Liberar alteração) na visão geral do pipeline para executar novamente a compilação e a implantação.

Limpar

Você pode continuar a modificar e usar a amostra para desenvolver seu próprio aplicativo. Se você tiver terminado de usar a amostra, exclua o aplicativo para evitar pagar pelo pipeline, repositório e armazenamento.

Para excluir o aplicativo

1. Abra o [console do AWS CloudFormation](#).
2. Exclua a pilha de aplicações – my-app.
3. Abra o [console do Amazon S3](#).
4. Exclua o bucket de artefato – **us-east-2-123456789012-my-app-pipe**.
5. Retorne ao console do AWS CloudFormation e exclua a pilha de infraestrutura – serverlessrepo-my-app-toolchain.

Os logs de função não estão associados à pilha de aplicativos ou de infraestrutura no AWS CloudFormation. Exclua o grupo de logs separadamente no console do CloudWatch Logs.

Para excluir o grupo de logs

1. Abra a [página Log groups](#) (Grupos de log) do console do Amazon CloudWatch.
2. Escolha o grupo de logs da função (`/aws/lambda/my-app-helloFromLambdaFunction-YV1VXMPK7QK`).
3. Escolha Actions (Ações) e selecione Delete log group (Excluir grupo de log).
4. Selecione Sim, excluir.

Implantações contínuas para funções do Lambda

Use implantações contínuas para controlar os riscos associados à introdução de novas versões da sua função do Lambda. Em uma implantação contínua, o sistema implanta automaticamente a nova versão da função e envia gradualmente uma quantidade crescente de tráfego para a nova versão. A quantidade de tráfego e a taxa de aumento são parâmetros que você pode configurar.

Configurar uma implementação sem interrupção utilizando AWS CodeDeploy e AWS SAM. CodeDeploy é um serviço que automatiza implantações de aplicativos para plataformas de computação da Amazon, como o Amazon EC2 e AWS Lambda. Para obter mais informações, consulte [O que é o CodeDeploy?](#). Usando o CodeDeploy para implantar sua função do Lambda, é possível monitorar com facilidade o status da implantação e iniciar uma reversão se detectar algum problema.

O AWS SAM é um framework de código aberto para a criação de aplicações sem servidor. Crie um modelo de AWS SAM (no formato YAML) para especificar a configuração dos componentes necessários para a implantação contínua. O AWS SAM usa o modelo para criar e configurar os componentes. Para obter mais informações, consulte [O que é o AWS SAM?](#).

Em uma implantação contínua, o AWS SAM executa estas tarefas:

- Configura sua função do Lambda e cria um alias.
A configuração de roteamento de alias é o recurso subjacente que implementa a implantação contínua.
- Cria um aplicativo CodeDeploy e um grupo de implantação.
O grupo de implantação gerencia a implantação contínua e a reversão (se necessário).
- Detecta quando você cria uma nova versão da sua função do Lambda.
- Aciona o CodeDeploy para iniciar a implantação da nova versão.

Exemplo AWS SAM Modelo Lambda

O exemplo a seguir mostra um [Modelo do AWS SAM](#) para uma implantação contínua simples.

```
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: nodejs12.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
```

Este modelo define uma função do Lambda denominada `myDateTimeFunction` com as propriedades a seguir.

AutoPublishAlias

A propriedade `AutoPublishAlias` cria um alias chamado `live`. Além disso, o framework AWS SAM detecta automaticamente quando você salva um código novo para a função. O framework publica uma nova versão da função e atualiza o alias do `live` para apontar para a nova versão.

DeploymentPreference

A propriedade `DeploymentPreference` determina a taxa na qual a aplicação do CodeDeploy muda o tráfego da versão original da função do Lambda para a nova versão. O valor `Linear10PercentEvery2Minutes` muda dez por cento adicionais do tráfego para a nova versão a cada dois minutos.

Para obter uma lista das configurações de implantação predefinidas, consulte [Configurações de implantação](#).

Para obter um tutorial detalhado sobre como usar o CodeDeploy com funções do Lambda, consulte [Implantar uma função do Lambda atualizada com o CodeDeploy](#).

Tipos de aplicações e casos de uso comuns do Lambda

Acionadores e funções do Lambda são os componentes principais ao criar aplicações no AWS Lambda. Uma função do Lambda é o código e tempo de execução que processam eventos, e um acionador é o serviço ou aplicação da AWS que invoca a função. Para ilustrar, considere os seguintes cenários:

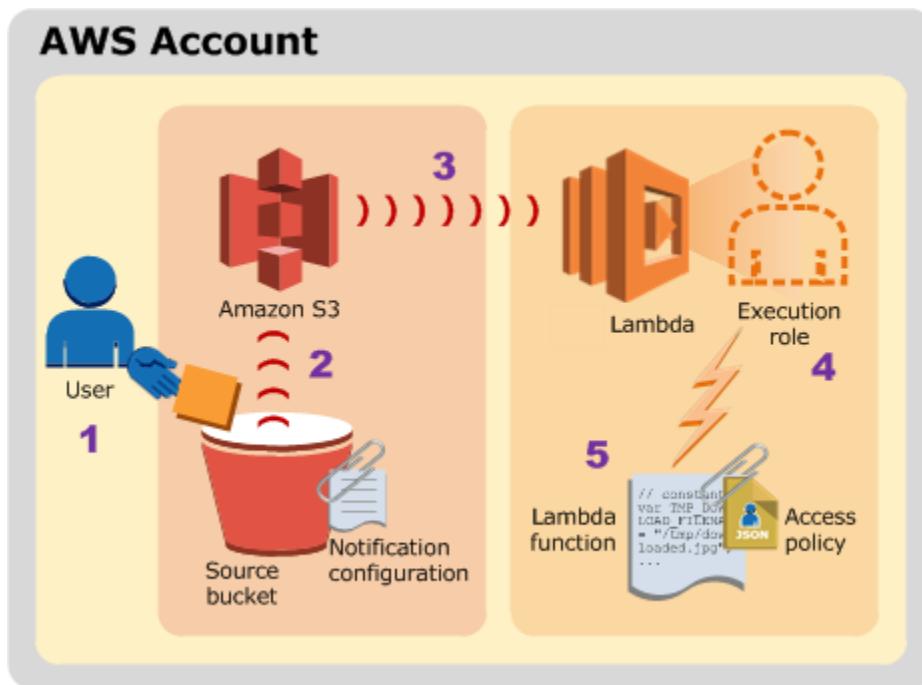
- Processamento de arquivos: suponha que você tenha um aplicação de compartilhamento de fotos. As pessoas usam o seu aplicação para fazer upload de fotos e o aplicação armazena essas fotos dos usuários no bucket do Amazon S3. Em seguida, o aplicativo cria uma versão em miniatura de cada foto e as exibe na página de perfil do usuário. Neste cenário, você pode optar por criar uma função do Lambda que cria uma miniatura automaticamente. O Amazon S3 é uma das fontes de eventos da AWS compatíveis que pode publicar eventos criados por objetos e invocar a função do Lambda. Seu código de função do Lambda pode ler o objeto da foto do bucket do S3, criar uma versão em miniatura e salvá-la em outro bucket do S3.
- Dados e análise: suponha que você está criando um aplicação de análise e armazenando dados brutos em uma tabela do DynamoDB. Quando você gravar, atualizar ou excluir itens em uma tabela, os DynamoDB Streams poderão publicar eventos de atualização do item para um fluxo associado à tabela. Neste caso, os dados de eventos fornecem a chave de item, o nome do evento (como, inserir, atualizar e excluir) e outros detalhes relevantes. Você pode escrever uma função do Lambda para gerar métricas personalizadas, agregando dados brutos.
- Sites: suponha que você esteja criando um site e deseja hospedar a lógica de backend no Lambda. Você pode invocar sua função do Lambda pelo HTTP usando o Amazon API Gateway como o endpoint HTTP. Agora, o cliente Web pode invocar a API e o API Gateway pode encaminhar a solicitação para o Lambda.
- Aplicações móveis: suponha que você tem uma aplicação móvel personalizada que produz eventos. Você pode criar uma função do Lambda para processar eventos publicados pela sua aplicação personalizada. Por exemplo, você pode configurar uma função do Lambda para processar os cliques na aplicação móvel personalizada.

O AWS Lambda oferece suporte a muitos produtos da AWS, como fontes de eventos. Para obter mais informações, consulte [Usar o AWS Lambda com outros serviços \(p. 277\)](#). Quando você configura essas fontes de eventos para acionar uma função do Lambda, essa função do Lambda é invocada automaticamente quando ocorrem eventos. Você define o mapeamento de fontes de eventos, que é a maneira como você identifica quais eventos são rastreados e qual função do Lambda é invocada.

Veja a seguir exemplos de introdução de fontes de eventos e como a experiência de ponta a ponta funciona.

Exemplo 1: o Amazon S3 envia eventos por push e invoca uma função do Lambda

O Amazon S3 pode publicar eventos de diferentes tipos, como eventos de objeto PUT, POST, COPY e DELETE, em um bucket. Usando o recurso de notificação do bucket, você pode configurar um mapeamento de origem de evento que direciona o Amazon S3 para invocar uma função do Lambda quando um tipo específico de evento ocorre, como mostrado na ilustração a seguir.



O diagrama ilustra a seguinte sequência:

1. O usuário cria um objeto em um bucket.
2. O Amazon S3 detecta o evento criado por objeto.
3. O Amazon S3 invoca sua função do Lambda usando as permissões fornecidas pela [função de execução \(p. 57\)](#).
4. O AWS Lambda executa a função do Lambda especificando o evento como um parâmetro.

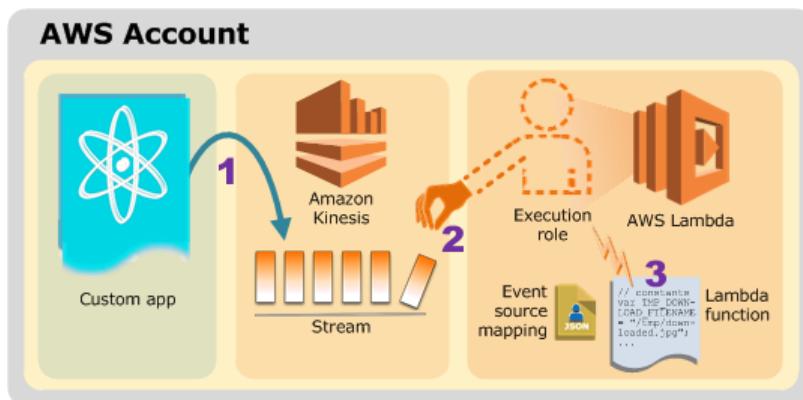
Você configura o Amazon S3 para invocar sua função como uma ação de notificação do bucket. Para conceder ao Amazon S3 permissão para invocar a função, atualize a [política baseada em recursos \(p. 62\)](#) da função.

Exemplo 2: o AWS Lambda extrai eventos de uma transmissão do Kinesis e invoca uma função do Lambda

Para fontes de eventos baseadas em sondagem, o AWS Lambda sonda a fonte e, em seguida, invoca a função do Lambda quando são detectados registros naquela fonte.

- [CreateEventSourceMapping \(p. 786\)](#)
- [UpdateEventSourceMapping \(p. 956\)](#)

O diagrama a seguir mostra como um aplicação personalizado grava os registros em uma transmissão do Kinesis.



O diagrama ilustra a seguinte sequência:

1. A aplicação personalizada grava os registros em uma transmissão do Kinesis.
2. O AWS Lambda sonda continuamente a transmissão e invoca a função do Lambda no momento em que o serviço detecta novos registros. O AWS Lambda sabe qual transmissão pesquisar e qual função do Lambda invocar com base no mapeamento de fontes de eventos criado no Lambda.
3. A função do Lambda é invocada com o evento de entrada.

Ao trabalhar com fontes de eventos com base em fluxo, você cria mapeamentos da fonte do evento no AWS Lambda. O Lambda lê itens da transmissão e invoca a função de forma síncrona. Você não precisa conceder ao Lambda permissão para invocar a função, mas ele precisa de permissão para ler a transmissão.

Melhores práticas para trabalhar com funções do AWS Lambda

As melhores práticas a seguir são recomendadas para usar o AWS Lambda:

Tópicos

- [Código da função \(p. 211\)](#)
- [Configuração da função \(p. 212\)](#)
- [Métricas e alarmes \(p. 213\)](#)
- [Trabalhar com streams \(p. 213\)](#)

Para obter mais informações sobre práticas recomendadas para aplicações do Lambda, consulte [Design de aplicações](#) no Guia do operador do Lambda.

Código da função

- Separe o manipulador do Lambda da lógica central. Isso permite que você crie uma função mais fácil para teste de unidade. No Node.js isso pode ser semelhante a:

```
exports.myHandler = function(event, context, callback) {
  var foo = event.foo;
  var bar = event.bar;
  var result = MyLambdaFunction (foo, bar);

  callback(null, result);
}

function MyLambdaFunction (foo, bar) {
  // MyLambdaFunction logic here
}
```

- Aproveite a reutilização do ambiente de execução para melhorar a performance da função. Inicialize clientes SDK e conexões de banco de dados fora do manipulador de funções e armazene em cache os ativos estáticos localmente no diretório /tmp. As invocações subsequentes processadas pela mesma instância da função podem reutilizar esses recursos. Isso economiza custos reduzindo o tempo de execução da função.

Para evitar possíveis vazamentos de dados entre invocações, não use o ambiente de execução para armazenar dados do usuário, eventos ou outras informações com implicações de segurança. Se sua função depende de um estado mutável que não pode ser armazenado na memória dentro do manipulador, considere criar uma função separada ou versões separadas de uma função para cada usuário.

- Use uma diretiva keep-alive para manter conexões persistentes. O Lambda limpa conexões ociosas ao longo do tempo. A tentativa de reutilizar uma conexão ociosa ao invocar uma função resultará em um erro de conexão. Para manter sua conexão persistente, use a diretiva keep-alive associada ao tempo de execução. Para obter um exemplo, consulte [Reutilizar conexões com keep-alive em Node.js](#).
- Use [variáveis de ambiente \(p. 99\)](#) para passar parâmetros operacionais para sua função. Por exemplo, se estiver gravando em um bucket do Amazon S3, em vez fixar no código o nome do bucket em que você está gravando, configure o nome do bucket como uma variável de ambiente.
- Controle as dependências em seu pacote de implantação da função. O ambiente de execução do AWS Lambda contém várias bibliotecas, como o AWS SDK for Node.js e os tempos de execução do Python (é possível encontrar uma lista completa: [Tempos de execução do Lambda \(p. 214\)](#)). Para habilitar o conjunto de recursos e atualizações de segurança mais recente, o Lambda atualizará periodicamente

essas bibliotecas. Essas atualizações podem introduzir alterações sutis ao comportamento de sua função do Lambda. Para ter controle total das dependências usadas por sua função, empacote todas as dependências em seu pacote de implantação.

- Minimize o tamanho do pacote de implantação para as necessidades de seu tempo de execução. Isso reduzirá a quantidade de tempo necessário para que seu pacote de implantação seja obtido por download e desempacotado antes da invocação. Para funções criadas em Java ou .NET Core, evite fazer upload da biblioteca inteira do AWS SDK como parte de seu pacote de implantação. Em vez disso, dependa seletivamente dos módulos que coletam os componentes do SDK necessários (por exemplo, DynamoDB, módulos do SDK do Amazon S3 e [bibliotecas principais do Lambda](#)).
- Reduza o tempo necessário para o Lambda desempacotar pacotes de implantação criados em Java colocando seus arquivos `.jar` de dependências em um diretório `lib/separado`. Isso é mais rápido do que colocar todo o código de sua função em um único jar com um grande número de arquivos `.class`. Para obter instruções, consulte [Implantar funções do Lambda em Java com arquivos .zip ou JAR \(p. 599\)](#).
- Minimize a complexidade de suas dependências. Prefira frameworks mais simples que sejam carregados rapidamente na inicialização do [ambiente de execução \(p. 219\)](#). Por exemplo, prefira frameworks de injeção de dependência Java (IoC) mais simples, como [Dagger](#) ou [Guice](#), em vez de frameworks mais complexos, como o [Spring Framework](#).
- Evite usar código recursivo em sua função do Lambda, em que a função chame a si mesma automaticamente até que alguns critérios arbitrários sejam atendidos. Isso pode levar a um volume não intencional de invocações da função e a custos elevados. Se você fizer isso acidentalmente, defina a simultaneidade reservada da função como o imediatamente para limitar todas as invocações da função, enquanto você atualiza o código.

Configuração da função

- Os testes de performance de sua função Lambda são uma parte essencial para garantir que você escolha a configuração do tamanho da memória ideal. Qualquer aumento no tamanho da memória dispara um aumento equivalente na CPU disponível para sua função. O uso de memória para sua função é determinado por invocação e pode ser visualizado no [Amazon CloudWatch](#). Em cada invocação, será feita uma entrada de `REPORT:`, conforme mostrado a seguir:

```
REPORT RequestId: 3604209a-e9a3-11e6-939a-754dd98c7be3 Duration: 12.34 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 18 MB
```

Ao analisar o campo `Max Memory Used:`, você poderá determinar se a função precisa de mais memória ou se você provisionou excessivamente o tamanho da memória de sua função.

Para encontrar a configuração de memória correta para suas funções, recomendamos usar o projeto de código aberto AWS Lambda Power Tuning. Para obter mais informações, consulte [AWS Lambda Power Tuning](#) no GitHub.

Para otimizar a performance da função, também recomendamos a implantação de bibliotecas que possam aproveitar as [Advanced Vector Extensions 2 \(AVX2\)](#). Isso permite processar cargas de trabalho exigentes, incluindo inferência de machine learning, processamento de mídia, High Performance Computing (HPC – Computação de alta performance, simulações científicas e modelagem financeira. Para obter mais informações, consulte [Creating faster AWS Lambda functions with AVX2](#).

- Teste a carga de sua função do Lambda para determinar um valor de tempo limite ideal. É importante analisar por quanto tempo sua função é executada para que você possa determinar melhor qualquer problema com um serviço de dependência que possa aumentar a simultaneidade da função além do que você espera. Isso é importante, principalmente quando sua função do Lambda faz chamadas de rede para recursos que não podem lidar com a escalabilidade do Lambda.
- Use permissões mais restritivas ao definir políticas do IAM. Compreenda os recursos e operações necessários para sua função do Lambda e limite a função de execução a essas permissões. Para obter mais informações, consulte [AWS Lambda Permissões \(p. 56\)](#).

- Familiarize-se com o [Cotas Lambda \(p. 53\)](#). O tamanho da carga, os descritores de arquivos e o espaço /tmp geralmente são ignorados ao determinar os limites de recursos de tempo de execução.
- Exclua as funções do Lambda que você não está mais usando. Fazendo isso, as funções não utilizadas não serão contadas desnecessariamente em relação ao limite do tamanho do pacote de implantação.
- Se você estiver usando o Amazon Simple Queue Service como uma fonte de eventos, certifique-se de que o valor do tempo de invocação esperado da função não exceda o valor do [Tempo limite de visibilidade](#) na fila. Isso se aplica a [CreateFunction \(p. 796\)](#) e [UpdateFunctionConfiguration \(p. 974\)](#).
 - No caso de CreateFunction, o AWS Lambda fará com que haja falha no processo de criação da função.
 - No caso de UpdateFunctionConfiguration, isso pode resultar em invocações duplicadas da função.

Métricas e alarmes

- Use [Trabalhar com métricas de função do AWS Lambda \(p. 717\)](#) e [Alarmes do CloudWatch](#), em vez de criar ou atualizar uma métrica no código da função do Lambda. É uma forma muito mais eficiente de monitorar a integridade de suas funções do Lambda permitindo detectar problemas no início do processo de desenvolvimento. Por exemplo, é possível configurar um alarme com base na duração esperada da invocação da função do Lambda para lidar com gargalos ou latências atribuíveis ao código da função.
- Utilize sua biblioteca de logs e as [Métricas e dimensões do AWS Lambda](#) para detectar erros de aplicativo (por exemplo, ERR, ERROR, WARNING etc.)

Trabalhar com streams

- Teste com diferentes tamanhos de lotes e de registros para que a frequência de sondagem de cada origem de evento seja ajustada para a rapidez com que a função pode concluir sua tarefa. [BatchSize \(p. 787\)](#) controla o número máximo de registros que podem ser enviados para sua função a cada invocação. Um tamanho de lote maior geralmente absorve de maneira mais eficiente a sobrecarga da invocação em um conjunto maior de registros aumentando a taxa de transferência.

Por padrão, o Lambda invoca sua função assim que os registros estão disponíveis no fluxo. Se o lote que o Lambda lê da transmissão tiver apenas um registro nele, o Lambda envia apenas um registro para a função. Para evitar invocar a função com um número pequeno de registros, você pode instruir à origem dos eventos para fazer o buffer dos registros por até cinco minutos, configurando uma janela de lote. Antes de invocar a função, o Lambda continua a ler registros do stream até coletar um lote inteiro, ou até que a janela de lote expire.

- Aumente a taxa de transferência de processamento de fluxos do Kinesis adicionando fragmentos. Uma transmissão do Kinesis é composto de um ou mais fragmentos. O Lambda sonará cada fragmento com no máximo uma invocação simultânea. Por exemplo, se o seu fluxo tiver 100 estilhaços ativos, haverá, no máximo, 100 invocações de função do Lambda em execução simultaneamente. O aumento do número de estilhaços aumentará diretamente o número máximo de invocações simultâneas de função do Lambda e poderá aumentar a taxa de transferência para processamento de transmissões do Kinesis. Se você estiver aumentando o número de fragmentos em uma transmissão do Kinesis, certifique-se de ter escolhido uma boa chave de partição (consulte [Chaves de partição](#)) para seus dados, para que os registros relacionados terminem nos mesmos fragmentos e seus dados sejam bem distribuídos.
- Use o [Amazon CloudWatch](#) no IteratorAge para determinar se a transmissão do Kinesis está sendo processada. Por exemplo, configure um alarme do CloudWatch com uma configuração máxima de 30000 (30 segundos).

Tempos de execução do Lambda

O Lambda oferece suporte a vários idiomas por meio do uso de [tempos de execução \(p. 19\)](#). Em uma [função definida como uma imagem de contêiner \(p. 90\)](#), você escolhe um tempo de execução e a distribuição Linux ao [criar a imagem de contêiner \(p. 267\)](#). Para alterar o tempo de execução, crie uma nova imagem de contêiner.

Quando você usa um arquivo .zip para o pacote de implantação, escolhe um tempo de execução ao criar a função. Para alterar o tempo de execução, você pode [atualizar a configuração da sua função \(p. 82\)](#). O tempo de execução é emparelhado com uma das distribuições do Amazon Linux. O ambiente de execução subjacente fornece bibliotecas e [variáveis de ambiente \(p. 99\)](#) adicionais que podem ser acessadas do código de sua função.

Amazon Linux

- Imagem: [amzn-ami-hvm-2018.03.0.20181129-x86_64-gp2](#)
- Kernel do Linux: 4.14.171-105.231.amzn1.x86_64

Amazon Linux 2

- Imagem: personalizada
- Kernel do Linux: 4.14.165-102.205.amzn2.x86_64

O Lambda invoca sua função em um [ambiente de execução \(p. 219\)](#). Um ambiente de execução fornece um ambiente de tempo de execução isolado e seguro, que gerencia os recursos necessários para executar a função. O Lambda reutiliza o ambiente de execução de uma invocação anterior, caso haja alguma disponível, ou pode criar um novo ambiente de execução.

Um tempo de execução pode oferecer suporte a uma única versão de um idioma, várias versões de um idioma ou vários idiomas. Tempos de execução específicos para uma versão de linguagem ou framework são [defasados \(p. 217\)](#) quando a versão chega ao fim da vida útil.

Tempos de execução Node.js

Nome	Identifier	SDK for JavaScript	Sistema operacional
Node.js 14	<code>nodejs14.x</code>	2.952.0	Amazon Linux 2
Node.js 12	<code>nodejs12.x</code>	2.952.0	Amazon Linux 2
Node.js 10	<code>nodejs10.x</code>	2.952.0	Amazon Linux 2

Note

Para obter informações sobre o fim do suporte a respeito do Node.js 10, consulte [the section called “Política de suporte ao tempo de execução” \(p. 217\)](#).

Tempos de execução do Python

Nome	Identifier	AWS SDK for Python	Sistema operacional	
Python 3.9	<code>python3.9</code>	boto3-1.17.100 botocore-1.20.100	Amazon Linux 2	

Nome	Identifier	AWS SDK for Python	Sistema operacional	
Python 3.8	<code>python3.8</code>	boto3-1.17.100 botocore-1.20.100	Amazon Linux 2	
Python 3.7	<code>python3.7</code>	boto3-1.17.100 botocore-1.20.100	Amazon Linux	
Python 3.6	<code>python3.6</code>	boto3-1.17.100 botocore-1.20.100	Amazon Linux	
Python 2.7	<code>python2.7</code>	boto3-1.17.100 botocore-1.20.100	Amazon Linux	

Important

O Python 2.7 chegou ao fim da vida útil em 1º de janeiro de 2020. O fim do suporte (fase 1) para o tempo de execução do Python 2.7 começou em 15 de julho de 2021. Para obter mais informações, consulte [Anunciando o fim do suporte para Python 2.7 no AWS Lambda](#) no AWS Computação do blog.

Tempos de execução do Ruby

Nome	Identifier	SDK for Ruby	Sistema operacional	
Ruby 2.7	<code>ruby2.7</code>	3.0.1	Amazon Linux 2	
Ruby 2.5	<code>ruby2.5</code>	3.0.1	Amazon Linux	

Note

Para obter informações sobre o fim do suporte sobre o Ruby 2.5, consulte [the section called “Política de suporte ao tempo de execução” \(p. 217\)](#).

Tempos de execução do Java

Nome	Identifier	JDK	Sistema operacional
Java 11	<code>java11</code>	amazon-corretto-11	Amazon Linux 2
Java 8	<code>java8.al2</code>	amazon-corretto-8	Amazon Linux 2
Java 8	<code>java8</code>	java-1.8.0-openjdk	Amazon Linux

Tempos de execução do Go

Nome	Identifier	Sistema operacional
Go 1.x	<code>go1.x</code>	Amazon Linux

Tempos de execução do .NET

Nome	Identifier	Sistema operacional	
.NET Core 3.1	dotnetcore3.1	Amazon Linux 2	
.NET Core 2.1	dotnetcore2.1	Amazon Linux	

Note

Para obter informações sobre o fim do suporte para o .NET Core 2.1, consulte [the section called "Política de suporte ao tempo de execução" \(p. 217\)](#).

Para usar outras linguagens no Lambda, você pode implementar um [tempo de execução personalizado](#) (p. 255). O ambiente de execução do Lambda fornece uma [interface de tempo de execução](#) (p. 224) para obter eventos de invocação e enviar respostas. Você pode implantar um tempo de execução personalizado junto com o código da função, ou em uma [camada](#) (p. 85).

Tempo de execução personalizado

Nome	Identifier	Sistema operacional
Tempo de execução personalizado	provided.al2	Amazon Linux 2
Tempo de execução personalizado	provided	Amazon Linux

Tópicos

- [Política de suporte ao tempo de execução](#) (p. 217)
- [AWS Lambda Ambiente de execução do](#) (p. 219)
- [Suporte ao tempo de execução para imagens de contêiner do Lambda](#) (p. 222)
- [AWS Lambda API de tempo de execução do](#) (p. 224)
- [API Extensions do Lambda](#) (p. 228)
- [API Lambda Logs](#) (p. 242)
- [Modificar o ambiente de execução](#) (p. 250)
- [Tempos de execução personalizados do AWS Lambda](#) (p. 255)
- [Tutorial: publicar um tempo de execução personalizado](#) (p. 258)
- [Usar a vetorização AVX2 no Lambda](#) (p. 264)

Política de suporte ao tempo de execução

Tempos de execução do Lambda (p. 214) Os para arquivos .zip se baseiam em uma combinação de sistema operacional, linguagem de programação e bibliotecas de software que estão sujeitos a manutenção e atualizações de segurança. Quando as atualizações de segurança não estiverem mais disponíveis para um componente de um tempo de execução, o Lambda torna o tempo de execução defasado.

A substituição (fim do suporte) para um tempo de execução ocorre em duas fases. Na fase um, o Lambda deixa de aplicar patches de segurança ou outras atualizações ao tempo de execução. Você não consegue mais criar funções que usam o tempo de execução, mas consegue continuar a atualizar funções existentes. Isso inclui atualizar a versão do tempo de execução e reverter para a versão anterior. Observe que as funções que usam o tempo de execução defasado não são mais qualificadas para receber suporte técnico.

Na fase dois, que começa pelo menos 30 dias após o início da primeira, você não consegue mais criar ou atualizar funções que usam o tempo de execução. Para atualizar uma função, você precisa migrá-la para uma versão de tempo de execução com suporte. Depois de migrar a função para uma versão de tempo de execução com suporte, não é possível reverter a função para o tempo de execução anterior.

O Lambda não bloqueia invocações de funções que usam versões de tempo de execução defasadas. As invocações de função continuam indefinidamente depois que a versão do tempo de execução chega ao fim do suporte. No entanto, a AWS recomenda fortemente que você migre funções para uma versão de tempo de execução com suporte para que você continue recebendo patches de segurança e permaneça qualificado para receber suporte técnico.

Important

O Python 2.7 chegou ao fim da vida útil em 1º de janeiro de 2020. O fim do suporte (fase 1) para o tempo de execução do Python 2.7 começou em 15 de julho de 2021. Para obter mais informações, consulte [Anunciando o fim do suporte para Python 2.7 no AWS Lambda](#) no blog da AWS Computação.

Os seguintes tempos de execução atingiram ou estão programados para o fim do suporte:

Datas de fim de suporte de tempos de execução

Nome	Identifier	Sistema operacional	Início da fase um do fim do suporte	Início da fase dois do fim do suporte
.NET Core 2.1	dotnetcore2.1	Amazon Linux	20 de setembro de 2021	20 de outubro de 2021
Python 2.7	python2.7	Amazon Linux	15 de julho de 2021	30 de setembro de 2021
Ruby 2.5	ruby2.5	Amazon Linux	30 de julho de 2021	30 de agosto de 2021
Node.js 10.x	nodejs10.x	Amazon Linux 2	30 de julho de 2021	17 de setembro de 2021
Node.js 8.10	nodejs8.10	Amazon Linux		6 de março de 2020
Node.js 6.10	nodejs6.10	Amazon Linux		12 de agosto de 2019
Borda do Node.js 4.3	nodejs4.3-edge	Amazon Linux		30 de abril de 2019

Nome	Identifier	Sistema operacional	Início da fase um do fim do suporte	Início da fase dois do fim do suporte
Node.js 4.3	nodejs4.3	Amazon Linux		6 de março de 2020
Node.js 0.10	nodejs	Amazon Linux		31 de outubro de 2016
.NET Core 2.0	dotnetcore2.0	Amazon Linux		30 de maio de 2019
.NET Core 1.0	dotnetcore1.0	Amazon Linux		30 de julho de 2019

Em quase todos os casos, a data do fim da vida útil de uma versão da linguagem ou de um sistema operacional é conhecida com antecedência. O Lambda notifica você por e-mail, caso você tenha funções usando um tempo de execução que está agendado para o fim do suporte nos próximos 60 dias. Em alguns casos, pode ser inviável enviar um aviso prévio sobre o término do suporte. Por exemplo, no caso de problemas de segurança que exigem uma atualização retro incompatível ou um componente de tempo de execução que não fornece uma programação de suporte de longo prazo (LTS).

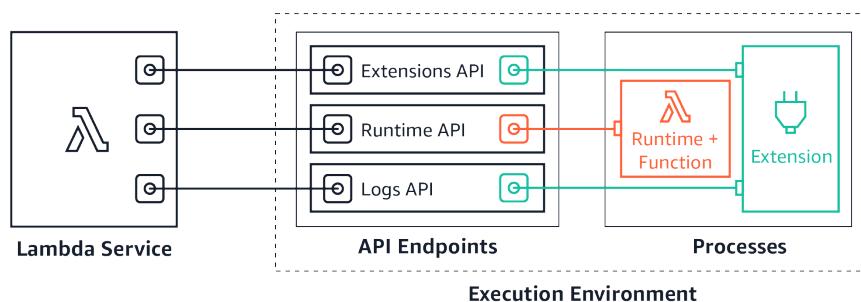
Políticas de suporte de framework e linguagem

- Node.js: github.com
- Python: devguide.python.org
- Ruby: www.ruby-lang.org
- Java: www.oracle.com Perguntas frequentes sobre o Corretto
- Go: golang.org
- .NET Core: dotnet.microsoft.com

AWS Lambda Ambiente de execução do

O Lambda invoca a função em um ambiente de execução, que fornece um ambiente do tempo de execução seguro e isolado. O ambiente de execução gerencia os recursos necessários para executar a função. O ambiente de execução também fornece suporte ao ciclo de vida para o tempo de execução da função e qualquer [extensão externa \(p. 178\)](#) associada à função.

O tempo de execução da função se comunica com o Lambda usando a [API de tempo de execução \(p. 224\)](#). As extensões se comunicam com o Lambda usando a [API Extensions \(p. 228\)](#). As extensões também podem receber mensagens de log da função assinando registros usando a [API Logs \(p. 242\)](#).



Quando você cria uma função do Lambda, você especifica informações de configuração, como a quantidade de memória disponível e o tempo máximo de execução permitido para sua função. O Lambda usa essas informações para configurar o ambiente de execução.

O tempo de execução da função e cada extensão externa são processos executados dentro do ambiente de execução. Permissões, recursos, credenciais e variáveis de ambiente são compartilhados entre a função e as extensões.

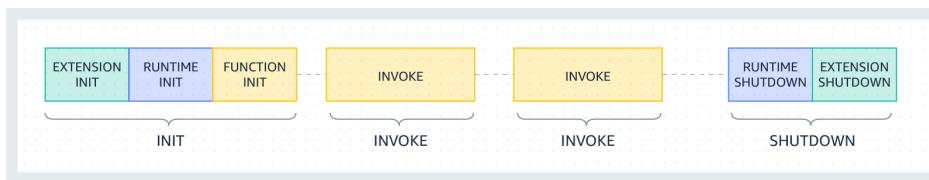
Ciclo de vida do ambiente de execução do Lambda

O ciclo de vida do ambiente de execução inclui as seguintes fases:

- **Init:** nessa fase, o Lambda cria ou descongela um ambiente de execução com os recursos configurados, faz download do código da função e de todas as camadas, inicializa todas as extensões e o tempo de execução, em seguida executa o código de inicialização da função (o código fora do handler principal). O init ocorre durante a primeira invocação, ou antes de invocações de função se você tiver habilitado [Simultaneidade provisionada \(p. 116\)](#).

O init é dividida em três subfases: Extension init, Runtime init, e Function init. Essas subfases garantem que todas as extensões e o tempo de execução concluam suas tarefas de configuração antes que o código da função seja executado.

- **Invoke:** Nesta fase, o Lambda chama o manipulador de função. Depois que a função é executada até a conclusão, o Lambda se prepara para manipular outra invocação de função.
- **Shutdown:** Esta fase é acionada se a função Lambda não receber quaisquer invocações por um período de tempo. No shutdown, o Lambda encerra o tempo de execução, alerta as extensões para permitir que elas parem de forma limpa e, em seguida, remove o ambiente. Lambda envia uma shutdown para cada extensão, que informa a extensão que o ambiente está prestes a ser encerrado.



Cada fase começa com um evento que o Lambda envia para o tempo de execução e para todas as extensões registradas. O tempo de execução e cada extensão indicam a conclusão enviando uma solicitação de API `Next`. O Lambda congela o ambiente de execução quando o tempo de execução e cada extensão tiverem sido concluídos e não houver eventos pendentes.

Tópicos

- [Fase de inicialização \(p. 220\)](#)
- [Fase de invocação \(p. 220\)](#)
- [Fase de desligamento \(p. 221\)](#)

Fase de inicialização

Na fase `Init`, o Lambda executa três tarefas:

- Iniciar todas as extensões (`Extension init`)
- Realizar bootstrap no tempo de execução (`Runtime init`)
- Executar o código estático da função (`Function init`)

A fase `Init` termina quando o tempo de execução e todas as extensões sinalizam que estão prontas enviando uma solicitação de API `Next`. A fase `Init` é limitada a 10 segundos. Se todas as três tarefas não forem concluídas em até 10 segundos, o Lambda repete a fase `Init` no momento da primeira invocação de função.

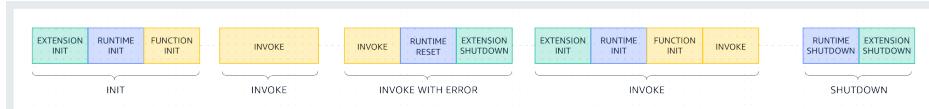
Fase de invocação

Quando uma função do Lambda é invocada em resposta a uma solicitação de API `Next`, o Lambda envia um evento `Invoke` para o tempo de execução e para cada extensão.

A configuração de tempo limite da função limita a duração de toda a fase `Invoke`. Por exemplo, se você definir o tempo limite da função como 360 segundos, a função e todas as extensões precisam ser concluídas em até 360 segundos. Observe que não há fase de pós-invocação independente. A duração é a soma de todo o tempo de invocação (tempo de execução + extensões) e não é calculada até que a função e todas as extensões tenham terminado a execução.

A fase de invocação termina após o tempo de execução e todas as extensões sinalizam que elas foram concluídas enviando uma solicitação de API `Next`.

Se a função do Lambda falhar ou expirar durante o `Invoke` fase, o Lambda redefine o ambiente de execução. A redefinição se comporta como um evento `Shutdown`. Primeiro, o Lambda desliga o tempo de execução. Depois, o Lambda envia um evento `Shutdown` para cada extensão externa registrada. O evento inclui o motivo do desligamento. Se outro evento `Invoke` resultar na reutilização desse ambiente de execução, o Lambda inicializará o tempo de execução e as extensões como parte da próxima invocação.



Fase de desligamento

Quando o Lambda está prestes a encerrar o tempo de execução, ele envia um evento `Shutdown` para o tempo de execução e para cada extensão externa. As extensões podem usar esse tempo para tarefas de limpeza finais. O evento `Shutdown` é uma resposta a uma solicitação de API `Next`.

Duração: toda a fase `Shutdown` é limitada a 2 segundos. Se o tempo de execução ou qualquer extensão não responder, o Lambda o encerrará por meio de um sinal (`SIGKILL`).

Após a conclusão da função e de todas as extensões, o Lambda mantém o ambiente de execução por algum tempo à espera de uma outra invocação de função. Na verdade, o Lambda congela o ambiente de execução. Quando a função é invocada novamente, o Lambda descongela o ambiente para reutilização. Reutilizar o ambiente de execução tem as seguintes implicações:

- Os objetos declarados fora do método do manipulador da função permanecem inicializados, fornecendo otimização adicional quando a função é invocada novamente. Por exemplo, se sua função do Lambda estabelecer uma conexão com o banco de dados, em vez de restabelecer a conexão, a conexão original é usada em invocações subsequentes. Recomendamos que você adicione lógica em seu código para verificar se uma conexão existente antes de criar outra.
- Cada ambiente de execução fornece 512 MB de espaço em disco no diretório `/tmp`. O conteúdo do diretório permanece quando o ambiente de execução é congelado, fornecendo cache transitório que pode ser usado para várias invocações. Você pode adicionar um código extra para verificar se o cache tem os dados que você armazenou. Para obter mais informações sobre limites de tamanho de implantação, consulte [Cotas Lambda \(p. 53\)](#).
- Processos em segundo plano ou retornos de chamada que foram iniciados pela função do Lambda e não foram concluídos quando a função terminou serão retomados se o Lambda reutilizar o ambiente de execução. Garanta que todos os processos em segundo plano ou retornos de chamadas no seu código sejam concluídos antes que o código seja encerrado.

Ao escrever o código da função, não presuma que o Lambda reutilizará automaticamente o ambiente de execução para invocações subsequentes da função. Outros fatores podem ditar a necessidade de o Lambda criar um ambiente de execução, o que pode gerar resultados inesperados, como falhas de conexão do banco de dados.

Suporte ao tempo de execução para imagens de contêiner do Lambda

AWSA oferece um conjunto de imagens de base de código aberto que você pode usar. Você também pode usar uma comunidade preferida ou imagem de base privada. O Lambda oferece software cliente que você adiciona à sua imagem base preferida para torná-lo compatível com o serviço do Lambda.

Tópicos

- [AWSImagens base para o Lambda \(p. 222\)](#)
- [Imagens de base para tempos de execução personalizados \(p. 222\)](#)
- [Clientes de interface de tempo de execução \(p. 223\)](#)
- [Emulador de interface de tempo de execução \(p. 223\)](#)

AWSImagens base para o Lambda

Você pode usar uma das imagens de base da AWS para o Lambda para criar a imagem de contêiner do código de função. As imagens de base são pré-carregadas com um tempo de execução de linguagem e outros componentes necessários para executar uma imagem de contêiner no Lambda. Você adiciona seu código de função e as dependências à imagem de base e, em seguida, empacota-os como uma imagem de contêiner.

AWSA manterá e atualizará regularmente essas imagens. Além disso, a AWS lançará imagens básicas da AWS quando qualquer tempo de execução gerenciado novo estiver disponível.

O Lambda oferece imagens de base para os seguintes tempos de execução:

- [Node.js \(p. 520\)](#)
- [Python \(p. 548\)](#)
- [Java \(p. 606\)](#)
- [.NET \(p. 674\)](#)
- [Go \(p. 643\)](#)
- [Ruby \(p. 574\)](#)

Imagens de base para tempos de execução personalizados

A AWS oferece imagens de base que contêm os componentes necessários do Lambda e o sistema operacional Amazon Linux ou Amazon Linux2. Você pode adicionar seu tempo de execução preferido, dependências e código a essas imagens.

Tags	Tempo de execução	Sistema operacional
al2	provided.al2	Amazon Linux 2
alami	fornecido	Amazon Linux

DockerHub: [amazon/aws-lambda-provided](#)

ECR PÚBLICO: public.ecr.aws/lambda/provided

Clientes de interface de tempo de execução

O cliente de interface de tempo de execução em sua imagem de contêiner gerencia a interação entre o Lambda e seu código de função. A [API Runtime \(p. 224\)](#), junto à [API Extensions \(p. 228\)](#), define uma interface HTTP simples para tempos de execução para receber eventos de invocação do Lambda e responder com indicações de sucesso ou falha.

Cada uma das imagens de base da AWS para o Lambda inclui um cliente de interface de tempo de execução. Se você escolher uma das imagens base para tempos de execução personalizados ou uma imagem de base alternativa, será necessário adicionar o cliente de interface de tempo de execução apropriado.

Para sua conveniência, o Lambda oferece um cliente de interface de tempo de execução de código aberto para cada um dos tempos de execução do Lambda compatíveis:

- [Node.js \(p. 521\)](#)
- [Python \(p. 549\)](#)
- [Java \(p. 607\)](#)
- [.NET \(p. 675\)](#)
- [Go \(p. 643\)](#)
- [Ruby \(p. 575\)](#)

Emulador de interface de tempo de execução

O Lambda oferece um runtime interface emulator (RIE – emulador de interface de tempo de execução) para você testar sua função localmente. OAWSImagens base para o Lambda e imagens base para tempos de execução personalizados incluem o RIE. Para outras imagens de base, você pode fazer o download do [emulador de interface de tempo de execução](#) do repositório do GitHub da AWS.

AWS Lambda API de tempo de execução do

O AWS Lambda fornece uma API HTTP para [tempos de execução personalizados \(p. 255\)](#) para receber eventos de invocação do Lambda e enviar dados de resposta de volta para o [ambiente de execução \(p. 214\)](#) do Lambda.

A especificação OpenAPI para a versão da API de tempo de execução 2018-06-01 está disponível aqui: [runtime-api.zip](#)

Para criar um URL de solicitação de API, os tempos de execução obtêm o endpoint da API da variável de ambiente do `AWS_LAMBDA_RUNTIME_API`, adiciona a versão da API e o caminho de recurso desejado.

Example Request

```
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next"
```

Métodos de API

- [Próxima invocação \(p. 224\)](#)
- [Resposta de invocação \(p. 225\)](#)
- [Erro de inicialização \(p. 225\)](#)
- [Erro de invocação \(p. 226\)](#)

Próxima invocação

Caminho: `/runtime/invocation/next`

Método – GET

O tempo de execução envia essa mensagem ao Lambda para solicitar um evento de invocação. O corpo da resposta contém a carga útil da invocação, que é um documento JSON que contém os dados do evento do acionador da função. Os cabeçalhos de resposta contêm dados adicionais sobre a invocação.

Cabeçalhos de resposta

- `Lambda-Runtime-Aws-Request-Id`: o ID da solicitação, que identifica a solicitação que acionou a invocação da função.

Por exemplo, `8476a536-e9f4-11e8-9739-2dfe598c3fcd`.

- `Lambda-Runtime-Deadline-Ms`: a data em que a função expira tempo em milissegundos do Unix.

Por exemplo, `1542409706888`.

- `Lambda-Runtime-Invoked-Function-Arn`: o ARN da função do Lambda, versão ou alias especificado na invocação.

Por exemplo, `arn:aws:lambda:us-east-2:123456789012:function:custom-runtime`.

- `Lambda-Runtime-Trace-Id`: o [cabeçalho de rastreamento do AWS X-Ray](#).

Por exemplo, `Root=1-5bef4de7-ad49b0e87f6ef6c87fc2e700;Parent=9a9197af755a6419;Sampled=1`.

- `Lambda-Runtime-Client-Context`: para invocações do AWS Mobile SDK, os dados sobre a aplicação cliente e o dispositivo.

- `Lambda-Runtime-Cognito-Identity`: para invocações do AWS Mobile SDK, os dados sobre o provedor de identidade do Amazon Cognito.

Não defina um tempo limite na solicitação GET, pois a resposta poderá estar atrasada. Entre o momento em que o Lambda inicializa o tempo de execução e o momento em que o tempo de execução tem um evento para retornar, o processo do tempo de execução pode ficar congelado por vários segundos.

O ID da solicitação rastreia a invocação dentro do Lambda. Use-o para especificar a invocação ao enviar a resposta.

O cabeçalho de rastreamento contém o ID de rastreamento, o ID pai e a decisão de amostragem. Se a solicitação for de amostra, a amostra da solicitação foi feita pelo Lambda ou um serviço upstream. O tempo de execução deve definir o `_X_AMZN_TRACE_ID` com o valor do cabeçalho. O X-Ray SDK lê isso para obter os IDs e determinar se deve rastrear a solicitação.

Resposta de invocação

Caminho: /runtime/invocation/*AwsRequestId*/response

Método – POST

Depois que a função for executada até a conclusão, o tempo de execução envia uma resposta de invocação para o Lambda. Para invocações síncronas, o Lambda envia a resposta de volta para o cliente.

Example solicitação com êxito

```
REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9
curl -X POST "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/
response" -d "SUCCESS"
```

Erro de inicialização

Se a função retornar um erro ou o tempo de execução encontrar um erro durante a inicialização, o tempo de execução usará esse método para relatar o erro ao Lambda.

Caminho: /runtime/init/error

Método – POST

Cabeçalhos

`Lambda-Runtime-Function-Error-Type`: tipo de erro encontrado pela extensão. Obrigatório: não

Este cabeçalho consiste em um valor de string. O Lambda aceita qualquer string, mas recomendamos o formato `<category.reason>`. Por exemplo:

- `Runtime.NoSuchHandler`
- `Extension.APIKeyNotFound`
- `Extension.ConfigInvalid`
- `Extension.UnknownReason`

Body parameters (Parâmetros do corpo)

`ErrorRequest`: informações adicionais sobre o erro. Obrigatório: não

Este campo é um objeto JSON com a seguinte estrutura:

```
{
    errorMessage: string (text description of the error),
```

```
    errorType: string,  
    stackTrace: array of strings  
}
```

Observe que o Lambda aceita qualquer valor para `errorType`.

O exemplo a seguir mostra uma mensagem de erro de função do Lambda na qual a função não pôde analisar os dados do evento fornecidos na chamada.

Example Erro de função

```
{  
  "errorMessage" : "Error parsing event data.",  
  "errorType" : "InvalidEventDataException",  
  "stackTrace": [ ]  
}
```

Parâmetros do corpo da resposta

- `StatusResponse` – String. Informações de status, enviadas com 202 códigos de resposta.
- `ErrorResponse`: informações adicionais de erro, enviadas com os códigos de resposta de erro. O `ErrorResponse` contém um tipo de erro e uma mensagem de erro.

Códigos de resposta

- 202: aceito
- 403: proibido
- 500: erro de contêiner. Estado não recuperável. A extensão deve sair imediatamente.

Example solicitação com erro de inicialização

```
ERROR={"\\"errorMessage\" : \\"Failed to load function.\\"", \\"errorType\" :  
  \\"InvalidFunctionException\"\\"}  
curl -X POST "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/init/error" -d "$ERROR"  
--header "Lambda-Runtime-Function-Error-Type: Unhandled"
```

Erro de invocação

Se a função retornar um erro ou o tempo de execução encontrar um erro, o tempo de execução usará esse método para relatar o erro ao Lambda.

Caminho: `/runtime/invocation/AwsRequestId/error`

Método – POST

Cabeçalhos

`Lambda-Runtime-Function-Error-Type`: tipo de erro encontrado pela extensão. Obrigatório: não

Este cabeçalho consiste em um valor de string. O Lambda aceita qualquer string, mas recomendamos o formato `<category.reason>`. Por exemplo:

- `Runtime.NoSuchHandler`
- `Extension.APIKeyNotFound`
- `Extension.ConfigInvalid`

- Extension.UnknownReason

Body parameters (Parâmetros do corpo)

ErrorRequest: informações adicionais sobre o erro. Obrigatório: não

Este campo é um objeto JSON com a seguinte estrutura:

```
{  
    errorMessage: string (text description of the error),  
    errorType: string,  
    stackTrace: array of strings  
}
```

Observe que o Lambda aceita qualquer valor para errorType.

O exemplo a seguir mostra uma mensagem de erro de função do Lambda na qual a função não pôde analisar os dados do evento fornecidos na chamada.

Example Erro de função

```
{  
    "errorMessage" : "Error parsing event data.",  
    "errorType" : "InvalidEventDataException",  
    "stackTrace": [ ]  
}
```

Parâmetros do corpo da resposta

- StatusResponse – String. Informações de status, enviadas com 202 códigos de resposta.
- ErrorResponse: informações adicionais de erro, enviadas com os códigos de resposta de erro. O ErrorResponse contém um tipo de erro e uma mensagem de erro.

Códigos de resposta

- 202: aceito
- 400: solicitação inválida
- 403: proibido
- 500: erro de contêiner. Estado não recuperável. A extensão deve sair imediatamente.

Example solicitação com erro

```
REQUEST_ID=156cb537-e2d4-11e8-9b34-d36013741fb9  
ERROR={"errorMessage" : "Error parsing event data.\\", \"errorType\" :  
    \"InvalidEventDataException\""}  
curl -X POST "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/  
error" -d "$ERROR" --header "Lambda-Runtime-Function-Error-Type: Unhandled"
```

API Extensions do Lambda

Autores de funções do Lambda usam extensões para integrar o Lambda às suas ferramentas preferidas para monitoramento, observabilidade, segurança e governança. Os autores de funções podem usar extensões da AWS, de parceiros da AWS e de projetos de código aberto. Para obter mais informações, consulte [Introducing AWS Lambda Extensions](#) no AWS Compute Blog.

Como um autor de extensões, você pode usar as API de extensões do Lambda para integrar profundamente no [ambiente de execução \(p. 219\)](#) do Lambda. Sua extensão pode inscrever-se para eventos de ciclo de vida do ambiente de execução e função. Em resposta a esses eventos, você pode iniciar novos processos, executar lógica e controlar e participar de todas as fases do ciclo de vida do Lambda: inicialização, invocação e desligamento. Além disso, você pode usar os [API Runtime Logs \(p. 242\)](#) (Logs de tempo de execução da API) para receber um stream de logs.

Uma extensão externa é executada como um processo independente no ambiente de execução e continua a ser executada após a invocação da função ser totalmente processada. Como as extensões são executadas como processos, elas podem ser escritas em uma linguagem diferente da função. Recomendamos que você implemente as extensões usando uma linguagem compilada. Nesse caso, a extensão é um binário autônomo compatível com todos os tempos de execução compatíveis. Se você usar uma linguagem não compilada, inclua um tempo de execução compatível na extensão.

Os seguintes [tempos de execução do Lambda \(p. 214\)](#) são compatíveis com extensões:

- .NET Core 3.1 (C#/PowerShell) (`dotnetcore3.1`)
- Tempo de execução personalizado (`provided`)
- Tempo de execução personalizado no Amazon Linux 2 (`provided.al2`)
- Java 11 (Corretto) (`java11`)
- Java 8 (Corretto) (`java8.al2`)
- Node.js 14.x (`nodejs14.x`)
- Node.js 12.x (`nodejs12.x`)
- Node.js 10.x (`nodejs10.x`)
- Python 3.9 (`python3.9`)
- Python 3.8 (`python3.8`)
- Python 3.7 (`python3.7`)
- Ruby 2.7 (`ruby2.7`)
- Ruby 2.5 (`ruby2.5`)

Observe que o tempo de execução Go 1.x não suporta extensões. Para suportar extensões, você pode criar funções Go `noprovided.al2`Tempo de execução. Para obter mais informações, consulte [Migração de funções do Lambda para o Amazon Linux 2](#).

O Lambda também oferece suporte a extensões internas. Uma extensão interna é executada como um thread separado no processo de tempo de execução. O tempo de execução inicia e interrompe a extensão interna. Uma maneira alternativa de se integrar com o ambiente do Lambda é usar [variáveis de ambiente com linguagem específica e scripts wrapper \(p. 250\)](#). Você pode usar isso para configurar o ambiente do tempo de execução e modificar o comportamento de inicialização do processo do tempo de execução.

Você pode adicionar extensões a uma função de duas maneiras. Para uma função implantada como um [arquivo.zip \(p. 37\)](#), você implanta a extensão como uma [camada \(p. 85\)](#). Para uma função definida como uma imagem de contêiner, você adiciona [as extensões \(p. 180\)](#) à imagem do contêiner.

Note

Por exemplo, extensões e scripts de wrapper, consulte [AWS Lambda Extensões](#) no repositório AWS Samples GitHub.

Tópicos

- [Ciclo de vida do ambiente de execução do Lambda \(p. 229\)](#)
- [Referência de API de extensões \(p. 237\)](#)

Ciclo de vida do ambiente de execução do Lambda

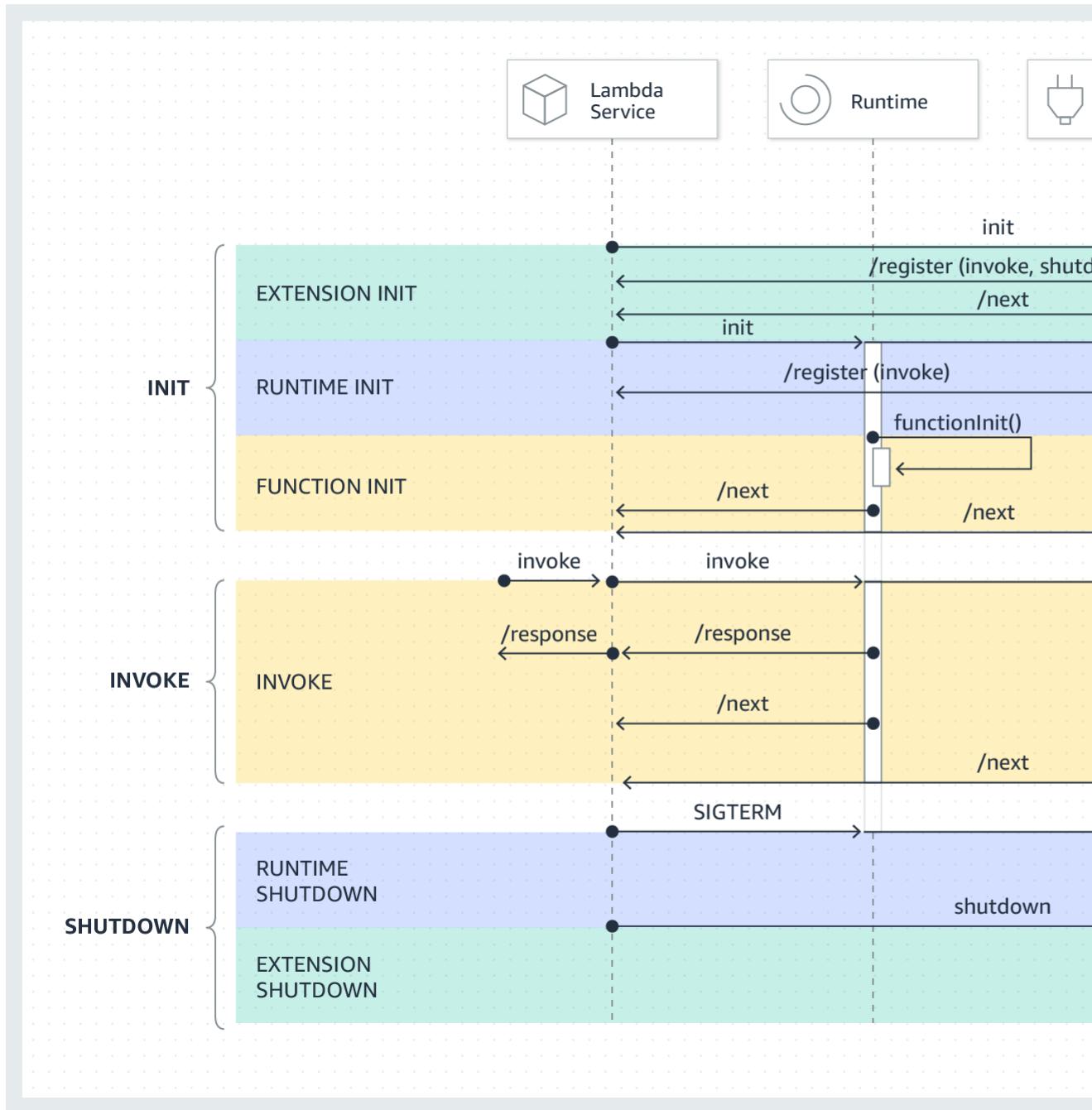
O ciclo de vida do ambiente de execução inclui as seguintes fases:

- **Init:** nessa fase, o Lambda cria ou descongela um ambiente de execução com os recursos configurados, faz download do código da função e de todas as camadas, inicializa todas as extensões e o tempo de execução, em seguida executa o código de inicialização da função (o código fora do handler principal). O `Init` ocorre durante a primeira invocação, ou antes de invocações de função se você tiver habilitado [Simultaneidade provisionada \(p. 116\)](#).

O `Init` é dividido em três subfases: `Extension init`, `Runtime init`, e `Function init`. Essas subfases garantem que todas as extensões e o tempo de execução concluam suas tarefas de configuração antes que o código da função seja executado.

- **Invoke:** Nesta fase, o Lambda chama o manipulador de função. Depois que a função é executada até a conclusão, o Lambda se prepara para manipular outra invocação de função.
- **Shutdown:** Esta fase é acionada se a função Lambda não receber quaisquer invocações por um período de tempo. No `Shutdown`, o Lambda encerra o tempo de execução, alerta as extensões para permitir que elas parem de forma limpa e, em seguida, remove o ambiente. O Lambda envia um `Shutdown` para cada extensão, que informa a extensão que o ambiente está prestes a ser encerrado.

Cada fase começa com um evento do serviço do Lambda para o tempo de execução e para todas as extensões registradas. O tempo de execução e cada conclusão do sinal de extensão enviando um `Next` solicitação de API. O Lambda congela o ambiente de execução quando cada processo é concluído e não há eventos pendentes.



Tópicos

- [Fase de inicialização \(p. 231\)](#)
- [Fase de invocação \(p. 220\)](#)
- [Fase de desligamento \(p. 221\)](#)
- [Permissões e configuração \(p. 236\)](#)
- [Tratamento de falhas \(p. 236\)](#)
- [Solução de problemas de extensões \(p. 237\)](#)

Fase de inicialização

Durante o `Extension init`, cada extensão precisa se registrar com o Lambda para receber eventos. O Lambda usa o nome completo do arquivo da extensão para validar que a extensão concluiu a sequência de bootstrap. Portanto, cada chamada de API `Register` deve incluir o cabeçalho `Lambda-Extension-Name` com o nome de arquivo completo da extensão.

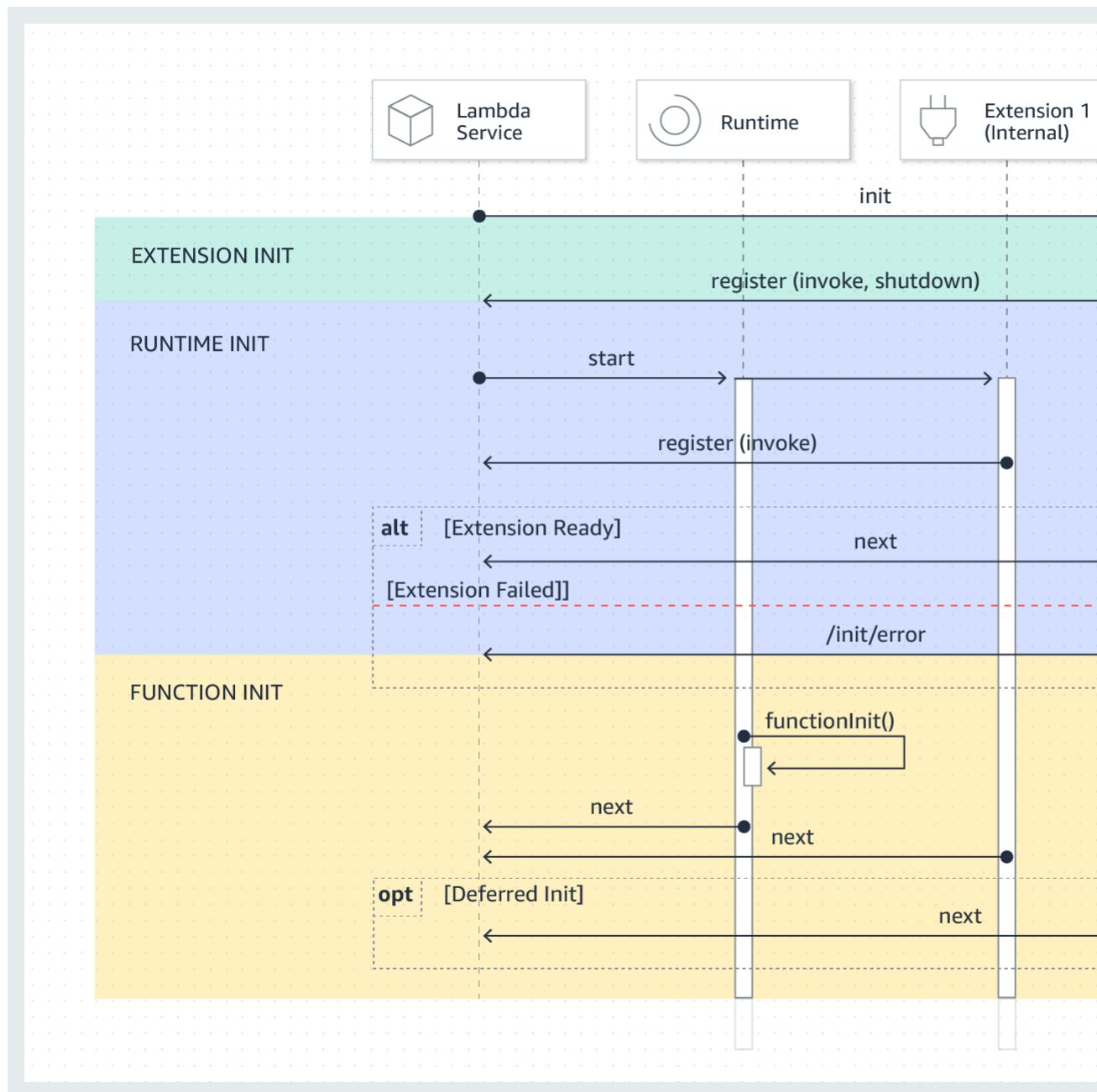
É possível registrar até 10 extensões para uma função. Esse limite é imposto pela chamada de API `Register`.

Depois que cada extensão é registrada, o Lambda inicia a fase `Runtime init`. O processo de tempo de execução chama `functionInit` para iniciar a `Function init` fase.

A fase `Init` é concluída após o tempo de execução e cada extensão registrada indica a conclusão enviando uma solicitação de API `Next`.

Note

As extensões podem concluir sua inicialização em qualquer momento da fase `Init`.



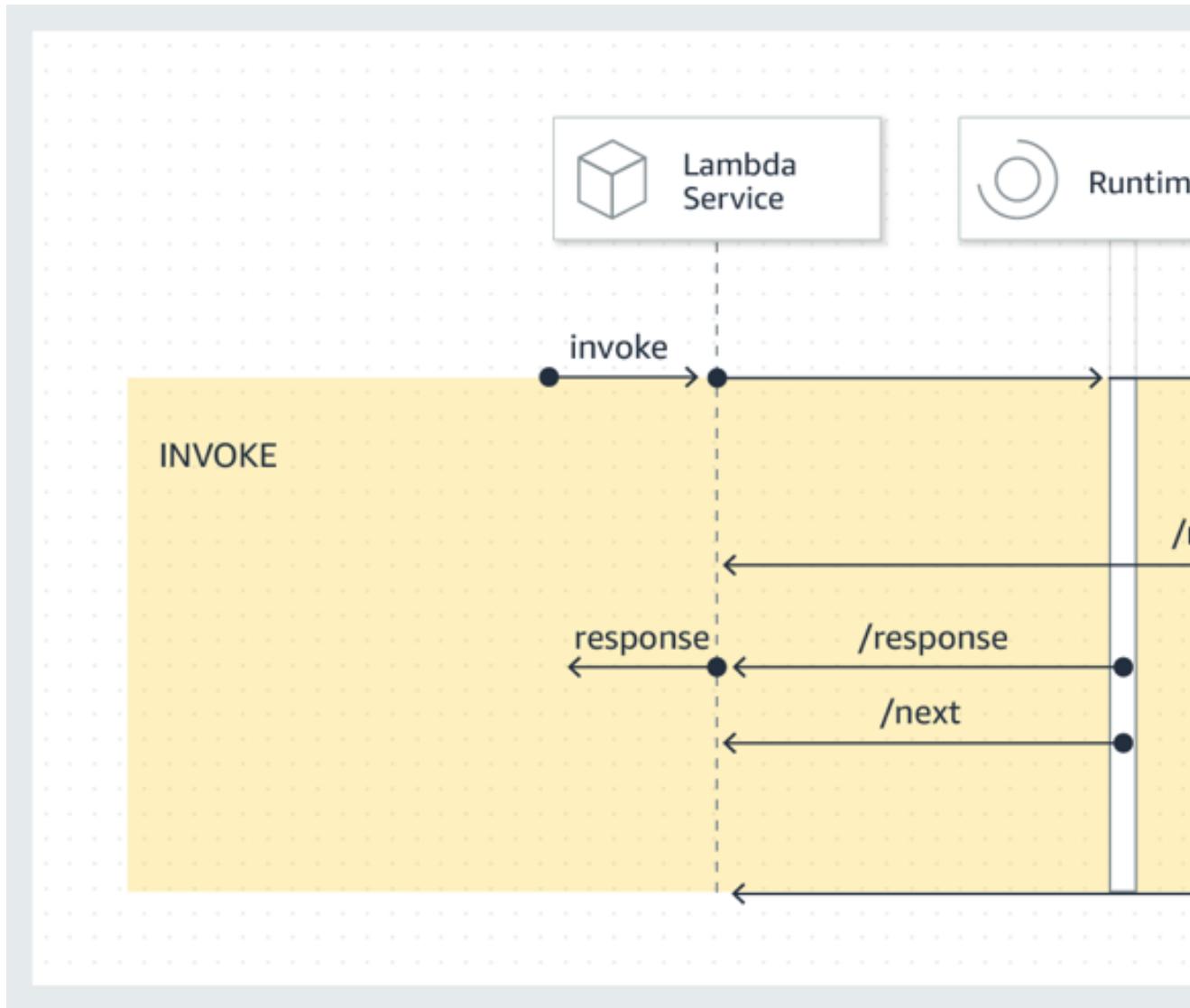
Fase de invocação

Quando uma função do Lambda é invocada em resposta a uma solicitação de API `Next`, o envia um evento `Invoke` para o tempo de execução e para cada extensão registrada para o evento `Invoke`.

Durante a invocação, as extensões externas são executadas em paralelo com a função. Elas também continuam em execução após a conclusão da função. Isso permite que você capture informações para diagnóstico ou para enviar logs, métricas e rastreamentos para um local de sua escolha.

Depois de receber a resposta da função do tempo de execução, o Lambda retorna a resposta ao cliente, mesmo que as extensões ainda estejam em execução.

A fase `Invoke` termina após o tempo de execução e todas as extensões sinalizam que elas foram concluídas enviando uma solicitação de API `Next`.



Carga útil do evento: o evento enviado para o tempo de execução (e a função do Lambda) leva toda a solicitação, os cabeçalhos (como `RequestId`) e carga útil. O evento enviado para cada extensão contém metadados que descrevem o conteúdo do evento. Este evento de ciclo de vida inclui o tipo do evento, o tempo em que a função times-out (`deadlineMs`), `requestId`, o nome de recurso da Amazon (ARN) da função chamada e cabeçalhos de rastreamento.

As extensões que desejam acessar o corpo do evento de função podem usar um SDK no tempo de execução que se comunica com a extensão. Os desenvolvedores de funções usam o SDK no tempo de execução para enviar a carga útil para a extensão quando a função é invocada.

Veja um exemplo de carga útil:

```
{  
  "eventType": "INVOCATION",  
  "deadlineMs": 676051,  
  "requestId": "3da1f2dc-3222-475e-9205-e2e6c6318895",  
  "invokedFunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:ExtensionTest",  
  "payload": "Some useful payload data..."}
```

```
"tracing": {  
    "type": "X-Amzn-Trace-Id",  
    "value":  
    "Root=1-5f35ae12-0c0fec141ab77a00bc047aa2;Parent=2be948a625588e32;Sampled=1"  
}
```

Límite de duração: a definição do tempo limite da função limita a duração de toda a fase `Invoke`. Por exemplo, se você definir o tempo limite da função como 360 segundos, a função e todas as extensões precisam ser concluídas em até 360 segundos. Observe que não há fase de pós-invocação independente. A duração é a soma de todo o tempo de invocação (tempo de execução + extensões) e não é calculada até que a função e todas as extensões tenham terminado a execução.

Impacto na performance e sobrecarga de extensão: as extensões podem afetar a performance da função. Como autor da extensão, você tem controle sobre o impacto de performance da extensão. Por exemplo, se a extensão executa operações com uso intenso de computação, a duração da função aumenta, pois a extensão e o código da função compartilham os mesmos recursos de CPU. Além disso, se a extensão executar operações extensas após a conclusão da invocação da função, a duração da função aumentará porque a fase `Invoke` continuará até que todas as extensões sinalizem que foram concluídas.

Note

O Lambda aloca a potência da CPU em proporção à configuração de memória da função. Você pode ver maior duração de execução e inicialização em configurações de memória mais baixas porque os processos de função e extensão estão competindo pelos mesmos recursos da CPU. Para reduzir a duração da execução e inicialização, tente aumentar a configuração de memória.

Para ajudar a identificar o impacto na performance apresentado pelas extensões na fase `Invoke`, o Lambda gera a métrica `PostRuntimeExtensionsDuration`. Essa métrica mede o tempo cumulativo gasto entre a solicitação de API `Next` do tempo de execução e a última solicitação de API `Next` da extensão. Para medir o aumento da memória usada, use a métrica `MaxMemoryUsed`. Para obter mais informações sobre métricas de funções, consulte [Trabalhar com métricas de função do AWS Lambda \(p. 717\)](#).

Os desenvolvedores de funções podem executar diferentes versões de suas funções lado a lado para entender o impacto de uma extensão específica. Recomendamos que os autores de extensão publiquem o consumo esperado de recursos para facilitar aos desenvolvedores de funções a escolha de uma extensão adequada.

Fase de desligamento

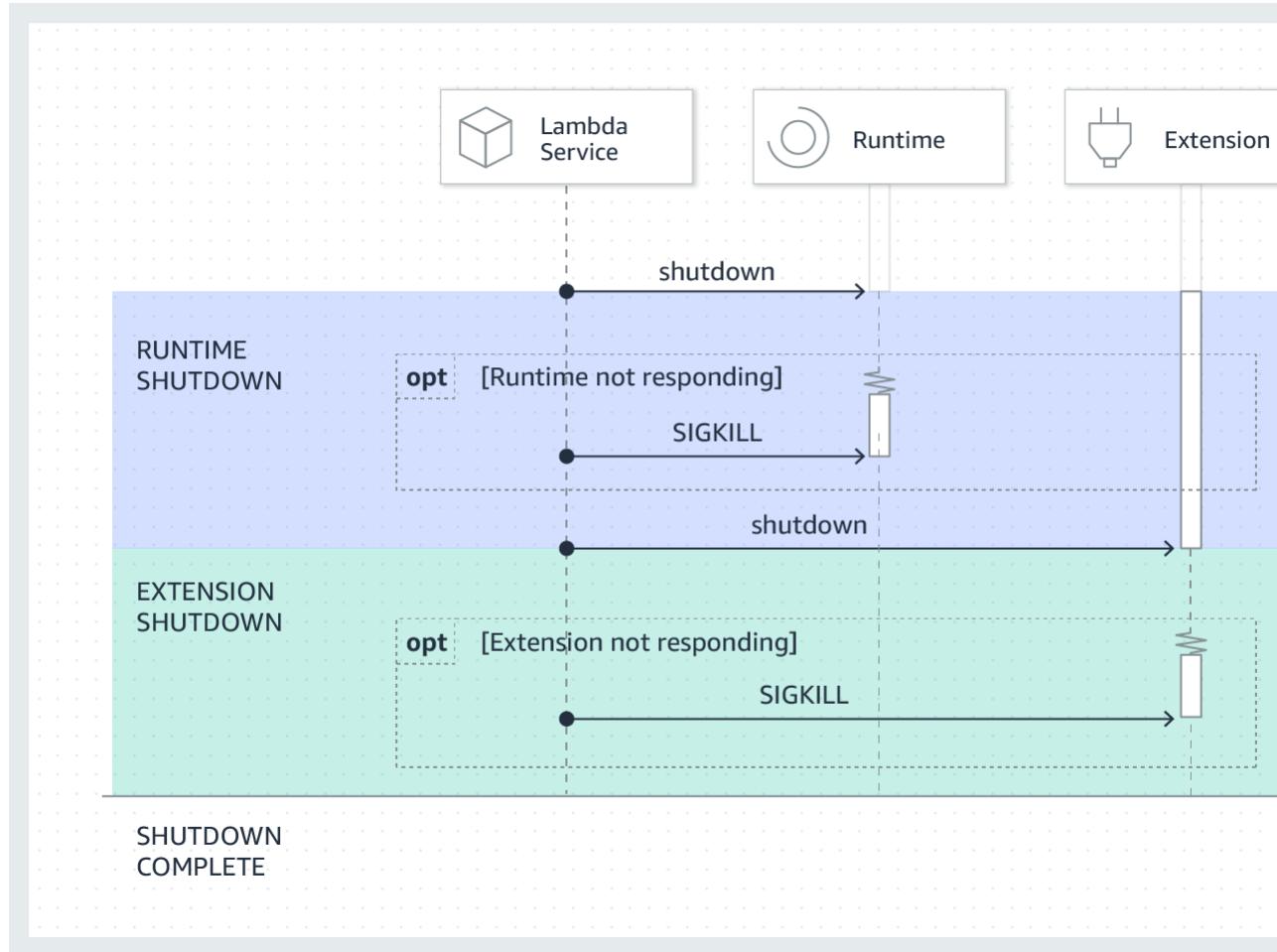
Quando o Lambda estiver prestes a encerrar o tempo de execução, ele enviará um evento `Shutdown` para o tempo de execução e, depois, para cada extensão externa registrada. As extensões podem usar esse tempo para tarefas de limpeza finais. O evento `Shutdown` é enviado em resposta a uma solicitação de API `Next`.

Límite de duração: a duração máxima da fase `Shutdown` depende da configuração das extensões registradas:

- 300 ms: uma função sem extensões registradas
- 500 ms: uma função com uma extensão interna registrada
- 2.000 ms: uma função com uma ou mais extensões externas registradas

Para uma função com extensões externas, o Lambda reserva até 300 ms (500 ms para um tempo de execução com uma extensão interna) para que o processo de tempo de execução execute um desligamento normal. Lambda aloca o restante do limite de 2.000 ms para extensões externas para desligar.

Se o tempo de execução ou uma extensão não responder ao evento `shutdown` dentro do limite, o Lambda encerrará o processo usando um sinal `SIGKILL`.



Carga útil do evento: o evento `Shutdown` contém o motivo do desligamento e o tempo restante em milissegundos.

O `shutdownReason` inclui os seguintes valores:

- `SPINDOWN`: desligamento normal
- `TIMEOUT`: limite da duração do tempo expirado
- `FAILURE`: condição de erro, como um evento `out-of-memory`

```
{  
  "eventType": "SHUTDOWN",  
  "shutdownReason": "reason for shutdown",  
  "deadlineMs": "the time and date that the function times out in Unix time milliseconds"  
}
```

Permissões e configuração

As extensões são executadas no mesmo ambiente de execução que a função do Lambda. As extensões também compartilham recursos com a função, como CPU, memória e armazenamento em disco /tmp. Além disso, as extensões usam a mesma função do AWS Identity and Access Management (IAM) e o mesmo contexto de segurança que a função.

Permissões de acesso ao sistema de arquivos e à rede: as extensões são executadas no mesmo sistema de arquivos e namespace de nome de rede que o tempo de execução da função. Isso significa que as extensões precisam ser compatíveis com o sistema operacional associado. Se uma extensão exigir regras adicionais de saída do tráfego de rede, você deverá aplicar essas regras à configuração da função.

Note

Como o diretório de código de função é somente leitura, as extensões não podem modificar o código da função.

Variáveis de ambiente: as extensões podem acessar as [variáveis de ambiente \(p. 99\)](#) da função, exceto as seguintes variáveis específicas do processo de tempo de execução:

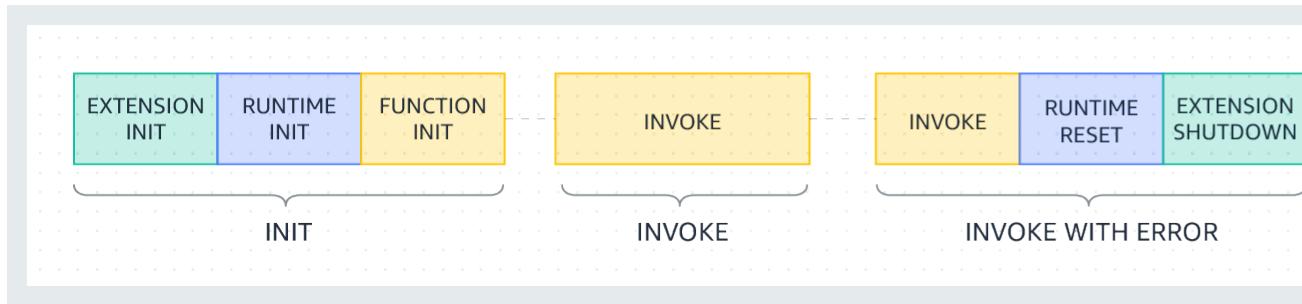
- AWS_EXECUTION_ENV
- AWS_LAMBDA_LOG_GROUP_NAME
- AWS_LAMBDA_LOG_STREAM_NAME
- AWS_XRAY_CONTEXT_MISSING
- AWS_XRAY_DAEMON_ADDRESS
- LAMBDA_RUNTIME_DIR
- LAMBDA_TASK_ROOT
- _AWS_XRAY_DAEMON_ADDRESS
- _AWS_XRAY_DAEMON_PORT
- _HANDLER

Tratamento de falhas

Falhas de inicialização: se uma extensão falhar, o Lambda reiniciará o ambiente de execução para impor um comportamento consistente e incentivar falhas rapidamente para extensões. Além disso, para alguns clientes, as extensões devem atender às necessidades críticas, como registro em log, segurança, governança e coleta de telemetria.

Invocar falhas (como falta de memória, tempo limite da função): como as extensões compartilham recursos com o tempo de execução, elas são afetadas pelo esgotamento da memória. Quando o tempo de execução falha, todas as extensões e o próprio tempo de execução participam da fase Shutdown. Além disso, o tempo de execução é reiniciado automaticamente como parte do chamado atual ou por meio de um mecanismo de reinicialização adiado.

Se houver uma falha (como um erro de tempo limite ou de tempo de execução de uma função) durante Invoke, o serviço Lambda executará uma redefinição. A redefinição se comporta como um evento Shutdown. Primeiro, o Lambda encerra o tempo de execução, depois, envia um evento Shutdown para cada extensão externa registrada. O evento inclui o motivo do desligamento. Se esse ambiente for usado para uma nova invocação, a extensão e o tempo de execução serão reinicializados como parte da próxima invocação.



Logs de extensões: o Lambda envia a saída de log de extensões para o CloudWatch Logs. O Lambda também gera um evento de log adicional para cada extensão durante `Init`. O evento de log registra o nome e a preferência de registro (evento, configuração) em caso de sucesso ou o motivo da falha em caso de falha.

Solução de problemas de extensões

- Se uma solicitação `Register` falhar, o cabeçalho `Lambda-Extension-Name` na chamada de API `Register` deverá conter o nome completo do arquivo da extensão.
- Se a solicitação `Register` falhar para uma extensão interna, a solicitação não deverá estar registrada para o evento `Shutdown`.

Referência de API de extensões

A especificação OpenAPI para a versão da API de extensões 2020-01-01 está disponível aqui: [extensions-api.zip](#)

É possível recuperar o valor do endpoint da API da variável de ambiente `AWS_LAMBDA_RUNTIME_API`. Para enviar uma solicitação `Register`, use o prefixo `2020-01-01/` antes de cada caminho da API. Por exemplo:

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-01-01/extension/register
```

Métodos de API

- [Register \(p. 237\)](#)
- [Next \(p. 238\)](#)
- [Erro de inicialização \(p. 239\)](#)
- [Erro de saída \(p. 240\)](#)

Register

Durante `Extension init`, todas as extensões precisam se registrar no Lambda para receber eventos. O Lambda usa o nome completo do arquivo da extensão para validar que a extensão concluiu a sequência de bootstrap. Portanto, cada chamada de API `Register` deve incluir o cabeçalho `Lambda-Extension-Name` com o nome de arquivo completo da extensão.

As extensões internas são iniciadas e interrompidas pelo processo de tempo de execução, portanto, elas não têm permissão para se registrar para o evento `Shutdown`.

Caminho – `/extension/register`

Método – POST

Cabeçalhos

Lambda-Extension-Name: o nome completo do arquivo da extensão. Obrigatório: sim. Tipo: string.

Body parameters (Parâmetros do corpo)

events: matriz dos eventos para registrar. Obrigatório: não Tipo: matriz de strings. Strings válidas: `INVOKE`, `SHUTDOWN`.

Cabeçalhos de resposta

- **Lambda-Extension-Identifier:** identificador de agente exclusivo gerado (string UUID) que é obrigatório para todas as solicitações subsequentes.

Códigos de resposta

- 200: o corpo de resposta contém o nome da função, a versão da função e o nome do manipulador.
- 400: solicitação inválida
- 403: proibido
- 500: erro de contêiner. Estado não recuperável. A extensão deve sair imediatamente.

Example Exemplo de corpo da solicitação

```
{  
    "events": [ "INVOKE", "SHUTDOWN" ]  
}
```

Example Exemplo de corpo da resposta

```
{  
    "functionName": "helloWorld",  
    "functionVersion": "$LATEST",  
    "handler": "lambda_function.lambda_handler"  
}
```

Next

As extensões enviam uma solicitação de API Next para receber o próximo evento, que pode ser um evento `Invoke` ou um evento `Shutdown`. O corpo da resposta contém a carga útil, que é um documento JSON com dados do evento.

A extensão envia uma solicitação de API Next para sinalizar que está pronta para receber novos eventos. Essa é uma chamada de bloqueio.

Não defina um tempo limite na chamada GET, pois a extensão pode ser suspensa por um período até que haja um evento a ser retornado.

Caminho – `/extension/event/next`

Método – GET

Parâmetros

Lambda-Extension-Identifier: identificador exclusivo para extensão (string UUID). Obrigatório: sim. Tipo: string UUID.

Cabeçalho da resposta

- `Lambda-Extension-Identifier`: identificador de agente exclusivo (string UUID).

Códigos de resposta

- 200: a resposta contém informações sobre o próximo evento (`EventInvoke` ou `EventShutdown`).
- 403: proibido
- 500: erro de contêiner. Estado não recuperável. A extensão deve sair imediatamente.

Erro de inicialização

A extensão usa esse método para relatar um erro de inicialização para o Lambda. Chame-a quando a extensão falhar ao inicializar após ter sido registrada. Depois que o Lambda recebe o erro, não há mais chamadas de API bem-sucedidas. A extensão deve sair depois de receber a resposta do Lambda.

Caminho – `/extension/init/error`

Método – POST

Cabeçalhos

`Lambda-Extension-Identifier`: identificador exclusivo para extensão. Obrigatório: sim. Tipo: string UUID.

`Lambda-Extension-Function-Error-Type`: tipo de erro encontrado pela extensão. Obrigatório: sim.

Este cabeçalho consiste em um valor de string. Lambda aceita qualquer string, mas recomendamos o formato `<category.reason>`. Por exemplo:

- `Runtime.NoSuchHandler`
- `Extension.APIKeyNotFound`
- `Extension.ConfigInvalid`
- `Extension.UnknownReason`

Body parameters (Parâmetros do corpo)

`ErrorRequest`: informações adicionais sobre o erro. Obrigatório: não

Este campo é um objeto JSON com a seguinte estrutura:

```
{  
    errorMessage: string (text description of the error),  
    errorType: string,  
    stackTrace: array of strings  
}
```

Observe que o Lambda aceita qualquer valor para `errorType`.

O exemplo a seguir mostra uma mensagem de erro de função do Lambda na qual a função não pode analisar os dados do evento fornecidos na chamada.

Example Erro de função

```
{
```

```
        "errorMessage" : "Error parsing event data.",
        "errorType" : "InvalidEventDataException",
        "stackTrace": [ ]
}
```

Corpo da resposta

- **Lambda-Extension-Identifier**: identificador de agente exclusivo (string UUID).

Códigos de resposta

- 202: aceito
- 400: solicitação inválida
- 403: proibido
- 500: erro de contêiner. Estado não recuperável. A extensão deve sair imediatamente.

Erro de saída

A extensão usa esse método para relatar um erro ao Lambda antes de sair. Chame-a quando encontrar uma falha inesperada. Depois que o Lambda recebe o erro, não há mais chamadas de API bem-sucedidas. A extensão deve sair depois de receber a resposta do Lambda.

Caminho – /extension/exit/error

Método – POST

Cabeçalhos

Lambda-Extension-Identifier: identificador exclusivo para extensão. Obrigatório: sim. Tipo: string UUID.

Lambda-Extension-Function-Error-Type: tipo de erro encontrado pela extensão. Obrigatório: sim.

Este cabeçalho consiste em um valor de string. Lambda aceita qualquer string, mas recomendamos o formato <category.reason>. Por exemplo:

- Runtime.NoSuchHandler
- Extension.APIKeyNotFound
- Extension.ConfigInvalid
- Extension.UnknownReason

Body parameters (Parâmetros do corpo)

ErrorRequest: informações adicionais sobre o erro. Obrigatório: não

Este campo é um objeto JSON com a seguinte estrutura:

```
{
    errorMessage: string (text description of the error),
    errorType: string,
    stackTrace: array of strings
}
```

Observe que o Lambda aceita qualquer valor para **errorType**.

O exemplo a seguir mostra uma mensagem de erro de função do Lambda na qual a função não pode analisar os dados do evento fornecidos na chamada.

Example Erro de função

```
{  
    "errorMessage" : "Error parsing event data.",  
    "errorType" : "InvalidEventDataException",  
    "stackTrace": [ ]  
}
```

Corpo da resposta

- **Lambda-Extension-Identifier:** identificador de agente exclusivo (string UUID).

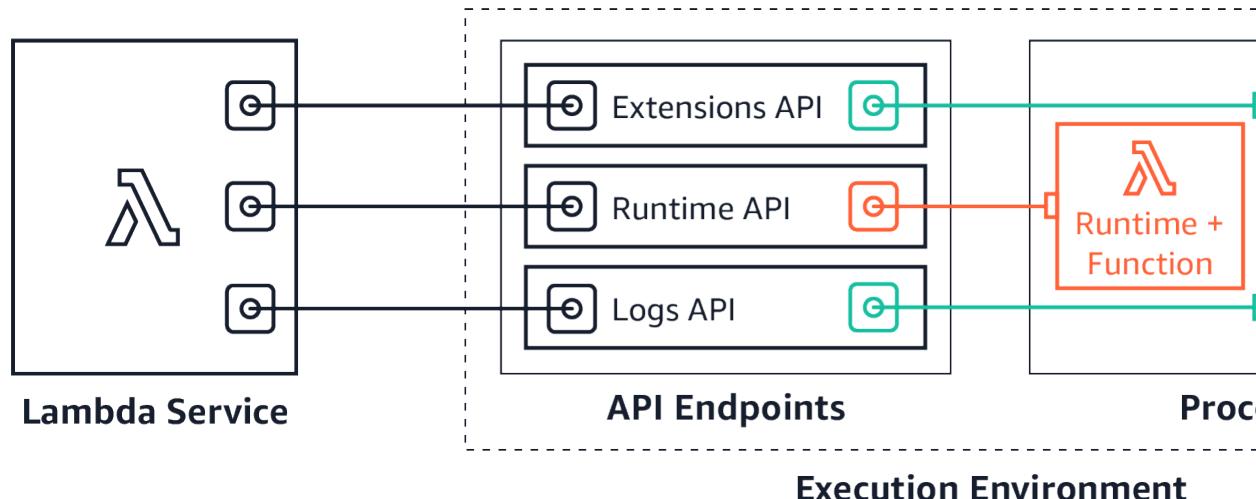
Códigos de resposta

- 202: aceito
- 400: solicitação inválida
- 403: proibido
- 500: erro de contêiner. Estado não recuperável. A extensão deve sair imediatamente.

API Lambda Logs

O Lambda captura automaticamente os logs de tempo de execução e os transmite para o Amazon CloudWatch. Essa transmissão de log contém os logs que seu código de função e extensões geram, e também os gerados pelo Lambda como parte da chamada de função.

[Extensões do Lambda \(p. 228\)](#) pode usar a API RLambda time Logs para assinar fluxos de log diretamente do Lambda [ambiente de execução \(p. 219\)](#). O Lambda transmite os logs para a extensão e a extensão pode processar, filtrar e enviar os logs para qualquer destino preferido.



A API Logs permite que as extensões se inscrevam em três fluxos de logs diferentes:

- Logs de função que a função do Lambda gera e grava em `stdout` ou `stderr`.
- Registros de extensão que o código de extensão gera.
- Logs da plataforma do Lambda que registram eventos e erros relacionados a invocações e extensões.

Note

O Lambda envia todos os logs para o CloudWatch, mesmo quando uma extensão se inscreve em uma ou mais transmissões de log.

Tópicos

- [Assinando para receber registros \(p. 243\)](#)
- [Uso de memória \(p. 243\)](#)
- [protocolos de destino \(p. 243\)](#)
- [Configuração de buffer \(p. 243\)](#)
- [Exemplo de assinatura \(p. 244\)](#)
- [Código de exemplo para Logs API \(p. 244\)](#)
- [Referência da API Logs \(p. 245\)](#)
- [Mensagens de log \(p. 246\)](#)

Assinando para receber registros

Uma extensão do Lambda pode se inscrever para receber registros enviando uma solicitação de assinatura para a API Logs.

Para se inscrever para receber registros, você precisa do identificador de extensão (`Lambda-Extension-Identifier`). Primeiro [registre a extensão \(p. 237\)](#) para receber o identificador de extensão. Em seguida, inscreva-se na API Logs durante a [inicialização \(p. 220\)](#). Após a conclusão da fase de inicialização, o Lambda não processa solicitações de assinatura.

Note

A assinatura da API de registros é idempotente. As solicitações de assinatura duplicadas não resultam em assinaturas duplicadas.

Uso de memória

O uso da memória aumenta linearmente à medida que o número de assinantes aumenta. As assinaturas consomem recursos de memória porque cada assinatura abre um novo buffer de memória para armazenar os logs. Para ajudar a otimizar o uso da memória, você pode ajustar a [configuração de buffer \(p. 243\)](#). O uso da memória buffer conta para o consumo geral de memória no ambiente de execução.

protocolos de destino

Você pode escolher um dos seguintes protocolos para receber os logs:

1. HTTP (recomendado): o Lambda entrega os logs a um endpoint HTTP local (`http://sandbox.localdomain:${PORT}/${PATH}`) como uma matriz de registros no formato JSON. O parâmetro `$PATH` é opcional. Observe que somente HTTP é suportado, não HTTPS. Você pode optar por receber registros através PUT ou POST.
2. TCP: o Lambda entrega logs a uma porta TCP no formato [JSON delimitado por nova linha \(NDJSON\)](#).

Recomendamos usar HTTP em vez de TCP. Com o TCP, a plataforma do Lambda não reconhece que os logs são entregues à camada da aplicação. Portanto, você poderá perder os logs se a extensão falhar. O HTTP não compartilha essa limitação.

Também recomendamos configurar o ouvinte HTTP local ou a porta TCP antes de se inscrever para receber logs. Durante a configuração, observe o seguinte:

- O Lambda envia logs somente para destinos que estão dentro do ambiente de execução.
- O Lambda tenta novamente enviar os logs (com recuo) se não houver listener ou se a solicitação POST ou PUT resultar em erro. Se o assinante do log falhar, ele continuará a receber logs depois que o Lambda reiniciar o ambiente de execução.
- Lambda reserva porto 9001. Não há outras restrições ou recomendações de número de porta.

Configuração de buffer

O Lambda pode armazenar logs de buffer e entregá-los ao assinante. Você pode configurar esse comportamento na solicitação de assinatura especificando os campos opcionais a seguir. Observe que o Lambda usa o valor padrão para qualquer campo que você não especificar.

- `timeoutMs`: o tempo máximo (em milissegundos) para colocar um lote em buffer. Padrão: 1.000. Mínimo: 25. Máximo: 30.000.

- maxBytes: o tamanho máximo (em bytes) dos logs para colocar em buffer na memória. Padrão: 262.144. Mínimo: 262.144. Máximo: 1.048.576.
- maxItems: o número máximo de eventos a serem colocados em buffer na memória. Padrão: 10.000. Mínimo: 1.000. Máximo: 10.000.

Durante a configuração de buffer, observe os seguintes pontos:

- O Lambda libera os logs se alguma das transmissões de entrada estiver fechada, por exemplo, se o tempo de execução falhar.
- Cada assinante pode especificar uma configuração de buffer diferente durante a solicitação de assinatura.
- Considere o tamanho do buffer que você precisa para ler os dados. Espere receber cargas úteis tão grandes quanto $2 * \text{maxBytes} + \text{metadata}$, onde `maxBytes` é configurado na solicitação de assinatura. Por exemplo, o Lambda adiciona os seguintes bytes de metadados a cada registro:

```
{  
  "time": "2020-08-20T12:31:32.123Z",  
  "type": "function",  
  "record": "Hello World"  
}
```

- Se o assinante não puder processar logs de entrada com rapidez suficiente, o Lambda pode descartar logs para manter a utilização da memória limitada. Para indicar o número de registros descartados, o Lambda envia um log `platform.logsDropped`.

Exemplo de assinatura

O exemplo a seguir mostra uma solicitação para se inscrever na plataforma e logs de funções.

```
PUT http://${AWS_LAMBDA_RUNTIME_API}/2020-08-15/logs/ HTTP/1.1  
{ "schemaVersion": "2020-08-15",  
  "types": [  
    "platform",  
    "function"  
  ],  
  "buffering": {  
    "maxItems": 1000,  
    "maxBytes": 10240,  
    "timeoutMs": 100  
  },  
  "destination": {  
    "protocol": "HTTP",  
    "URI": "http://sandbox.localdomain:8080/lambda_logs"  
  }  
}
```

Se o pedido for bem-sucedido, o assinante receberá uma resposta de sucesso HTTP 200.

```
HTTP/1.1 200 OK  
"OK"
```

Código de exemplo para Logs API

Para obter exemplos de código mostrando como enviar logs para um destino personalizado, consulte [Using AWS Lambda extensions to send logs to custom destinations](#) no AWS Compute Blog.

Para exemplos de código Python e Go que mostram como desenvolver uma extensão básica do Lambda e assinar a API Logs, consulte [Extensões do AWS Lambda](#) no repositório de exemplos da AWS no GitHub. Para obter mais informações sobre a criação de uma extensão do Lambda, consulte [the section called “API de extensões” \(p. 228\)](#).

Referência da API Logs

É possível recuperar o endpoint da API Logs da variável de ambiente `AWS_LAMBDA_RUNTIME_API`. Para enviar uma solicitação de API, use o prefixo `2020-08-15/` antes do caminho da API. Por exemplo:

```
http://${AWS_LAMBDA_RUNTIME_API}/2020-08-15/logs/
```

A especificação OpenAPI para a versão da API de logs, 2020-08-15, está disponível aqui: [logs-api-request.zip](#)

Subscribe

Para se inscrever em um ou mais transmissões de log disponíveis no ambiente de execução do Lambda, as extensões enviam uma solicitação de assinatura da API.

Caminho – `/logs`

Método – PUT

Body parameters (Parâmetros do corpo)

`destination` – Consulte [the section called “protocolos de destino” \(p. 243\)](#). Obrigatório: sim. Tipo: strings.

`buffering` – Consulte [the section called “Configuração de buffer” \(p. 243\)](#). Obrigatório: não Tipo: strings.

`types`: uma matriz dos tipos de logs a serem recebidos. Obrigatório: sim. Tipo: matriz de strings. Valores válidos: “plataforma”, “função”, “extensão”.

`schemaVersion`: obrigatório: não Valor padrão: “2020-08-15”. Defina como “2021-03-18” para que a extensão receba [platform.runtimeDone \(p. 248\)](#) mensagens.

Parâmetros de resposta

As especificações OpenAPI para as respostas de assinatura, versão 2020-08-15, estão disponíveis para HTTP e TCP:

- HTTP: [logs-api-http-response.zip](#)
- TCP: [logs-api-tcp-response.zip](#)

Códigos de resposta

- 200: solicitação concluída com êxito
- 202: solicitação aceita. Resposta a uma solicitação de assinatura durante testes locais.
- 4XX: solicitação inválida
- 500: erro do serviço

Se o pedido for bem-sucedido, o assinante receberá uma resposta de sucesso HTTP 200.

```
HTTP/1.1 200 OK
"OK"
```

Se a solicitação falhar, o assinante receberá uma resposta de erro. Por exemplo:

```
HTTP/1.1 400 OK
{
    "errorType": "Logs.ValidationError",
    "errorMessage": "URI port is not provided; types should not be empty"
}
```

Mensagens de log

A API Logs permite que as extensões se inscrevam em três fluxos de logs diferentes:

- Função: logs que a função do Lambda gera e grava em `stdout` ou `stderr`.
- Extensão: logs gerados pelo código da extensão.
- Plataforma: logs gerados pela plataforma de tempo de execução, que registram eventos e erros relacionados a invocações e extensões.

Tópicos

- [Registros de função \(p. 246\)](#)
- [Logs de extensões \(p. 246\)](#)
- [Logs de plataformas \(p. 246\)](#)

Registros de função

A função e as extensões internas do Lambda geram logs de funções e os gravam em `stdout` ou `stderr`.

O exemplo a seguir mostra o formato de uma mensagem de log de função. { "time": "2020-08-20T12:31:32.123Z", "type": "function", "record": "ERROR encountered. Rastreamento da pilha:\n\\my-function (line 10)\n" }

Logs de extensões

As extensões podem gerar logs de extensões. O formato do log é o mesmo que para um log de funções.

Logs de plataformas

O Lambda gera mensagens de log para eventos de plataforma como `platform.end`, `platform.start` e `platform.fault`.

Opcionalmente, você pode se inscrever na versão 2021-03-18 do esquema da API de logs, que inclui a mensagem de log `platform.runtimeDone`.

Exemplo de mensagens de log da plataforma

O exemplo a seguir mostra o início da plataforma e os logs finais da plataforma. Esses logs indicam a hora de início da invocação e o horário de término para a invocação que o `requestID` especifica.

```
{
```

```
{  
    "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"}  
}
```

O log de relatório da plataforma inclui métricas sobre a invocação que o requestId especifica. O campo initDurationMs é incluído no log somente se a invocação incluir uma partida a frio. Se o rastreamento do AWS X-Ray estiver ativo, o log incluirá metadados do X-Ray. O exemplo a seguir mostra um log de relatório de plataforma para uma invocação que incluiu uma partida a frio.

```
{  
    "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  
        }  
    }  
}
```

O log da plataforma captura erros de ambiente de execução ou tempo de execução. O exemplo a seguir mostra uma mensagem de log de falha da plataforma.

```
{  
    "time": "2020-08-20T12:31:32.123Z",  
    "type": "platform.fault",  
    "record": "RequestId: d783b35e-a91d-4251-af17-035953428a2c Process exited before  
completing request"  
}
```

O Lambda gera um log de extensão de plataforma quando uma extensão se registra com a API Extensions. O exemplo a seguir mostra uma mensagem de extensão da plataforma.

```
{  
    "time": "2020-08-20T12:31:32.123Z",  
    "type": "platform.extension",  
    "record": {"name": "Foo.bar",  
        "state": "Ready",  
        "events": ["INVOKE", "SHUTDOWN"]  
    }  
}
```

O Lambda gera um log de assinatura de logs de plataforma quando uma extensão se inscreve na API Logs. O exemplo a seguir mostra uma mensagem de assinatura de logs.

```
{  
    "time": "2020-08-20T12:31:32.123Z",  
    "type": "platform.logsSubscription",  
    "record": {"name": "Foo.bar",  
        "state": "Subscribed",  
        "types": ["function", "platform"],  
    }  
}
```

```
}
```

O Lambda gera um log descartado de logs de plataforma quando uma extensão não é capaz de processar o número de logs que está recebendo. O exemplo a seguir mostra uma mensagem de log `platform.logsDropped`.

```
{
    "time": "2020-08-20T12:31:32.123Z",
    "type": "platform.logsDropped",
    "record": {"reason": "Consumer seems to have fallen behind as it has not acknowledged receipt of logs.",
               "droppedRecords": 123,
               "droppedBytes": 12345
    }
}
```

Mensagens `runtimedone` da plataforma

Se você definir a versão do esquema como “2021-03-18” na solicitação de assinatura, o Lambda enviará uma mensagem `platform.runtimedone` após a conclusão da invocação da função com êxito ou com um erro. A extensão pode usar esta mensagem para interromper toda a coleção de telemetria para esta invocação de função.

A especificação OpenAPI para o tipo de evento do log no esquema versão 2021-03-18 está disponível aqui: [schema-2021-03-18.zip](#)

O Lambda gera a mensagem de log `Next` quando o tempo de execução envia uma solicitação de API do tempo de execução `platform.runtimedone` ou `Error`. O `platform.runtimedone` log informa os consumidores da API Logs que a invocação de função foi concluída. As extensões podem usar essas informações para decidir quando enviar toda a telemetria coletada durante essa invocação.

Examples

O Lambda envia a mensagem `platform.runtimedone` depois que o tempo de execução envia a solicitação `NEXT`, quando a invocação da função for concluída. Os exemplos a seguir mostram mensagens para cada um dos valores de status: sucesso, falha e tempo limite.

Example Exemplo de mensagem de sucesso

```
{
    "time": "2021-02-04T20:00:05.123Z",
    "type": "platform.runtimedone",
    "record": {
        "requestId": "6f7f0961f83442118a7af6fe80b88",
        "status": "success"
    }
}
```

Example Exemplo de mensagem de falha

```
{
    "time": "2021-02-04T20:00:05.123Z",
    "type": "platform.runtimedone",
    "record": {
        "requestId": "6f7f0961f83442118a7af6fe80b88",
        "status": "failure"
    }
}
```

Example Exemplo de mensagem de tempo limite

```
{  
    "time": "2021-02-04T20:00:05.123Z",  
    "type": "platform.runtimeDone",  
    "record": {  
        "requestId": "6f7f0961f83442118a7af6fe80b88",  
        "status": "timeout"  
    }  
}
```

Modificar o ambiente de execução

É possível usar [extensões internas \(p. 178\)](#) para modificar o processo de tempo de execução. As extensões internas não são processos separados. Elas são executadas como parte do processo de tempo de execução.

O Lambda fornece específicos do idioma [Variáveis de ambiente \(p. 99\)](#) que você pode definir para adicionar opções e ferramentas ao tempo de execução. O Lambda também fornece [scripts wrapper \(p. 252\)](#), que permitem ao Lambda delegar a inicialização do tempo de execução ao seu script. Você pode criar um script wrapper para personalizar o comportamento de inicialização do tempo de execução.

Variáveis de ambiente específicas de linguagem

O Lambda é compatível com formas somente de configuração para permitir que o código seja pré-carregado durante a inicialização da função por meio das seguintes variáveis de ambiente específicas do idioma:

- **JAVA_TOOL_OPTIONS**: no Java 11 e Java 8 (`java8.a12`), o Lambda é compatível com essa variável de ambiente para definir variáveis da linha de comando adicionais no Lambda. Essa variável de ambiente permite especificar a inicialização de ferramentas, especificamente o lançamento de agentes de linguagem de programação nativa ou Java usando as opções `agentlib` ou `javaagent`.
- **NODE_OPTIONS**: no Node.js 10x e posterior, o Lambda é compatível com essa variável de ambiente.
- **DOTNET_STARTUP_HOOKS**: no .NET Core 3.1 e superior, essa variável de ambiente especifica um caminho para um assembly (dll) que o Lambda pode usar.

Usar variáveis de ambiente específicas de linguagem é a maneira preferida de definir propriedades de inicialização.

Exemplo: interceptar invocações do Lambda com `javaagent`

A Java Virtual Machine (JVM) tenta localizar a classe especificada com o parâmetro `javaagent` para a JVM e invocar seu método `premain` antes do ponto de entrada da aplicação.

O exemplo a seguir usa [Byte Buddy](#), uma biblioteca para criar e modificar classes Java durante o tempo de execução de uma aplicação Java sem a ajuda de um compilador. Byte Buddy oferece uma API adicional para gerar agentes Java. Neste exemplo, a classe `Agent` intercepta todas as chamadas do método `handleRequest` feitas para a classe [RequestStreamHandler](#). Essa classe é usada internamente no tempo de execução para envolver as invocações do manipulador.

```
import com.amazonaws.services.lambda.runtime.RequestStreamHandler;
import net.bytebuddy.agent.builder.AgentBuilder;
import net.bytebuddy.asm.Advice;
import net.bytebuddy.matcher.ElementMatchers;

import java.lang.instrument.Instrumentation;

public class Agent {

    public static void premain(String agentArgs, Instrumentation inst) {
        new AgentBuilder.Default()
            .with(new AgentBuilder.InitializationStrategy.SelfInjection.Eager())
            .type(ElementMatchers.isSubTypeOf(RequestStreamHandler.class))
            .transform((builder, typeDescription, classLoader, module) -> builder
                .method(ElementMatchers.nameContains("handleRequest"))

```

```
        .intercept(Advice.to(TimerAdvice.class)))
        .installOn(inst);
    }
}
```

O agente no exemplo anterior usa o método `TimerAdvice`. `TimerAdvice` mede quantos milissegundos são gastos com a chamada do método e registra a hora e os detalhes do método, como nome e argumentos transmitidos.

```
import static net.bytebuddy.asm.Advice.AllArguments;
import static net.bytebuddy.asm.Advice.Enter;
import static net.bytebuddy.asm.Advice.OnMethodEnter;
import static net.bytebuddy.asm.Advice.OnMethodExit;
import static net.bytebuddy.asm.Advice.Origin;

public class TimerAdvice {

    @OnMethodEnter
    static long enter() {
        return System.currentTimeMillis();
    }

    @OnMethodExit
    static void exit(@Origin String method, @Enter long start, @AllArguments Object[] args)
    {
        StringBuilder sb = new StringBuilder();
        for (Object arg : args) {
            sb.append(arg);
            sb.append(", ");
        }
        System.out.println(method + " method with args: " + sb.toString() + " took " +
        (System.currentTimeMillis() - start) + " milliseconds");
    }
}
```

O método `TimerAdvice` acima tem as seguintes dependências.

```
*'com.amazonaws'*, *name*: *'aws-lambda-java-core'*, *version*: *'1.2.1'*  
*'net.bytebuddy'*, *name*: *'byte-buddy-dep'*, *version*: *'1.10.14'*  
*'net.bytebuddy'*, *name*: *'byte-buddy-agent'*, *version*: *'1.10.14'*
```

Depois de criar uma camada que contém o JAR do agente, você pode passar o nome do JAR para a JVM do tempo de execução definindo uma variável de ambiente.

```
JAVA_TOOL_OPTIONS=-javaagent:"/opt/ExampleAgent-0.0.jar"
```

Depois de invocar a função com `{key=lambdaInput}`, você poderá encontrar a seguinte linha nos logs:

```
public java.lang.Object lambdainternal.EventHandlerLoader
$PojoMethodRequestHandler.handleRequest
(java.lang.Object,com.amazonaws.services.lambda.runtime.Context) method with args:
{key=lambdaInput}, lambdainternal.api.LambdaContext@4d9d1b69, took 106 milliseconds
```

Exemplo: adicionar um gancho de desligamento ao processo de tempo de execução da JVM

Quando uma extensão é registrada durante um evento Shutdown, o processo de tempo de execução obtém até 500 ms para lidar com o desligamento normal. Você pode se conectar ao processo de tempo de execução e, quando a JVM inicia seu processo de desligamento, ela inicia todos os ganchos registrados. Para registrar um gancho de desligamento, você deve [fazer o registro como uma extensão \(p. 237\)](#). Você não precisa se registrar explicitamente para o evento Shutdown, pois ele é enviado automaticamente para o tempo de execução.

```
import java.lang.instrument.Instrumentation;

public class Agent {

    public static void premain(String agentArgs, Instrumentation inst) {
        // Register the extension.
        //
        ...

        // Register the shutdown hook
        addShutdownHook();
    }

    private static void addShutdownHook() {
        // Shutdown hooks get up to 500 ms to handle graceful shutdown before the runtime is
        // terminated.
        //
        // You can use this time to egress any remaining telemetry, close open database
        // connections, etc.
        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            // Inside the shutdown hook's thread we can perform any remaining task which
            // needs to be done.
            }));
    }
}
```

Exemplo: recuperar o InvokedFunctionArn

```
@OnMethodEnter
static long enter() {
    String invokedFunctionArn = null;
    for (Object arg : args) {
        if (arg instanceof Context) {
            Context context = (Context) arg;
            invokedFunctionArn = context.getInvokedFunctionArn();
        }
    }
}
```

Scripts wrapper

Você pode criar um script wrapper para personalizar o comportamento de inicialização do tempo de execução da função do Lambda. Um script wrapper permite que você defina parâmetros de configuração que não podem ser definidos por meio de variáveis de ambiente específicas de linguagem.

Note

As chamadas podem falhar se o script wrapper não iniciar com êxito o processo de tempo de execução.

Os seguintes [tempos de execução do Lambda \(p. 214\)](#) são compatíveis com scripts wrapper:

- Node.js 14.x
- Node.js 12.x
- Node.js 10.x
- Python 3.9
- Python 3.8
- Ruby 2.7
- Java 11
- Java 8 (java8.al2)
- .NET Core 3.1

Quando você usa um script wrapper para sua função, o Lambda inicia o tempo de execução usando seu script. O Lambda envia ao script o caminho para o intérprete e todos os argumentos originais para a inicialização padrão do tempo de execução. O script pode estender ou transformar o comportamento de inicialização do programa. Por exemplo, o script pode injetar e alterar argumentos, definir variáveis de ambiente ou capturar métricas, erros e outras informações de diagnóstico.

Você especifica o script definindo o valor da variável de ambiente `AWS_LAMBDA_EXEC_WRAPPER` como o caminho do sistema de arquivos de um binário ou script executável.

Exemplo: criar e usar um script wrapper com o Python 3.8

No exemplo a seguir, você cria um script wrapper para iniciar o interpretador Python com a opção `-X importtime`. Quando você executa a função, o Lambda gera uma entrada de log para mostrar a duração de cada importação.

Como criar e usar um script wrapper com o Python 3.8

1. Para criar o script wrapper, cole o seguinte código em um arquivo chamado `importtime_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[@]}"
```

2. Para conceder permissões executáveis ao script, insira `chmod +x importtime_wrapper` pela linha de comando.
3. Implante o script como uma [camada do Lambda \(p. 85\)](#).
4. Crie uma função usando o console do Lambda.

- a. Abra o [console do lambda](#).
 - b. Escolha Create function.
 - c. Em Basic information (Informações básicas), em Function name (Nome da função), insira **wrapper-test-function**.
 - d. Em Runtime (Tempo de execução), selecione Python 3.8.
 - e. Escolha Create function.
5. Adicione a camada à função.
- a. Escolha sua função e, em seguida, escolha Código se ela ainda não estiver selecionada.
 - b. Escolha Add a layer.
 - c. Em Choose a layer (Escolher uma camada), selecione o Name (Nome) e a Version (Versão) da camada compatível criada anteriormente.
 - d. Escolha Adicionar.
6. Adicione o código e a variável de ambiente à função.
- a. No [editor de código \(p. 40\)](#) da função, cole o seguinte código de função:

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

- b. Escolha Save (Salvar).
 - c. Em Environment variables (Variáveis de ambiente), selecione Edit (Editar).
 - d. Escolha Add environment variable (Adicionar variável de ambiente).
 - e. Em Key (Chave), insira AWS_LAMBDA_EXEC_WRAPPER.
 - f. Em Value (Valor), insira /opt/importtime_wrapper.
 - g. Escolha Save (Salvar).
7. Para executar a função, selecione Test (Testar).

Como o script wrapper iniciou o interpretador Python com a opção `-X importtime`, os logs mostram o tempo necessário para cada importação. Por exemplo:

```
...
2020-06-30T18:48:46.780+01:00 import time: 213 | 213 | simplejson
2020-06-30T18:48:46.780+01:00 import time: 50 | 263 | simplejson.raw_json
...
```

Tempos de execução personalizados do AWS Lambda

Você pode implementar um tempo de execução do AWS Lambda em qualquer linguagem de programação. Tempo de execução é um programa que executa um método de manipulador da função do Lambda quando a função é invocada. Você pode incluir um tempo de execução em seu pacote de implantação da função na forma de um arquivo executável chamado `bootstrap`.

Um tempo de execução é responsável por executar o código de configuração da função, lendo o nome do manipulador de uma variável de ambiente e lendo eventos de invocação da API de tempo de execução do Lambda. O tempo de execução transmite os dados de eventos para o manipulador de funções e publica a resposta do manipulador de volta no Lambda.

Seu tempo de execução personalizado é executado no [ambiente de execução \(p. 214\)](#) padrão do Lambda. Pode ser um script de shell, um script em uma linguagem incluída no Amazon Linux ou um arquivo executável binário compilado no Amazon Linux.

Para começar a usar tempos de execução personalizados, consulte [Tutorial: publicar um tempo de execução personalizado \(p. 258\)](#). Você também pode explorar um tempo de execução personalizado implementado em C++ em [awslabs/aws-lambda-cpp](#) no GitHub.

Tópicos

- [Usar um tempo de execução personalizado \(p. 255\)](#)
- [Criar um tempo de execução personalizado \(p. 255\)](#)

Usar um tempo de execução personalizado

Para usar um tempo de execução personalizado, defina o tempo de execução de sua função como `provided`. O tempo de execução pode ser incluído no pacote de implantação da sua função ou em uma [camada \(p. 85\)](#).

Example function.zip

```
.\n### bootstrap\n### function.sh
```

Se houver um arquivo denominado `bootstrap` no pacote de implantação, o Lambda o executará. Se não, o Lambda procurará um tempo de execução nas camadas da função. Se o arquivo de bootstrap não for encontrado ou não for executável, sua função retornará um erro na invocação.

Criar um tempo de execução personalizado

O ponto de entrada de um tempo de execução personalizado é um arquivo executável denominado `bootstrap`. O arquivo de bootstrap pode ser o tempo de execução ou ele pode invocar outro arquivo que criará o tempo de execução. O exemplo a seguir usa um pacote de versão do Node.js para executar um tempo de execução JavaScript em um arquivo separado denominado `runtime.js`.

Example bootstrap

```
#!/bin/sh
```

```
cd $LAMBDA_TASK_ROOT
./node-v11.1.0-linux-x64/bin/node runtime.js
```

Seu código de tempo de execução é responsável por executar algumas tarefas de inicialização. Em seguida, ele processa os eventos de invocação em um loop até ser encerrado. As tarefas de inicialização são executadas uma vez [por instância da função \(p. 219\)](#) a fim de preparar o ambiente para processar invocações.

Tarefas de inicialização

- Recuperar configurações: leia variáveis de ambiente para obter detalhes sobre a função e o ambiente.
 - `_HANDLER`: a localização do manipulador, da configuração da função. O formato padrão é `file.method`, em que `file` é o nome do arquivo sem uma extensão, e `method` é o nome de um método ou da função que é definido no arquivo.
 - `LAMBDA_TASK_ROOT`: o diretório que contém o código da função.
 - `AWS_LAMBDA_RUNTIME_API`: o host e a porta da API de tempo de execução.

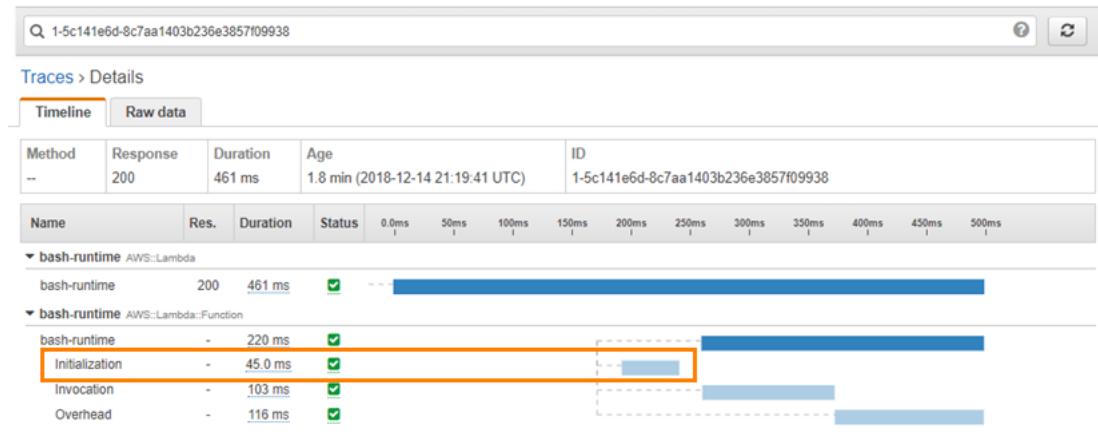
Consulte [Variáveis de ambiente com tempo de execução definido \(p. 102\)](#) para obter uma lista completa das variáveis disponíveis.

- Inicializar a função: carregar o arquivo do manipulador e executar qualquer código global ou estático que ele contenha. As funções devem criar recursos estáticos como clientes SDK e conexões de banco de dados uma vez e reutilizá-los para várias invocações.
- Lidar com erros: se ocorrer um erro, chame a API [erro de inicialização \(p. 225\)](#) e saia imediatamente.

A inicialização é considerada no tempo de execução e tempo limite cobrados. Quando uma execução aciona a inicialização de uma nova instância da função, é possível ver o tempo de inicialização nos logs e no [rastreamento do AWS X-Ray \(p. 477\)](#).

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
```



Enquanto ela é executada, um tempo de execução usa a [interface de tempo de execução do Lambda \(p. 224\)](#) para gerenciar eventos de entrada e relatar erros. Depois de executar as tarefas de inicialização, o tempo de execução processa eventos de entrada em um loop. No código de tempo de execução, execute as seguintes etapas em ordem.

Tarefas de processamento

- Obter um evento: chame a API [próxima invocação \(p. 224\)](#) para obter o próximo evento. O corpo da resposta contém os dados do evento. Os cabeçalhos da resposta contêm o ID da solicitação e outras informações.
- Propagar o cabeçalho de rastreamento: obtenha o cabeçalho de rastreamento do X-Ray do cabeçalho `Lambda-Runtime-Trace-Id` na resposta da API. Defina a variável de ambiente `_X_AMZN_TRACE_ID` localmente com o mesmo valor. O X-Ray SDK usa esse valor para conectar dados de rastreamento entre serviços.
- Criar um contexto de objeto: crie um objeto com informações de contexto de variáveis do ambiente e cabeçalhos na resposta da API.
- Invocar o manipulador de funções: transmita o evento e o objeto de contexto para o manipulador.
- Processar a resposta: chame a API de [resposta de invocação \(p. 225\)](#) para publicar a resposta do manipulador.
- Processar erros: se ocorrer um erro, chame a API [erro de invocação \(p. 226\)](#).
- Limpeza: libere recursos não utilizados, envie dados para outros serviços ou execute tarefas adicionais antes de receber o próximo evento.

Você pode incluir o tempo de execução no pacote de implantação da sua função ou distribuir o tempo de execução separadamente em uma camada de função. Para ver uma demonstração de exemplo, consulte [Tutorial: publicar um tempo de execução personalizado \(p. 258\)](#).

Tutorial: publicar um tempo de execução personalizado

Neste tutorial, você criará uma função do Lambda com um tempo de execução personalizado. Você começa incluindo o tempo de execução no pacote de implantação da função. Em seguida, você o migra para uma camada gerenciada de forma independente da função. Por fim, você compartilha a camada de tempo de execução com o mundo atualizando sua política de permissões com base em recursos.

Prerequisites

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em [Conceitos básicos do Lambda \(p. 9\)](#) para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, você precisa de um terminal de linha de comando ou de um shell para executar os comandos. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

Você deve ver a saída a seguir:

```
aws-cli/2.0.57 Python/3.7.4 Darwin/19.6.0 exe/x86_64
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

Você precisa de uma função do IAM para criar uma função do Lambda. A função precisa de permissão para enviar logs ao CloudWatch Logs e acessar o AWS que sua função usa. Se você ainda não tiver uma função para o desenvolvimento de funções, crie uma agora.

Para criar uma função de execução

1. Abra a [página Roles](#) (Funções) no console do IAM.
2. Selecione Create role.
3. Crie uma função com as seguintes propriedades.
 - Entidade confiável-Lambda.
 - Permissions (Permissões): AWSLambdaBasicExecutionRole.
 - Nome da função – **lambda-role**.

A política AWSLambdaBasicExecutionRole tem as permissões necessárias para a função gravar logs no CloudWatch Logs.

Criar uma função do

Criar uma função do Lambda com um tempo de execução personalizado. Este exemplo inclui dois arquivos, um arquivo bootstrap de tempo de execução e um manipulador de funções. Ambos são implementados em Bash.

O tempo de execução carrega um script de função do pacote de implantação. Ele usa duas variáveis para localizar o script. `LAMBDA_TASK_ROOT` informa onde o pacote foi extraído e `_HANDLER` inclui o nome do script.

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 "$HEADERS" -X GET "http://${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 -X POST "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/$REQUEST_ID/response" -d "$RESPONSE"
done
```

Depois de carregar o script, o tempo de execução processa eventos em um loop. Ele usa a API de tempo de execução para recuperar um evento de invocação do Lambda, transmite o evento para o handler e publica a resposta de volta no Lambda. Para obter o ID de solicitação, o tempo de execução salva os cabeçalhos na resposta da API em um arquivo temporário e lê o cabeçalho `Lambda-Runtime-Aws-Request-Id` do arquivo.

Note

Os tempo de execução têm responsabilidades adicionais, incluindo o processamento de erros e o fornecimento de informações de contexto para o manipulador. Para obter mais detalhes, consulte [Criar um tempo de execução personalizado \(p. 255\)](#).

O script define uma função do manipulador que usa dados de eventos, os registra no `stderr` e os retorna.

Example function.sh

```
function handler () {
    EVENT_DATA=$1
    echo "$EVENT_DATA" 1>&2;
    RESPONSE="Echoing request: '$EVENT_DATA'

    echo $RESPONSE
}
```

Salve os arquivos em um diretório do projeto denominado `runtime-tutorial`.

```
runtime-tutorial
```

```
# bootstrap
# function.sh
```

Torne os arquivos executáveis e os adicione a um arquivo .zip.

```
runtime-tutorial$ chmod 755 function.sh bootstrap
runtime-tutorial$ zip function.zip function.sh bootstrap
  adding: function.sh (deflated 24%)
  adding: bootstrap (deflated 39%)
```

Crie uma função chamada bash-runtime.

```
runtime-tutorial$ aws lambda create-function --function-name bash-runtime \
--zip-file fileb://function.zip --handler function.handler --runtime provided \
--role arn:aws:iam::123456789012:role/lambda-role
{
    "FunctionName": "bash-runtime",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:bash-runtime",
    "Runtime": "provided",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Handler": "function.handler",
    "CodeSha256": "mv/xRv84LPCxdpcbKvmwuuFzwo7sLwUO1VxcUv3wKlM=",
    "Version": "$LATEST",
    "TracingConfig": {
        "Mode": "PassThrough"
    },
    "RevisionId": "2e1d51b0-6144-4763-8e5c-7d5672a01713",
    ...
}
```

Invoque a função e verifique a resposta.

```
runtime-tutorial$ aws lambda invoke --function-name bash-runtime --payload
'{"text":"Hello"}' response.txt --cli-binary-format raw-in-base64-out
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
runtime-tutorial$ cat response.txt
Echoing request: '{"text":"Hello"}'
```

Criar uma camada

Para separar o código do tempo de execução do código da função, crie uma camada que contenha apenas o tempo de execução. As camadas permitem que você desenvolva dependências da sua função de forma independente e podem reduzir o uso de armazenamento quando você usa a mesma camada com várias funções.

Crie um arquivo de camada que contenha o arquivo bootstrap.

```
runtime-tutorial$ zip runtime.zip bootstrap
  adding: bootstrap (deflated 39%)
```

Crie uma camada com o comando publish-layer-version.

```
runtime-tutorial$ aws lambda publish-layer-version --layer-name bash-runtime --zip-file
fileb://runtime.zip
```

```
{  
    "Content": {  
        "Location": "https://awslambda-us-west-2-layers.s3.us-west-2.amazonaws.com/  
snapshots/123456789012/bash-runtime-018c209b...",  
        "CodeSha256": "bXVLhHi+D3H1QbDARUVPrDwlC7bssPxySQqt1QZqusE=",  
        "CodeSize": 584,  
        "UncompressedCodeSize": 0  
    },  
    "LayerArn": "arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime",  
    "LayerVersionArn": "arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime:1",  
    "Description": "",  
    "CreatedDate": "2018-11-28T07:49:14.476+0000",  
    "Version": 1  
}
```

Isso cria a primeira versão da camada.

Atualizar a função

Para usar a camada de tempo de execução com a função, configure a função para usar a camada e remova o código de tempo de execução da função.

Atualize a configuração de função para extrair na camada.

```
runtime-tutorial$ aws lambda update-function-configuration --function-name bash-runtime \  
--layers arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime:1  
{  
    "FunctionName": "bash-runtime",  
    "Layers": [  
        {  
            "Arn": "arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime:1",  
            "CodeSize": 584,  
            "UncompressedCodeSize": 679  
        }  
    ]  
    ...  
}
```

Isso adiciona o tempo de execução à função no /opt directory. O Lambda usa esse tempo de execução, mas somente se você o remover do pacote de implantação da função. Atualize o código da função para incluir apenas o script do manipulador.

```
runtime-tutorial$ zip function-only.zip function.sh  
adding: function.sh (deflated 24%)  
runtime-tutorial$ aws lambda update-function-code --function-name bash-runtime --zip-file  
fileb://function-only.zip  
{  
    "FunctionName": "bash-runtime",  
    "CodeSize": 270,  
    "Layers": [  
        {  
            "Arn": "arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime:7",  
            "CodeSize": 584,  
            "UncompressedCodeSize": 679  
        }  
    ]  
    ...  
}
```

Invoque a função para verificar se ela funciona com a camada de tempo de execução.

```
runtime-tutorial$ aws lambda invoke --function-name bash-runtime --payload
'{"text":"Hello"}' response.txt
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
runtime-tutorial$ cat response.txt
Echoing request: '{"text":"Hello"}'
```

Atualizar o tempo de execução

Para registrar informações sobre o ambiente de execução, atualize o script de tempo de execução para gerar variáveis de ambiente.

Example bootstrap

```
#!/bin/sh

set -euo pipefail

echo "## Environment variables:"
env

# Initialization - load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"
...
```

Crie uma segunda versão da camada com o novo código.

```
runtime-tutorial$ zip runtime.zip bootstrap
updating: bootstrap (deflated 39%)
runtime-tutorial$ aws lambda publish-layer-version --layer-name bash-runtime --zip-file
file:///runtime.zip
```

Configure a função para usar a nova versão da camada.

```
runtime-tutorial$ aws lambda update-function-configuration --function-name bash-runtime \
--layers arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime:2
```

Compartilhar a camada

Adicione uma instrução de permissão à sua camada de tempo de execução para compartilhá-la com outras contas.

```
runtime-tutorial$ aws lambda add-layer-version-permission --layer-name bash-runtime --
version-number 2 \
--principal "*" --statement-id publish --action lambda:GetLayerVersion
{
    "Statement": "{\"Sid\":\"publish\",\"Effect\":\"Allow\",\"Principal\":\"*\",\"Action\":
\"lambda:GetLayerVersion\", \"Resource\":\"arn:aws:lambda:us-west-2:123456789012:layer:bash-
runtime:2\"}",
    "RevisionId": "9d5fe08e-2a1e-4981-b783-37ab551247ff"
}
```

Você pode adicionar várias instruções que concedam cada uma permissão a uma única conta, contas em uma organização ou todas as contas.

Limpar

Exclua cada versão da camada.

```
runtime-tutorial$ aws lambda delete-layer-version --layer-name bash-runtime --version-number 1
runtime-tutorial$ aws lambda delete-layer-version --layer-name bash-runtime --version-number 2
```

Como a função contém uma referência à versão 2 da camada, ela ainda existe no Lambda. A função continua a funcionar, mas as funções não podem mais ser configuradas para usar a versão excluída. Se você modificar a lista de camadas na função, deverá especificar uma nova versão ou omitir a camada excluída.

Exclua a função do tutorial com o comando `delete-function`.

```
runtime-tutorial$ aws lambda delete-function --function-name bash-runtime
```

Usar a vetorização AVX2 no Lambda

A AVX2 (Advanced Vector Extensions 2) é uma extensão de vetorização para o conjunto de instruções Intel x86 que pode desempenhar instruções de SIMD (Single Instruction, Multiple Data) sobre vetores de 256 bits. Para algoritmos vetorizáveis com operação [altamente paralelizável](#), o uso da AVX2 pode melhorar a performance da CPU, resultando em latências mais baixas e maior taxa de transferência. Use o conjunto de instruções da AVX2 para cargas de trabalho com uso intensivo de computação, como inferência de machine learning, processamento multimídia, simulações científicas e aplicações de modelagem financeira.

Para usar o AVX2 com sua função do Lambda, seu código de função deve acessar o código otimizado para AVX2. Em algumas linguagens, você pode instalar a versão de bibliotecas e pacotes compatível com a AVX2. Para outras linguagens, você pode recompilar o código e as dependências com os sinalizadores do compilador apropriados definidos (se o compilador aceitar vetorização automática). Você também pode compilar seu código com bibliotecas de terceiros que usam AVX2 para otimizar operações matemáticas. Por exemplo, Intel Math Kernel Library (Intel MKL), OpenBLAS (Basic Linear Algebra Subprograms) e AMD BLAS-like Library Instantiation Software (BLIS). Linguagens autovetorizadas, como Java, usam a AVX2 para cálculos automaticamente.

Você pode criar novas cargas de trabalho do Lambda ou mover cargas de trabalho habilitadas para AVX2 existentes para o Lambda sem nenhum custo adicional.

Para obter mais informações sobre a AVX2, consulte [Extensões de vetor avançadas 2](#) na Wikipédia.

Compilar a partir da origem

Se sua função do Lambda usar uma biblioteca C ou C++ para executar operações vetorizáveis com uso alto uso computação, você poderá definir os sinalizadores do compilador apropriados e recompilar o código da função. Em seguida, o compilador vetorizará o código automaticamente.

Para o compilador `gcc` ou `clang`, adicione `-march=haswell` ao comando ou defina `-mavx2` como uma opção de comando.

```
~ gcc -march=haswell main.c
or
~ gcc -mavx2 main.c

~ clang -march=haswell main.c
or
~ clang -mavx2 main.c
```

Para usar uma biblioteca específica, siga as instruções na documentação da biblioteca para realizar a compilação e a criação dela. Por exemplo, para criar o TensorFlow a partir da origem, você pode seguir as [instruções de instalação](#) no site do TensorFlow. Use a opção de compilação `-march=haswell`.

Ativar a AVX2 para Intel MKL

A Intel MKL é uma biblioteca de operações matemáticas otimizadas que usam implicitamente instruções AVX2 quando a plataforma de computação oferece compatibilidade. Frameworks como PyTorch [fazem criações com Intel MKL por padrão](#), portanto, você não precisa habilitar a AVX2.

Algumas bibliotecas, como o TensorFlow, oferecem opções no processo de compilação para especificar a otimização da Intel MKL. Por exemplo, com o TensorFlow, use a opção `--config=mkl`.

Você também pode criar bibliotecas Python científicas populares, como SciPy e NumPy, com a Intel MKL. Para obter instruções sobre como criar essas bibliotecas com a Intel MKL, consulte [Numpy/Scipy com Intel MKL e Intel Compilers](#) no site da Intel.

Para obter mais informações sobre a Intel MKL e bibliotecas semelhantes, consulte [Math Kernel Library](#) na Wikipédia, o [site da OpenBLAS](#) e o [repositório da AMD BLIS](#) no GitHub.

Compatibilidade com AVX2 em outras linguagens

Se você não usar bibliotecas C ou C++ e não criar com a Intel MKL, ainda poderá ter certa melhoria de performance da AVX2 em suas aplicações. Observe que a melhoria real depende da capacidade do compilador ou intérprete de utilizar os recursos AVX2 em seu código.

Python

Os usuários de Python geralmente usam bibliotecas SciPy e NumPy para cargas de trabalho com uso intensivo de computação. Você pode compilar essas bibliotecas para habilitar a AVX2 ou usar as versões das bibliotecas habilitadas para Intel MKL.

Nó

Para cargas de trabalho com uso intensivo de computação, use as versões habilitadas para AVX2 ou Intel MKL das bibliotecas de que você precisa.

Java

O compilador JIT do Java pode vetorizar automaticamente seu código para execução com instruções da AVX2. Para obter informações sobre como detectar código vetorizado, consulte a apresentação [Code vectorization in the JVM](#) no site OpenJDK.

Go

O compilador padrão do Go não tem compatibilidade com vetorização automática atualmente, mas você pode usar o [gccgo](#), o compilador GCC para Go. Defina a opção do `-mavx2`:

```
gcc -o avx2 -mavx2 -Wall main.c
```

Intrínsecos

É possível usar [funções intrínsecas](#) em várias linguagens para vetorizar seu código manualmente para usar a AVX2. No entanto, não recomendamos essa abordagem. Gravar código vetorizado manualmente requer um esforço significativo. Além disso, depurar e manter esse código é mais difícil do que usar código que dependa da autovetorização.

Usar imagens de contêiner com o Lambda

Você pode empacotar seu código e suas dependências como uma imagem de contêiner usando ferramentas como a command line interface (CLI – interface da linha de comando) do Docker. Em seguida, você pode fazer upload da imagem para o registro do contêiner hospedado no Amazon Elastic Container Registry (Amazon ECR).

AWSA oferece um conjunto de imagens de base de código aberto que você pode usar para criar a imagem do contêiner para o código de função. Você também pode usar imagens de base alternativas de outros registros de contêiner. A AWS também oferece um cliente de tempo de execução de código aberto que você adiciona à sua imagem de base alternativa para torná-la compatível com o serviço do Lambda.

Além disso, a AWS oferece um emulador de interface de tempo de execução para você testar as funções localmente usando ferramentas como a CLI do Docker.

Não há cobrança adicional para empacotar e implantar funções como imagens de contêiner. Quando uma função implantada como uma imagem de contêiner é invocada, você paga por solicitações de invocação e duração da execução. Você será cobrado com relação ao armazenamento de imagens de contêiner no Amazon ECR. Para obter mais informações, consulte a [Definição de preço do Amazon ECR](#).

Tópicos

- [Criar imagens de contêiner do Lambda \(p. 267\)](#)
- [Testar imagens de contêiner do Lambda no local \(p. 274\)](#)

Criar imagens de contêiner do Lambda

Você pode empacotar o código da função do Lambda e dependências como uma imagem de contêiner, usando ferramentas como a CLI do Docker. Em seguida, você pode fazer upload da imagem para o registro do contêiner hospedado no Amazon Elastic Container Registry (Amazon ECR). Observe que você deve criar a função do Lambda a partir da mesma conta que o registro de contêiner no Amazon ECR.

O AWS fornece um conjunto de [imagens básicas \(p. 222\)](#) de código aberto que você pode usar para criar sua imagem de contêiner. Essas imagens básicas incluem um [cliente de interface de tempo de execução \(p. 223\)](#) para gerenciar a interação entre o Lambda e o código da função.

Você também pode usar uma imagem base alternativa de outro registro de contêiner. O Lambda fornece clientes de interface de tempo de execução de código aberto que você adiciona a uma imagem básica alternativa para torná-la compatível com o Lambda.

Para aplicações de exemplo, incluindo um exemplo de Node.js e um de Python, consulte [Container image support for Lambda](#) no Blog da AWS.

Depois de criar uma imagem de contêiner no registro do contêiner do Amazon ECR, você pode [criar e executar \(p. 90\)](#) a função do Lambda.

Tópicos

- [Tipos de imagem \(p. 267\)](#)
- [Ferramentas de contêiner \(p. 267\)](#)
- [Requisitos do Lambda para imagens de contêiner \(p. 268\)](#)
- [Configurações de imagem de contêiner \(p. 268\)](#)
- [Criar uma imagem a partir de uma imagem básica da AWS para o Lambda \(p. 268\)](#)
- [Criar uma imagem a partir de uma imagem básica alternativa \(p. 271\)](#)
- [Criar uma imagem usando o toolkit do AWS SAM \(p. 272\)](#)

Tipos de imagem

Você pode usar umAWSA imagem básica da ou uma imagem básica alternativa, como Alpine ou Debian. O Lambda é compatível com qualquer imagem que esteja em conformidade com um dos seguintes formatos de manifesto de imagem:

- Esquema 2 de manifesto V2 de imagem de Docker (usado com o Docker versão 1.10 e posteriores)
- Especificações de Open Container Initiative (OCI – Iniciativa de contêiner aberto) (v1.0.0 e posteriores)

O Lambda é compatível com imagens de até 10 GB de tamanho.

Ferramentas de contêiner

Para criar sua imagem de contêiner, você pode usar qualquer ferramenta de desenvolvimento que ofereça suporte a um dos seguintes formatos de manifesto de imagem de contêiner:

- Esquema 2 de manifesto V2 de imagem de Docker (usado com o Docker versão 1.10 e posteriores)
- Especificações OCI (v1.0.0 e acima)

Por exemplo, você pode usar a CLI do Docker para criar, testar e implantar suas imagens de contêiner.

Requisitos do Lambda para imagens de contêiner

Para implantar uma imagem de contêiner no Lambda, observe os seguintes requisitos:

1. A imagem do contêiner deve implementar a [API Runtime \(p. 224\)](#) do Lambda. Os [clientes de interface de tempo de execução \(p. 223\)](#) de código aberto da AWS implementam a API. Você pode adicionar um cliente de interface de tempo de execução à sua imagem base preferida para torná-lo compatível com o Lambda.
2. Deve ser possível executar a imagem do contêiner em um sistema de arquivos somente leitura. O código da função pode acessar um diretório /tmp gravável com 512 MB de armazenamento.
3. O usuário padrão do Lambda deve ser capaz de ler todos os arquivos necessários para executar seu código de função. O Lambda segue as práticas recomendadas de segurança definindo um usuário padrão do Linux com permissões menos privilegiadas. Verifique se o código da aplicação não depende de arquivos que outros usuários do Linux estejam restringidos de executar.
4. O Lambda é compatível apenas com imagens de contêiner baseadas em Linux.

Configurações de imagem de contêiner

O Lambda é compatível com as seguintes configurações de imagem de contêiner no Dockerfile:

- `ENTRYPOINT`: especifica o caminho absoluto para o ponto de entrada da aplicação.
- `CMD`: especifica parâmetros adicionais que você deseja transmitir com `ENTRYPOINT`.
- `WORKDIR`: especifica o caminho absoluto para o diretório de trabalho.
- `ENV`: especifica uma variável de ambiente para a função do Lambda.

Note

O Lambda ignora os valores de quaisquer configurações de imagem de contêiner não compatíveis existentes no Dockerfile.

Para obter mais informações sobre como o Docker usa as configurações de imagem de contêiner, consulte [ENTRYPOINT](#) na referência do Dockerfile no site Docker Docs. Para obter mais informações sobre como usar `ENTRYPOINT` e `CMD`, consulte [Demystifying ENTRYPOINT and CMD in Docker](#) no Blog de código aberto da AWS.

Você pode especificar as configurações de imagem de contêiner no Dockerfile ao criar sua imagem. Você também pode substituir essas configurações usando o console do Lambda ou a API do Lambda. Isso permite implantar várias funções que implementam a mesma imagem de contêiner, mas com diferentes configurações de tempo de execução.

Warning

Quando você especifica `ENTRYPOINT` ou `CMD` no Dockerfile ou como uma substituição, certifique-se de inserir o caminho absoluto. Além disso, não use links simbólicos como ponto de entrada para o contêiner.

Criar uma imagem a partir de uma imagem básica da AWS para o Lambda

Para criar uma imagem de contêiner para uma nova função do Lambda, você pode começar com uma imagem básica da AWS para o Lambda.

Note

A AWS fornece atualizações periodicamente para as imagens básicas da AWS para o Lambda. Se o Dockerfile incluir o nome da imagem na propriedade FROM, seu cliente Docker extrai a versão mais recente da imagem do Docker Hub. Para usar a imagem básica atualizada, você deve reconstruir a imagem do contêiner e [atualizar o código da função \(p. 92\)](#).

Pré-requisitos

- A AWS Command Line Interface (AWS CLI)

As instruções a seguir usam a AWS CLI para chamar operações da API de serviço da AWS. Para instalar a AWS CLI, consulte [Instalar, atualizar e desinstalar a AWS CLI](#) no Manual do usuário da AWS Command Line Interface.

- Docker Desktop

As instruções a seguir usam os comandos da CLI do Docker para criar a imagem do contêiner. Para instalar a CLI do Docker, consulte [Get Docker](#) no site Docker Docs.

- Código da função

Para criar uma imagem a partir de uma imagem básica da AWS para o Lambda

1. Em sua máquina local, crie um diretório de projeto para sua nova função.
2. Crie um diretório nomeado `app` no diretório do projeto e, em seguida, adicione o código do manipulador de funções ao diretório da aplicação.
3. Use um editor de texto para criar um novo Dockerfile.

As imagens básicas da AWS fornecem as seguintes variáveis de ambiente:

- `LAMBDA_TASK_ROOT=/var/task`
- `LAMBDA_RUNTIME_DIR=/var/runtime`

Instale qualquer dependência no diretório `${LAMBDA_TASK_ROOT}` junto do manipulador de funções para garantir que o tempo de execução do Lambda possa localizá-la quando a função for invocada.

O exemplo a seguir mostra um Dockerfile para Node.js, Python e Ruby:

Node.js 14

```
FROM public.ecr.aws/lambda/nodejs:14
# Alternatively, you can pull the base image from Docker Hub: amazon/aws-lambda-nodejs:12

# Assumes your function is named "app.js", and there is a package.json file in the
# app directory
COPY app.js package.json  ${LAMBDA_TASK_ROOT}

# Install NPM dependencies for function
RUN npm install

# Set the CMD to your handler (could also be done as a parameter override outside
# of the Dockerfile)
CMD [ "app.handler" ]
```

Python 3.8

```
FROM public.ecr.aws/lambda/python:3.8
```

```
# Copy function code
COPY app.py ${LAMBDA_TASK_ROOT}

# Install the function's dependencies using file requirements.txt
# from your project folder.

COPY requirements.txt .
RUN pip3 install -r requirements.txt --target "${LAMBDA_TASK_ROOT}"

# Set the CMD to your handler (could also be done as a parameter override outside
# of the Dockerfile)
CMD [ "app.handler" ]
```

Ruby 2.7

```
FROM public.ecr.aws/lambda/ruby:2.7

# Copy function code
COPY app.rb ${LAMBDA_TASK_ROOT}

# Copy dependency management file
COPY Gemfile ${LAMBDA_TASK_ROOT}

# Install dependencies under LAMBDA_TASK_ROOT
ENV GEM_HOME=${LAMBDA_TASK_ROOT}
RUN bundle install

# Set the CMD to your handler (could also be done as a parameter override outside
# of the Dockerfile)
CMD [ "app.LambdaFunction::Handler.process" ]
```

- Crie sua imagem do Docker com o comando `docker build`. Insira um nome para a imagem. O exemplo a seguir nomeia a imagem como `hello-world`.

```
docker build -t hello-world .
```

- Inicie a imagem do Docker com o comando `docker run`. Para esse exemplo, insira `hello-world` como o nome da imagem.

```
docker run -p 9000:8080 hello-world
```

- (Opcional) Teste sua aplicação localmente usando o [emulador de interface de tempo de execução \(p. 274\)](#). Em uma nova janela de terminal, publique um evento no seguinte endpoint usando um comando `curl`:

```
curl -XPOST "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca a função do em execução na imagem do contêiner e retorna uma resposta.

- Autentique a CLI do Docker no seu registro do Amazon ECR.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-
stdin 123456789012.dkr.ecr.us-east-1.amazonaws.com
```

- Crie um repositório no Amazon ECR usando o comando `create-repository`.

```
aws ecr create-repository --repository-name hello-world --image-scanning-configuration
scanOnPush=true --image-tag-mutability MUTABLE
```

9. Marque sua imagem para corresponder ao nome do repositório e implante a imagem no Amazon ECR usar o comando `docker push`.

```
docker tag hello-world:latest 123456789012.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
docker push 123456789012.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

Agora que a imagem de contêiner reside no registro do contêiner do Amazon ECR, você pode [criar e executar](#) (p. 90) a função do Lambda.

Criar uma imagem a partir de uma imagem básica alternativa

Pré-requisitos

- A AWS CLI
- Docker Desktop
- Código da função

Para criar uma imagem usando uma imagem básica alternativa

1. Escolha uma imagem de base. O Lambda é compatível com todas as distribuições Linux, como Alpine, Debian e Ubuntu.
2. Em sua máquina local, crie um diretório de projeto para sua nova função.
3. Crie um diretório nomeado `app` no diretório do projeto e, em seguida, adicione o código do manipulador de funções ao diretório da aplicação.
4. Use um editor de texto para criar um novo Dockerfile com a seguinte configuração:
 - Defina a propriedade `FROM` como o URI da imagem básica.
 - Adicione instruções para instalar o cliente de interface de tempo de execução.
 - Defina a propriedade `ENTRYPOINT` para chamar o cliente de interface de tempo de execução.
 - Defina o argumento `CMD` para especificar o manipulador de funções do Lambda.

O exemplo a seguir mostra um Dockerfile para Python:

```
# Define function directory
ARG FUNCTION_DIR="/function"

FROM python:buster as build-image

# Install aws-lambda-cpp build dependencies
RUN apt-get update && \
    apt-get install -y \
    g++ \
    make \
    cmake \
    unzip \
    libcurl4-openssl-dev

# Include global arg in this stage of the build
ARG FUNCTION_DIR
# Create function directory
RUN mkdir -p ${FUNCTION_DIR}
```

```
# Copy function code
COPY app/* ${FUNCTION_DIR}

# Install the runtime interface client
RUN pip install \
    --target ${FUNCTION_DIR} \
    awslambdaric

# Multi-stage build: grab a fresh copy of the base image
FROM python:buster

# 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 build image dependencies
COPY --from=build-image ${FUNCTION_DIR} ${FUNCTION_DIR}

ENTRYPOINT [ "/usr/local/bin/python", "-m", "awslambdaric" ]
CMD [ "app.handler" ]
```

- Crie sua imagem do Docker com o comando `docker build`. Insira um nome para a imagem. O exemplo a seguir nomeia a imagem como `hello-world`.

```
docker build -t hello-world .
```

- (Opcional) Teste sua aplicação localmente usando o [emulador de interface de tempo de execução \(p. 274\)](#).
- Autentique a CLI do Docker no seu registro do Amazon ECR.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-
stdin 123456789012.dkr.ecr.us-east-1.amazonaws.com
```

- Crie um repositório no Amazon ECR usando o comando `create-repository`.

```
aws ecr create-repository --repository-name hello-world --image-scanning-configuration
scanOnPush=true --image-tag-mutability MUTABLE
```

- Marque sua imagem para corresponder ao nome do repositório e implante a imagem no Amazon ECR usando o comando `docker push`.

```
docker tag hello-world:latest 123456789012.dkr.ecr.us-east-1.amazonaws.com/hello-
world:latest
docker push 123456789012.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

Agora que a imagem de contêiner reside no registro do contêiner do Amazon ECR, você pode [criar e executar \(p. 90\)](#) a função do Lambda.

Criar uma imagem usando o toolkit do AWS SAM

Você pode usar o toolkit do AWS Serverless Application Model (AWS SAM) para criar e implantar uma função definida como uma imagem de contêiner. Para um novo projeto, você pode usar o comando AWS SAM da CLI do `init` para configurar a estrutura do projeto em seu tempo de execução preferido.

No modelo do AWS SAM, defina o tipo `Runtime` como `Image` e forneça o URI da imagem básica.

Para obter mais informações, consulte [Building applications](#) no Guia do desenvolvedor do AWS Serverless Application Model.

Testar imagens de contêiner do Lambda no local

O AWS Lambda Runtime Interface Emulator (RIE) é um proxy para a API Runtime do Lambda que permite testar localmente sua função do Lambda empacotada como uma imagem de contêiner. O emulador é um servidor web leve que converte solicitações HTTP em eventos JSON para passar para a função do Lambda na imagem do contêiner.

As imagens de base da AWS para o Lambda incluem o componente RIE. Se você usar uma imagem de base alternativa, poderá testá-la sem adicionar o RIE. Você também pode criar o componente RIE em sua imagem de base. A AWS oferece um componente RIE de código aberto no repositório da AWS no GitHub.

Você pode usar o emulador para testar se seu código de função é compatível com o ambiente do Lambda. Também use o emulador para testar se sua função do Lambda é executada até a conclusão com êxito e fornece o resultado esperado. Se você criar extensões e agentes em sua imagem de contêiner, poderá usar o emulador para testar se as extensões e os agentes funcionam corretamente com a API Extensions do Lambda.

Para obter exemplos de como usar o RIE, consulte [Container image support for Lambda](#) no blog da AWS.

Tópicos

- [Diretrizes para o uso do RIE \(p. 274\)](#)
- [Variáveis de ambiente \(p. 274\)](#)
- [Testar uma imagem com o RIE incluído nela \(p. 275\)](#)
- [Criar o RIE na sua imagem de base \(p. 275\)](#)
- [Testar uma imagem sem adicionar RIE a ela \(p. 276\)](#)

Diretrizes para o uso do RIE

Observe as diretrizes a seguir ao usar o Runtime Interface Emulator:

- O RIE não emula as configurações de segurança e autenticação do Lambda ou a orquestração do Lambda.
- O emulador apenas oferece suporte a arquiteturas Linux x86-64.
- O emulador não é compatível com rastreamento do AWS X-Ray ou outras integrações do Lambda.

Variáveis de ambiente

O emulador de interface de tempo de execução é compatível com um subconjunto de [variáveis de ambiente \(p. 99\)](#) para a função do Lambda na imagem em execução local.

Se sua função usar credenciais de segurança, você poderá configurá-las definindo as seguintes variáveis de ambiente:

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`
- `AWS_SESSION_TOKEN`
- `AWS_REGION`

Para definir o tempo limite da função, configure `AWS_LAMBDA_FUNCTION_TIMEOUT`. Digite o número máximo de segundos que você deseja permitir que a função seja executada.

O emulador não preenche as variáveis de ambiente do Lambda a seguir. No entanto, você pode configurá-las para corresponder aos valores esperados quando a função é executada no serviço do Lambda:

- AWS_LAMBDA_FUNCTION_VERSION
- AWS_LAMBDA_FUNCTION_NAME
- AWS_LAMBDA_FUNCTION_MEMORY_SIZE

Testar uma imagem com o RIE incluído nela

As imagens de base da AWS para o Lambda incluem o emulador de interface de tempo de execução. Você também pode seguir estas etapas se você criou o RIE em sua imagem de base alternativa.

Para testar sua função do Lambda com o emulador

1. Crie sua imagem localmente usando o comando `docker build`.

```
docker build -t myfunction:latest .
```

2. Execute sua imagem de contêiner localmente usando o comando `docker run`.

```
docker run -p 9000:8080 myfunction:latest
```

Esse comando executa a imagem como um contêiner e inicia um endpoint localmente em `localhost:9000/2015-03-31/functions/function/invocations`.

3. Em uma nova janela de terminal, publique um evento no seguinte endpoint usando um comando `curl`:

```
curl -XPOST "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca a função do Lambda em execução na imagem do contêiner e retorna uma resposta.

Criar o RIE na sua imagem de base

Você pode criar RIE em uma imagem de base. Faça download do RIE do GitHub para sua máquina local e atualize seu Dockerfile para instalar o RIE.

Para criar o emulador em sua imagem

1. Crie um script e salve-o no diretório do projeto. Defina permissões de execução para o arquivo de script.

O script verifica a presença da variável de ambiente `AWS_LAMBDA_RUNTIME_API`, o que indica a presença da API de tempo de execução. Se a API Runtime estiver presente, o script executa o [cliente de interface de tempo de execução \(p. 223\)](#). Caso contrário, o script executa o emulador de interface de tempo de execução.

O exemplo abaixo mostra um script típico para uma função do Node.js.

```
#!/bin/sh
if [ -z "${AWS_LAMBDA_RUNTIME_API}" ]; then
    exec /usr/local/bin/aws-lambda-rie /usr/local/bin/npx aws-lambda-ric $@
else
```

```
    exec /usr/local/bin/npx aws-lambda-ric $@
fi
```

O exemplo abaixo mostra um script típico para uma função do Python.

```
#!/bin/sh
if [ -z "${AWS_LAMBDA_RUNTIME_API}" ]; then
    exec /usr/local/bin/aws-lambda-rie /usr/local/bin/python -m awslambdaric $@
else
    exec /usr/local/bin/python -m awslambdaric $@
fi
```

2. Faça download do [emulador de interface de tempo de execução](#) do GitHub para o diretório do projeto.
3. Copie o script, instale o pacote do emulador e altere o `ENTRYPOINT` para executar o novo script adicionando as seguintes linhas ao seu Dockerfile:

```
COPY ./entry_script.sh /entry_script.sh
ADD aws-lambda-rie /usr/local/bin/aws-lambda-rie
ENTRYPOINT [ "/entry_script.sh" ]
```

4. Crie sua imagem localmente usando o comando `docker build`.

```
docker build -t myfunction:latest .
```

Testar uma imagem sem adicionar RIE a ela

Instale o emulador de interface de tempo de execução na sua máquina local. Ao executar a imagem do contêiner, defina o ponto de entrada para ser o emulador.

Para testar uma imagem sem adicionar RIE a ela

1. No diretório do projeto, execute o comando a seguir para fazer download do RIE do GitHub e instalá-lo em sua máquina local.

```
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
```

2. Execute sua função do Lambda usando o comando `docker run`.

```
docker run -d -v ~/.aws-lambda-rie:/aws-lambda -p 9000:8080 \
--entrypoint /aws-lambda/aws-lambda-rie hello-world:latest <image entrypoint> \
<(optional) image command>
```

Isso executa a imagem como um contêiner e inicia um endpoint localmente em `localhost:9000/2015-03-31/functions/function/invocations`.

3. Publique um evento no seguinte endpoint usando um comando `curl`:

```
curl -XPOST "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca a função do em execução na imagem do contêiner e retorna uma resposta.

Usar o AWS Lambda com outros serviços

O AWS Lambda se integra a outros serviços da AWS para invocar funções. É possível configurar triggers para invocar uma função em resposta a eventos de ciclo de vida de recursos, responder a solicitações HTTP de entrada, consumir eventos de uma fila ou [serem executados em uma programação \(p. 315\)](#).

Cada serviço que se integra ao Lambda envia dados para sua função no JSON como um evento. A estrutura do documento do evento é diferente para cada tipo de evento e contém dados sobre o recurso ou a solicitação que acionou a função. Os tempos de execução do Lambda convertem o evento em um objeto e o transmitem para a função.

O exemplo a seguir mostra um evento de teste de um [balanceador de carga da aplicação \(p. 370\)](#) que representa uma solicitação GET para /lambda?query=1234ABCD.

Example de um Application Load Balancer

```
{
    "requestContext": {
        "elb": {
            "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:123456789012: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-2.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
}
```

Note

O tempo de execução do Lambda converte o documento do evento em um objeto e o transmite ao [manipulador da função \(p. 18\)](#). Para linguagens compiladas, o Lambda fornece definições para tipos de evento em uma biblioteca. Consulte os tópicos a seguir para obter mais informações.

- [Construir funções do Lambda com Java \(p. 592\)](#)

- [Criar funções do Lambda com Go \(p. 633\)](#)
- [Construir funções do Lambda com C# \(p. 661\)](#)
- [Construir funções do Lambda com o PowerShell \(p. 692\)](#)

Para serviços que geram uma fila ou fluxo de dados, você cria um[Mapeamento de origens do \(p. 170\)](#)no Lambda e conceder permissão ao Lambda para acessar o outro serviço no[Função de execução do \(p. 57\)](#). O Lambda lê dados do outro serviço, cria um evento e invoca a função.

Serviços dos quais o Lambda lê eventos

- [Amazon DynamoDB \(p. 333\)](#)
- [Amazon Kinesis \(p. 387\)](#)
- [Amazon MQ \(p. 411\)](#)
- [Amazon Managed Streaming for Apache Kafka \(p. 419\)](#)
- [Apache Kafka autogerenciado \(p. 377\)](#)
- [Amazon Simple Queue Service \(p. 465\)](#)

Outros serviços invocam a função diretamente. Você concede permissão ao outro serviço na [política baseada em recursos \(p. 62\)](#) da função e configura o outro serviço para gerar eventos e invocar a função. Dependendo do serviço, a invocação pode ser síncrona ou assíncrona. Para a invocação síncrona, o outro serviço aguarda a resposta da função e pode [tentar novamente quando houver erros \(p. 176\)](#).

Para obter mais informações sobre arquiteturas de serviço do Lambda, consulte [Arquiteturas orientadas por eventos](#) no guia do operador do Lambda.

Serviços que invocam funções do Lambda de forma síncrona

- [Elastic Load Balancing Application Load Balancer \(p. 370\)](#)
- [Amazon Cognito \(p. 330\)](#)
- [Amazon connect \(p. 332\)](#)
- [Amazon Lex \(p. 408\)](#)
- [Amazon Alexa \(p. 280\)](#)
- [Amazon API Gateway \(p. 281\)](#)
- [Amazon CloudFront \(Lambda@Edge\) \(p. 325\)](#)
- [Amazon Kinesis Data Firehose \(p. 386\)](#)
- [Amazon Simple Storage Service Batch \(p. 452\)](#)
- [Secrets Manager \(p. 454\)](#)

Para a invocação assíncrona, o Lambda coloca o evento em filas antes de transmiti-lo para a função. O outro serviço obtém uma resposta de êxito assim que o evento é colocado em fila e não tem conhecimento do que acontece posteriormente. Se ocorrer um erro, o Lambda processará [novas tentativas \(p. 176\)](#) e poderá enviar eventos com falha a um [destino \(p. 164\)](#) que você configurar. Você pode desabilitar novas tentativas para uma função definindo o valor de simultaneidade reservada da função como zero.

Serviços que invocam funções do Lambda de forma assíncrona

- [Amazon Simple Storage Service \(p. 431\)](#)
- [Amazon Simple Notification Service \(p. 457\)](#)
- [Amazon Simple Email Service \(p. 455\)](#)
- [AWS CloudFormation \(p. 323\)](#)
- [Amazon CloudWatch Logs \(p. 322\)](#)

- [Amazon CloudWatch Events \(p. 315\)](#)
- [AWS CodeCommit \(p. 327\)](#)
- [AWS Config \(p. 331\)](#)
- [AWS IoT \(p. 374\)](#)
- [AWS IoT Eventos \(p. 375\)](#)
- [AWS CodePipeline \(p. 328\)](#)

Além disso, alguns serviços se integram ao Lambda de formas que não envolvem a invocação de funções.

Serviços que se integram ao Lambda de outras formas

- [Amazon Elastic File System \(p. 372\)](#)
- [AWS X-Ray \(p. 477\)](#)

Consulte os tópicos deste capítulo para obter mais detalhes sobre cada serviço e eventos de exemplo que você pode usar para testar a função.

Usar o AWS Lambda com o Alexa

Você pode usar funções do Lambda para criar serviços que oferecem novas habilidades para a Alexa, o assistente de voz no Amazon Echo. O Alexa Skills Kit fornece as APIs, as ferramentas e a documentação para criar essas novas habilidades, com base em seus próprios serviços em execução como funções do Lambda. Os usuários do Amazon Echo podem acessar essas novas habilidades fazendo perguntas ao Alexa ou fazendo solicitações.

O Alexa Skills Kit está disponível no GitHub.

- [SDK for Java do Alexa Skills Kit](#)
- [Alexa Skills Kit SDK for Node.js](#)
- [SDK for Python do Alexa Skills Kit](#)

Example Evento de casa inteligente da Alexa

```
{  
  "header": {  
    "payloadVersion": "1",  
    "namespace": "Control",  
    "name": "SwitchOnOffRequest"  
  },  
  "payload": {  
    "switchControlAction": "TURN_ON",  
    "appliance": {  
      "additionalApplianceDetails": {  
        "key2": "value2",  
        "key1": "value1"  
      },  
      "applianceId": "sampleId"  
    },  
    "accessToken": "sampleAccessToken"  
  }  
}
```

Para obter mais informações, consulte [Hospedar uma skill personalizada como função do AWS Lambda](#) no guia Crie skills com a documentação do desenvolvedor do Alexa Skills Kit.

Usar o AWS Lambda com o Amazon API Gateway

Você pode criar uma API da Web com um endpoint HTTP para a função do Lambda usando o Amazon API Gateway. O API Gateway fornece ferramentas para criar e documentar APIs da Web que direcionam as solicitações HTTP para as funções do Lambda. Você pode proteger o acesso à sua API com controles de autenticação e autorização. Suas APIs podem veicular o tráfego pela Internet ou podem ser acessadas somente dentro da VPC.

Como adicionar um endpoint público à sua função do Lambda

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Em Functional overview (Visão geral funcional), escolha Add trigger (Adicionar trigger).
4. Selecione API Gateway.
5. Em API, escolha Create an API (Criar uma API).
6. Em Security (Segurança), escolha Open (Abrir).
7. Escolha Adicionar.

As APIs do API Gateway são compostas por estágios, recursos, métodos e integrações. O estágio e o recurso determinam o caminho do endpoint:

Formato do caminho da API

- `/prod/`: o estágio e o recurso raiz de `prod`.
- `/prod/user`: o estágio de `prod` e o recurso de `user`.
- `/dev/{proxy+}`: qualquer rota no estágio de `dev`.
- `/`: (APIs HTTP) o estágio padrão e o recurso raiz.

Uma integração do Lambda mapeia uma combinação de caminho e método de HTTP para uma função do Lambda. Você pode configurar o API Gateway para transmitir o corpo da solicitação de HTTP no estado em que se encontra (integração personalizada) ou para encapsular o corpo da solicitação em um documento que inclui todas as informações de solicitação, inclusive cabeçalhos, recursos, caminho e método.

O Amazon API Gateway invoca sua função [de modo síncrono \(p. 158\)](#) com um evento que contém uma representação JSON da solicitação de HTTP. Para uma integração personalizada, o evento é o corpo da solicitação. Para uma integração de proxy, o evento tem uma estrutura definida. O exemplo abaixo mostra um evento de proxy de uma API REST do API Gateway.

Example Evento de proxy [event.json](#) do API Gateway (API REST)

```
{  
    "resource": "/",  
    "path": "/",  
    "httpMethod": "GET",  
    "requestContext": {  
        "resourcePath": "/",  
        "httpMethod": "GET",  
        "path": "/Prod/",  
        ...  
    },  
    "headers": {  
        "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/  
apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",  
    }  
}
```

```

    "accept-encoding": "gzip, deflate, br",
    "Host": "70ixmpl4fl.execute-api.us-east-2.amazonaws.com",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/80.0.3987.132 Safari/537.36",
    "X-Amzn-Trace-Id": "Root=1-5e66d96f-7491f09xmpl79d18acf3d050",
    ...
},
"multiValueHeaders": {
    "accept": [
        "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*
;q=0.8,application/signed-exchange;v=b3;q=0.9"
    ],
    "accept-encoding": [
        "gzip, deflate, br"
    ],
    ...
},
"queryStringParameters": null,
"multiValueQueryStringParameters": null,
"pathParameters": null,
"stageVariables": null,
"body": null,
"isBase64Encoded": false
}

```

O exemplo mostra um evento de uma solicitação GET para o caminho raiz do estágio Prod de uma API REST. A forma e o conteúdo do evento variam de acordo com o [tipo de API](#) (p. 285) e a configuração.

O API Gateway aguarda uma resposta de sua função e retransmite o resultado para o chamador. Para uma integração personalizada, defina uma resposta de integração e uma resposta de método para converter a saída da função em uma resposta de HTTP. Para uma integração de proxy, a função deve responder com uma representação da resposta em um formato específico.

O exemplo abaixo mostra um objeto de resposta de uma função Node.js. O objeto de resposta representa uma resposta de HTTP bem-sucedida que contém um documento JSON.

Example [index.js](#): objeto de resposta de integração de 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}"
}

```

O tempo de execução do Lambda serializa o objeto de resposta em JSON e o envia para a API. A API analisa a resposta e a utiliza para criar uma resposta de HTTP que, então, é enviada para o cliente que fez a solicitação original.

Example Resposta 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
}
```

Os recursos em sua API definem um ou mais métodos, como GET ou POST. Os métodos têm uma integração que direciona solicitações para uma função do Lambda ou outro tipo de integração. Você pode definir cada recurso e método individualmente ou usar tipos especiais de recurso e método para corresponder todas as solicitações que se enquadram em um padrão. Um recurso proxy captura todos os caminhos abaixo de um recurso. O método ANY captura todos os métodos HTTP.

Secções

- Permissions (p. 283)
 - Tratamento de erros com uma API do API Gateway (p. 284)
 - Escolher um tipo de API (p. 285)
 - Aplicativos de exemplo (p. 287)
 - Tutorial: Usar o AWS Lambda com o Amazon API Gateway (p. 287)
 - Código de exemplo da função do (p. 298)
 - Criar um microsserviço simples usando o Lambda e o API Gateway (p. 300)
 - Modelo do AWS SAM para uma aplicação do API Gateway (p. 302)

Permissions

O Amazon API Gateway recebe permissão para invocar sua função por meio da [política baseada em recursos](#) ([p. 62](#)) da função. Você pode conceder permissão de invocar a uma API inteira ou conceder acesso limitado a um estágio, recurso ou método.

Ao adicionar uma API à sua função usando o console do Lambda, o console do API Gateway ou em um modelo de AWS SAM, a política baseada em recursos da função é atualizada automaticamente. O exemplo abaixo mostra uma política de função com uma instrução que foi adicionada por um modelo do AWS SAM.

Example política de função

```
{
    "Version": "2012-10-17",
    "Id": "default",
    "Statement": [
        {
            "Sid": "nodejs-apig-functiongetEndpointPermissionProd-BWDBXMPLEXE2F",
            "Effect": "Allow",
            "Principal": {
                "Service": "apigateway.amazonaws.com"
            },
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:us-east-2:123456789012:function:nodejs-apig-
function-1G3MXMPLXVXYI",
            "Condition": {
                "ArnLike": {
                    "AWS:SourceArn": "arn:aws:execute-api:us-east-2:123456789012:ktyvxmpls1/*/*GET/*"
                }
            }
        }
    ]
}
```

```
    ]  
}
```

Confirme a política de função ([p. 62](#)) na guia Permissions (Permissões) do console do Lambda.

Você pode gerenciar manualmente as permissões da política de função com as seguintes operações da API:

- [AddPermission \(p. 775\)](#)
- [RemovePermission \(p. 942\)](#)
- [GetPolicy \(p. 869\)](#)

Para conceder permissão de invocação a uma API existente, use o comando `add-permission`.

```
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/"
```

Você deve ver a saída a seguir:

```
{  
  "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 a sua função e a API estiverem em regiões diferentes, o identificador de região no ARN de origem deverá corresponder à região da função, e não à região da API. Quando o API Gateway invoca uma função, ela usa um ARN de recurso baseado no ARN da API, mas modificado para corresponder à região da função.

O ARN de origem no exemplo concede permissão a uma integração no método GET do recurso raiz no estágio padrão de uma API com o ID `mnh1xmpli7`. Você pode usar um asterisco no ARN de origem para conceder permissões a vários estágios, métodos ou recursos.

Padrões de recursos

- `mnh1xmpli7/*/GET/*`: método GET em todos os recursos, em todos os estágios.
- `mnh1xmpli7/prod/ANY/user`: método ANY no recurso `user` no estágio `prod`.
- `mnh1xmpli7/*/*/*`: qualquer método em todos os recursos, em todos os estágios.

Para obter detalhes sobre como exibir a política e remover instruções, consulte [Limpar políticas baseadas em recursos \(p. 66\)](#).

Tratamento de erros com uma API do API Gateway

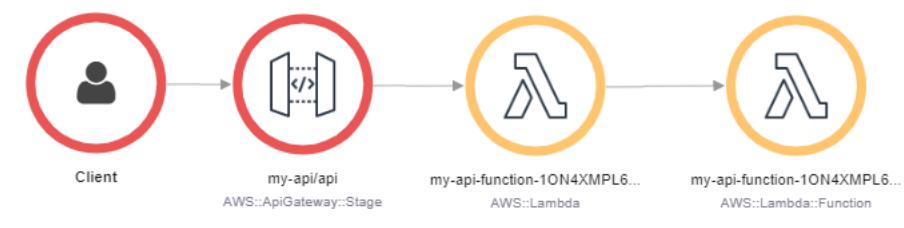
O API Gateway trata todos os erros de invocação e função como erros internos. Se a API do Lambda rejeitar a solicitação de invocação, o API Gateway retornará um código de erro 500. Se a função é executada, mas retorna um erro ou retorna uma resposta no formato errado, o API Gateway retorna

um código de erro 502. Em ambos os casos, o corpo da resposta do API Gateway é `{"message": "Internal server error"}`.

Note

O API Gateway não tenta novamente nenhuma invocação do Lambda. Se o Lambda retornar um erro, o API Gateway retornará uma resposta de erro ao cliente.

O exemplo a seguir mostra um mapa de rastreamento do X-Ray para uma solicitação que resultou em um erro de função e um erro 502 do API Gateway. O cliente recebe a mensagem de erro genérica.

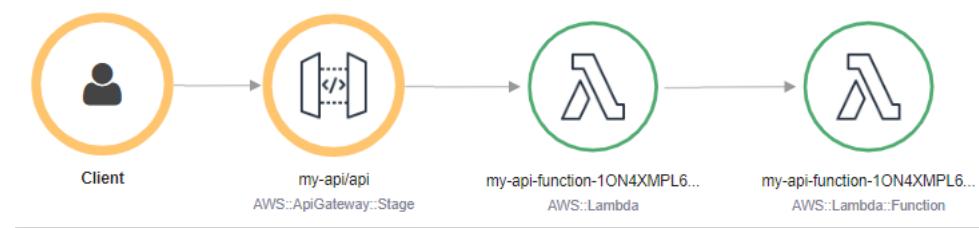


Para personalizar a resposta de erro, é necessário detectar erros no código e formatar uma resposta no formato necessário.

Example [index.js](#): erro de formatação

```
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
}
```

O API Gateway converte essa resposta em um erro HTTP com um código de status e um corpo personalizado. No mapa de rastreamento, o nó da função é verde porque ele tratou o erro.



Escolher um tipo de API

O API Gateway é compatível com três tipos de API que invocam as funções do Lambda:

- API HTTP: uma API RESTful leve e de baixa latência.
- API REST: uma API RESTful personalizável e com muitos recursos.
- API WebSocket: uma API da Web que mantém conexões persistentes com clientes para uma comunicação duplex completa.

As APIs HTTP e REST são ambas APIs RESTful que processam solicitações de HTTP e retornam respostas. As APIs HTTP são mais recentes e são criadas com a versão 2 da API do API Gateway. Os recursos abaixo são novos para APIs HTTP:

Recursos da API HTTP

- Implantações automáticas: quando você modifica rotas ou integrações, as alterações são implantadas automaticamente em estágios com implantação automática habilitada.
- Estágio padrão: você pode criar um estágio padrão (`$default`) para veicular solicitações no caminho raiz do URL da API. Para estágios nomeados, você deve incluir o nome do estágio no início do caminho.
- Configuração de CORS: você pode configurar sua API para adicionar cabeçalhos CORS às respostas de saída em vez de adicioná-las manualmente no código da função.

As APIs REST são APIs RESTful clássicas às quais o API Gateway oferece suporte desde o lançamento. As APIs REST têm atualmente mais recursos de personalização, integração e gerenciamento.

Recursos da API REST

- Tipos de integração: APIs REST são compatíveis com integrações personalizadas do Lambda. Com uma integração personalizada, você pode enviar apenas o corpo da solicitação para a função ou aplicar um modelo de transformação ao corpo da solicitação antes de enviá-la para a função.
- Controle de acesso: as APIs REST oferecem suporte a mais opções de autenticação e autorização.
- Monitoramento e rastreamento: as APIs REST oferecem suporte ao monitoramento do AWS X-Ray e a outras opções de log.

Para obter uma comparação detalhada, consulte [Escolher entre APIs HTTP e REST](#) no Guia do desenvolvedor do API Gateway.

As APIs WebSocket também usam a API do API Gateway versão 2 e oferecem suporte a um conjunto de recursos semelhante. Use uma API WebSocket para aplicativos que se beneficiam de uma conexão persistente entre o cliente e a API. As APIs WebSocket oferecem uma comunicação duplex completa, o que significa que tanto o cliente quanto a API podem enviar mensagens continuamente sem esperar por uma resposta.

As APIs HTTP oferecem suporte a um formato de evento simplificado (versão 2.0). O exemplo abaixo mostra um evento de uma API HTTP.

Example [event-v2.json](#): evento de proxy do API Gateway (API HTTP)

```
{  
    "version": "2.0",  
    "routeKey": "ANY /nodejs-apig-function-1G3Xmplzxvxyi",  
    "rawPath": "/default/nodejs-apig-function-1G3Xmplzxvxyi",  
    "rawQueryString": "",  
    "cookies": [  
        "s_fid=7AABXMPL1AFD9BBF-0643Xmpl09956DE2",  
        "regStatus=pre-register"  
    ],  
    "headers": {  
        "accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",  
        "accept-encoding": "gzip, deflate, br",  
        ...  
    },  
    "requestContext": {  
        "accountId": "123456789012",  
        "apiId": "r3pmxmplak",  
        "domainName": "r3pmxmplak.execute-api.us-east-2.amazonaws.com",  
        "stage": "prod"  
    }  
}
```

```
"domainPrefix": "r3pmxmplak",
"http": {
    "method": "GET",
    "path": "/default/nodejs-apig-function-1G3XMPPLZXVXYI",
    "protocol": "HTTP/1.1",
    "sourceIp": "205.255.255.176",
    "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"
},
"requestId": "JKJaXmPLvHcESEA=",
"routeKey": "ANY /nodejs-apig-function-1G3XMPPLZXVXYI",
"stage": "default",
"time": "10/Mar/2020:05:16:23 +0000",
"timeEpoch": 1583817383220
},
"isBase64Encoded": true
}
```

Para obter mais informações, consulte [Integrações do AWS Lambda](#) no Guia do desenvolvedor do API Gateway.

Aplicativos de exemplo

O repositório do GitHub desse guia fornece a aplicação de exemplo a seguir para o API Gateway.

- [API Gateway com Node.js](#): uma função com um modelo do AWS SAM que cria uma API REST com o rastreamento do AWS X-Ray habilitado. Ele inclui scripts para implantar, invocar a função, testar a API e limpar.

O Lambda também fornece [esquemas \(p. 28\)](#) e [modelos \(p. 29\)](#) que você pode usar para criar uma aplicação do API Gateway no console do Lambda.

Tutorial: Usar o AWS Lambda com o Amazon API Gateway

Neste exemplo, você cria uma API simples usando o Amazon API Gateway. Um Amazon API Gateway é um conjunto de recursos e métodos. Para este tutorial, você cria um recurso (`DynamoDBManager`) e define um método (`POST`) nele. O método é respaldado por uma função do Lambda (`LambdaFunctionOverHttps`). Ou seja, quando você chama a API por meio de um endpoint HTTPS, o Amazon API Gateway invoca a função do Lambda.

O método `POST` no recurso `DynamoDBManager` oferece suporte para as seguintes operações do DynamoDB:

- Criar, atualizar e excluir um item.
- Ler um item.
- Examinar um item.
- Outras operações (echo, ping), não relacionadas ao DynamoDB, que você pode usar para testes.

A carga útil da solicitação que você envia na solicitação `POST` identifica a operação do DynamoDB e fornece os dados necessários. Por exemplo:

- Veja a seguir um exemplo de carga de solicitação para uma operação de criação de item do DynamoDB:

```
{
```

```
    "operation": "create",
    "tableName": "lambda-apigateway",
    "payload": {
        "Item": {
            "id": "1",
            "name": "Bob"
        }
    }
}
```

- Veja a seguir um exemplo de carga de solicitação para uma operação de leitura de item do DynamoDB:

```
{
    "operation": "read",
    "tableName": "lambda-apigateway",
    "payload": {
        "Key": {
            "id": "1"
        }
    }
}
```

- Veja a seguir um exemplo de carga de solicitação para uma operação echo. Em seguida, uma solicitação HTTP POST é enviada ao endpoint, usando os seguintes dados no corpo de solicitação.

```
{
    "operation": "echo",
    "payload": {
        "somekey1": "somevalue1",
        "somekey2": "somevalue2"
    }
}
```

Note

O API Gateway oferece recursos avançados, tais como:

- Transmissão de toda a solicitação – uma função do Lambda pode receber toda a solicitação HTTP (em vez de apenas o corpo da solicitação) e definir a resposta HTTP (em vez de apenas o corpo da resposta) usando o tipo de integração AWS_PROXY.
- Métodos genéricos – mapeia todos os métodos de um recurso da API para uma única função do Lambda com um único mapeamento, usando o método genérico ANY.
- Recursos genéricos – mapeia todos os subcaminhos de um recurso para uma função do Lambda sem qualquer configuração adicional usando o novo parâmetro de caminho ({proxy +}).

Para saber mais sobre esses recursos do API Gateway, consulte [Configurar a integração de proxy para um recurso de proxy](#).

Prerequisites

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em [Conceitos básicos do Lambda \(p. 9\)](#) para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, você precisa de um terminal de linha de comando ou de um shell para executar os comandos. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

Você deve ver a saída a seguir:

```
aws-cli/2.0.57 Python/3.7.4 Darwin/19.6.0 exe/x86_64
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

Criar a função de execução

Crie a [função de execução \(p. 57\)](#). Essa função usa uma política personalizada para conceder permissão à sua função para acessar os recursos necessários da AWS. Primeiro você cria a política, depois a função de execução.

Para criar uma política personalizada

1. Abra a [página da política](#) no console do IAM.
2. Selecione Create Policy (Criar política).
3. Selecione a guia JSON. Cole a seguinte política personalizada na caixa de entrada.

```
{
    "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. Escolha Next: Tags (Próximo: tags).
5. Selecione Next: Review.
6. Para o nome da política, insira **lambda-apigateway-policy**.
7. Insira Criar política.

Esta política inclui permissões para que a função acesse o DynamoDB e o CloudWatch Logs.

Para criar uma função de execução

1. Abra a [página Roles](#) (Funções) no console do IAM.
2. Selecione Create role.
3. Para a entidade confiável, escolha AWS Service (Produto da AWS), e para o caso de uso, escolha Lambda.
4. Escolha Próximo: Permissões.
5. Na caixa de pesquisa de política, insira **lambda-apigateway-policy**.
6. Nos resultados, selecione **lambda-apigateway-policy** e escolha Next: Tags (Próximo: Tags).
7. Selecione Next: Review.
8. Para o nome da função, insira **lambda-apigateway-role**.
9. Insira Criar função.

Observe o nome de recurso da Amazon (ARN) da função de execução para uso posterior.

Criar a função

O código de amostra a seguir recebe uma entrada de evento do API Gateway e processa as mensagens que ela contém. Na ilustração, o código grava alguns dos dados de entrada no CloudWatch Logs.

Note

Para o código de amostra em outras linguagens, consulte [Código de exemplo da função do \(p. 298\)](#).

Example index.js

```
console.log('Loading function');

var AWS = require('aws-sdk');
var dynamo = new AWS.DynamoDB.DocumentClient();

/**
 * Provide an event that contains the following keys:
 *
 * - operation: one of the operations in the switch statement below
 * - tableName: required for operations that interact with DynamoDB
 * - payload: a parameter to pass to the operation being performed
 */
exports.handler = function(event, context, callback) {
    //console.log('Received event:', JSON.stringify(event, null, 2));

    var operation = event.operation;

    if (event.tableName) {
        event.payload.TableName = event.tableName;
    }

    switch (operation) {
        case 'create':
            dynamo.put(event.payload, callback);
            break;
        case 'read':
            dynamo.get(event.payload, callback);
            break;
    }
}
```

```
        case 'update':
            dynamo.update(event.payload, callback);
            break;
        case 'delete':
            dynamo.delete(event.payload, callback);
            break;
        case 'list':
            dynamo.scan(event.payload, callback);
            break;
        case 'echo':
            callback(null, "Success");
            break;
        case 'ping':
            callback(null, "pong");
            break;
        default:
            callback(`Unknown operation: ${operation}`);
    }
};
```

Para criar a função

1. Copie o código de amostra em um arquivo chamado `index.js`.
2. Crie um pacote de implantação.

```
zip function.zip index.js
```

3. Crie uma função do Lambda com o comando `create-function`. Para o parâmetro `role`, insira o ARN da função de execução criada anteriormente.

```
aws lambda create-function --function-name LambdaFunctionOverHttps \
--zip-file file://function.zip --handler index.handler --runtime nodejs12.x \
--role arn:aws:iam::123456789012:role/service-role/lambda-apigateway-role
```

Testar a função do Lambda

Invoque a função manualmente usando os dados de evento de exemplo. Recomendamos que você invoque a função usando o console, pois ele fornece uma interface de usuário amigável para revisar os resultados da execução, incluindo o resumo da execução, os logs gravados pelo código e os resultados retornados pela função (porque o console sempre realiza uma execução síncrona invocando a função do Lambda com o tipo de invocação `RequestResponse`).

Para testar a função do Lambda

1. Copie o JSON a seguir em um arquivo e salve-o como `input.txt`.

```
{
    "operation": "echo",
    "payload": {
        "somekey1": "somevalue1",
        "somekey2": "somevalue2"
    }
}
```

2. Execute o seguinte comando `invoke`:

```
aws lambda invoke --function-name LambdaFunctionOverHttps \
--payload file://input.txt outputfile.txt
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

Criar uma API usando o Amazon API Gateway

Nesta etapa, você associa sua função do Lambda a um método na API que você criou usando o Amazon API Gateway e testa a experiência completa. Ou seja, quando uma solicitação HTTP for enviada para um método de API, o Amazon API Gateway invocará sua função do Lambda.

Primeiro, você cria uma API (`DynamoDBOperations`) usando o Amazon API Gateway com um recurso (`DynamoDBManager`) e um método (`POST`). Associe o método `POST` à sua função do Lambda. Em seguida, teste a experiência completa.

Criar a API

Execute o seguinte comando `create-rest-api` para criar a API `DynamoDBOperations` para este tutorial.

```
aws apigateway create-rest-api --name DynamoDBOperations
```

Você deve ver a saída a seguir:

```
{  
    "id": "bs8fqo6bp0",  
    "name": "DynamoDBOperations",  
    "createdDate": 1539803980,  
    "apiKeySource": "HEADER",  
    "endpointConfiguration": {  
        "types": [  
            "EDGE"  
        ]  
    }  
}
```

Salve o ID da API para uso em outros comandos. Você também precisa do ID do recurso raiz da API. Para obter o ID, execute o comando `get-resources`.

```
API=bs8fqo6bp0  
aws apigateway get-resources --rest-api-id $API
```

Você deve ver a saída a seguir:

```
{  
    "items": [  
        {  
            "path": "/",  
            "id": "e8kitthgdb"  
        }  
    ]  
}
```

No momento existe apenas o recurso raiz, porém mais recursos serão adicionados na próxima etapa.

Criar um recurso na API

Execute o seguinte comando `create-resource` para criar um recurso (`DynamoDBManager`) na API que você criou na seção anterior.

```
aws apigateway create-resource --rest-api-id $API --path-part DynamoDBManager \
--parent-id e8kitthgdb
```

Você deve ver a saída a seguir:

```
{
  "path": "/DynamoDBManager",
  "pathPart": "DynamoDBManager",
  "id": "iuig5w",
  "parentId": "e8kitthgdb"
}
```

Observe o ID na resposta. Esse é o ID do recurso `DynamoDBManager` que você criou.

Criar método POST no recurso

Execute o seguinte comando `put-method` para criar um método `POST` no recurso `DynamoDBManager` em sua API.

```
RESOURCE=iuig5w
aws apigateway put-method --rest-api-id $API --resource-id $RESOURCE \
--http-method POST --authorization-type NONE
```

Você deve ver a saída a seguir:

```
{
  "apiKeyRequired": false,
  "httpMethod": "POST",
  "authorizationType": "NONE"
}
```

Nós especificamos `NONE` para o parâmetro `--authorization-type`, o que significa que solicitações não autenticadas para esse método são compatíveis. Isso é aceitável para testes, mas em produção, você deve usar a autenticação baseada em chave ou função.

Definir a função do Lambda como o destino do método POST

Execute o comando a seguir para definir a função do Lambda como o ponto de integração para o método `POST`. Esse é o método que o Amazon API Gateway invoca quando você faz uma solicitação HTTP para o endpoint do método `POST`. Esse comando e outros usam ARNs que incluem seu ID de conta e região. Salve-os para variáveis (você pode encontrar o ID da sua conta na função ARN que você usou para criar a função).

```
REGION=us-east-2
ACCOUNT=123456789012
aws apigateway put-integration --rest-api-id $API --resource-id $RESOURCE \
--http-method POST --type AWS --integration-http-method POST \
--uri arn:aws:apigateway:$REGION:lambda:path/2015-03-31/functions/arn:aws:lambda:$REGION:
$ACCOUNT:function:LambdaFunctionOverHttps/invocations
```

Você deve ver a saída a seguir:

```
{
  "type": "AWS",
  "httpMethod": "POST",
  "uri": "arn:aws:apigateway:us-east-2:lambda:path/2015-03-31/functions/
arn:aws:lambda:us-east-2:123456789012:function:LambdaFunctionOverHttps/invocations",
  "passthroughBehavior": "WHEN_NO_MATCH",
```

```
        "timeoutInMillis": 29000,  
        "cacheNamespace": "iuig5w",  
        "cacheKeyParameters": []  
    }  
}
```

O `--integration-http-method` é o método que o API Gateway usa para se comunicar com o AWS Lambda. O `--uri` é o identificador exclusivo do endpoint para o qual o Amazon API Gateway pode enviar a solicitação.

Defina o content-type da resposta POST do método de resposta e resposta de integração como JSON da seguinte forma:

- Execute o comando a seguir para definir a resposta do método POST como JSON. Este é o tipo de resposta que o seu método de API retorna.

```
aws apigateway put-method-response --rest-api-id $API \
--resource-id $RESOURCE --http-method POST \
--status-code 200 --response-models application/json=Empty
```

Você deve ver a saída a seguir:

```
{  
    "statusCode": "200",  
    "responseModels": {  
        "application/json": "Empty"  
    }  
}
```

- Execute o comando a seguir para definir a resposta de integração do método POST como JSON. Este é o tipo de resposta que a função do Lambda retorna.

```
aws apigateway put-integration-response --rest-api-id $API \
--resource-id $RESOURCE --http-method POST \
--status-code 200 --response-templates application/json=""
```

Você deve ver a saída a seguir:

```
{  
    "statusCode": "200",  
    "responseTemplates": {  
        "application/json": null  
    }  
}
```

Note

Se encontrar um erro ao executar esse comando, você poderá usar caracteres de escape ao redor do campo de modelo de resposta para atribuir mais clareza. O texto `application/json=""` se torna `{"\"application/json\"": ""}"`.

Implantar a API

Nesta etapa, você implanta a API que você criou em um estágio chamado prod.

```
aws apigateway create-deployment --rest-api-id $API --stage-name prod
```

Você deve ver a saída a seguir:

```
{  
  "id": "20vgpsz",  
  "createdDate": 1539820012  
}
```

Conceder permissão de invocação à API

Agora que você tem uma API criada usando o Amazon API Gateway implantada, você pode testá-la. Primeiro, você precisa adicionar permissões para que o Amazon API Gateway possa invocar sua função do Lambda quando você enviar solicitações HTTP para o método POST.

Para fazer isso, você precisa adicionar uma permissão à política de permissões associada à sua função do Lambda. Execute o comando `add-permission` do AWS Lambda a seguir para conceder ao principal do serviço do Amazon API Gateway (`apigateway.amazonaws.com`) as permissões para invocar sua função do Lambda (`LambdaFunctionOverHttps`).

```
aws lambda add-permission --function-name LambdaFunctionOverHttps \  
--statement-id apigateway-test-2 --action lambda:InvokeFunction \  
--principal apigateway.amazonaws.com \  
--source-arn "arn:aws:execute-api:$REGION:$ACCOUNT:$API/*/POST/DynamoDBManager"
```

Você deve ver a saída a seguir:

```
{  
  "Statement": "{\"Sid\":\"apigateway-test-2\",\"Effect\":\"Allow\",\"Principal\":  
  {\"Service\":\"apigateway.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\":  
  \"arn:aws:lambda:us-east-2:123456789012:function:LambdaFunctionOverHttps\",\"Condition\":  
  \"\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:execute-api:us-east-2:123456789012:mnh1yprki7/  
  */POST/DynamoDBManager\"}}}"  
}
```

Você deve conceder essa permissão para habilitar os testes (se você acessa o Amazon API Gateway e escolher Testar para testar o método de API, você precisa dessa permissão). Observe que o `--source-arn` especifica um caractere curinga (*) como o valor de estágio (indica apenas testes). Isso permite que você teste sem implantar a API.

Note

Se a sua função e a API estiverem em regiões diferentes, o identificador de região no ARN de origem deverá corresponder à região da função, e não à região da API.

Agora, execute o mesmo comando novamente, mas desta vez conceda à sua API implantada permissões para invocar a função do Lambda.

```
aws lambda add-permission --function-name LambdaFunctionOverHttps \  
--statement-id apigateway-prod-2 --action lambda:InvokeFunction \  
--principal apigateway.amazonaws.com \  
--source-arn "arn:aws:execute-api:$REGION:$ACCOUNT:$API/prod/POST/DynamoDBManager"
```

Você deve ver a saída a seguir:

```
{  
  "Statement": "{\"Sid\":\"apigateway-prod-2\",\"Effect\":\"Allow\",\"Principal\":  
  {\"Service\":\"apigateway.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\":  
  \"arn:aws:lambda:us-east-2:123456789012:function:LambdaFunctionOverHttps\",\"Condition\":  
  \"\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:execute-api:us-east-2:123456789012:mnh1yprki7/  
  prod/POST/DynamoDBManager\"}}}"  
}
```

Você concede essa permissão para que sua API implantada tenha permissão para invocar a função do Lambda. Observe que o `--source-arn` especifica um `prod` que é o nome do estágio usado ao implantar a API.

Criar uma tabela do Amazon DynamoDB

Crie a tabela do DynamoDB usada pela função do Lambda.

Como criar uma tabela do DynamoDB

1. Abra o [console do DynamoDB](#).
2. Escolha Create table.
3. Crie uma tabela com as configurações a seguir.
 - Table name (Nome da tabela – `lambda-apigateway`)
 - Primary key (Chave primária): `id` (string)
4. Escolha Create (Criar).

Acionar a função com uma solicitação HTTP

Nesta etapa, você está pronto para enviar uma solicitação HTTP para o endpoint do método `POST`. Você pode usar Curl ou um método (`testInvokeMethod`) fornecido pelo Amazon API Gateway.

Você pode usar os comandos da CLI do Amazon API Gateway para enviar uma solicitação HTTP `POST` para o endpoint do recurso (`DynamoDBManager`). Como você implantou seu Amazon API Gateway, pode usar o Curl para invocar os métodos para a mesma operação.

A função do Lambda oferece suporte para o uso da operação `create` para criar um item em sua tabela do DynamoDB. Para solicitar essa operação, use o seguinte JSON:

Example `create-item.json`

```
{  
    "operation": "create",  
    "tableName": "lambda-apigateway",  
    "payload": {  
        "Item": {  
            "id": "1234ABCD",  
            "number": 5  
        }  
    }  
}
```

Salve a entrada de teste em um arquivo chamado `create-item.json`. Execute o comando `testInvokeMethod` do Amazon API Gateway para enviar a solicitação do método HTTP `POST` para o endpoint do recurso (`DynamoDBManager`).

```
aws apigateway testInvokeMethod --rest-api-id $API \  
--resource-id $RESOURCE --http-method POST --path-with-query-string "" \  
--body file://create-item.json
```

Ou então, você pode usar o seguinte comando Curl:

```
curl -X POST -d "{\"operation\":\"create\", \"tableName\":\"lambda-apigateway\",  
\"payload\":{\"Item\":{\"id\":\"1\", \"name\":\"Bob\"}}}" https://$API.execute-api.  
$REGION.amazonaws.com/prod/DynamoDBManager
```

Para enviar a solicitação para a operação echo compatível com sua função do Lambda, você pode usar a seguinte carga útil de solicitação:

Example echo.json

```
{  
  "operation": "echo",  
  "payload": {  
    "somekey1": "somevalue1",  
    "somekey2": "somevalue2"  
  }  
}
```

Salve a entrada de teste em um arquivo chamado echo.json. Execute o comando `test-invoke-method` da CLI do Amazon API Gateway para enviar uma solicitação de método HTTP POST ao endpoint do recurso (DynamoDBManager) usando o JSON anterior no corpo na solicitação.

```
aws apigateway test-invoke-method --rest-api-id $API \  
--resource-id $RESOURCE --http-method POST --path-with-query-string "" \  
--body file://echo.json
```

Ou então, você pode usar o seguinte comando Curl:

```
curl -X POST -d "{\"operation\":\"echo\", \"payload\":{\"somekey1\":\"somevalue1\",  
\"somekey2\":\"somevalue2\"}}" https://$API.execute-api.$REGION.amazonaws.com/prod/  
DynamoDBManager
```

Limpar recursos da

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua conta da AWS.

Como excluir a função do Lambda

1. Abra a página [Functions](#) (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Actions, Delete.
4. Escolha Delete.

Para excluir a função de execução

1. Abra a página [Roles](#) (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Selecione Delete role (Excluir função).
4. Escolha Sim, excluir.

Para excluir a API

1. Abra a página [APIs](#) do console do API Gateway.
2. Selecione a API que você criou.
3. Escolha Actions, Delete.

4. Escolha Delete.

Para excluir uma tabela do DynamoDB

1. Abra a [página Tables](#) (Tabelas) no console do DynamoDB.
2. Selecione a tabela que você criou.
3. Escolha Delete.
4. Digite **delete** na caixa de texto.
5. Escolha Delete.

Código de exemplo da função do

O código de amostra está disponível para as seguintes linguagens.

Tópicos

- [Node.js \(p. 298\)](#)
- [Python 3 \(p. 299\)](#)
- [Go \(p. 300\)](#)

Node.js

O exemplo a seguir processa mensagens do API Gateway e gerencia documentos do DynamoDB com base no método de solicitação.

Example index.js

```
console.log('Loading function');

var AWS = require('aws-sdk');
var dynamo = new AWS.DynamoDB.DocumentClient();

/**
 * Provide an event that contains the following keys:
 *
 * - operation: one of the operations in the switch statement below
 * - tableName: required for operations that interact with DynamoDB
 * - payload: a parameter to pass to the operation being performed
 */
exports.handler = function(event, context, callback) {
    //console.log('Received event:', JSON.stringify(event, null, 2));

    var operation = event.operation;

    if (event.tableName) {
        event.payload.TableName = event.tableName;
    }

    switch (operation) {
        case 'create':
            dynamo.put(event.payload, callback);
            break;
        case 'read':
            dynamo.get(event.payload, callback);
            break;
        case 'update':
            dynamo.update(event.payload, callback);
            break;
        case 'delete':
            dynamo.delete(event.payload, callback);
            break;
    }
}
```

```
        break;
    case 'delete':
        dynamo.delete(event.payload, callback);
        break;
    case 'list':
        dynamo.scan(event.payload, callback);
        break;
    case 'echo':
        callback(null, "Success");
        break;
    case 'ping':
        callback(null, "pong");
        break;
    default:
        callback(`Unknown operation: ${operation}`);
}
};
```

Compaca o código do exemplo para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Node.js com arquivos .zip \(p. 517\)](#).

Python 3

O exemplo a seguir processa mensagens do API Gateway e gerencia documentos do DynamoDB com base no método de solicitação.

Example LambdaFunctionOverHttps.py

```
from __future__ import print_function

import boto3
import json

print('Loading function')


def handler(event, context):
    '''Provide an event that contains the following keys:
       - operation: one of the operations in the operations dict below
       - tableName: required for operations that interact with DynamoDB
       - payload: a parameter to pass to the operation being performed
    '''
    #print("Received event: " + json.dumps(event, indent=2))

    operation = event['operation']

    if 'tableName' in event:
        dynamo = boto3.resource('dynamodb').Table(event['tableName'])

    operations = {
        'create': lambda x: dynamo.put_item(**x),
        'read': lambda x: dynamo.get_item(**x),
        'update': lambda x: dynamo.update_item(**x),
        'delete': lambda x: dynamo.delete_item(**x),
        'list': lambda x: dynamo.scan(**x),
        'echo': lambda x: x,
        'ping': lambda x: 'pong'
    }

    if operation in operations:
        return operations[operation](event.get('payload'))
    else:
```

```
raise ValueError('Unrecognized operation "{}".format(operation))
```

Compaca o código do exemplo para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Python com arquivos .zip \(p. 543\)](#).

Go

O exemplo a seguir processa mensagens do API Gateway e registra informações sobre a solicitação.

Example LambdaFunctionOverHttps.go

```
import (
    "context"
    "fmt"
    "github.com/aws/aws-lambda-go/events"
    runtime "github.com/aws/aws-lambda-go/runtime"
)

func main() {
    runtime.Start(handleRequest)
}

func handleRequest(ctx context.Context, request events.APIGatewayProxyRequest) (events.APIGatewayProxyResponse, error) {
    fmt.Printf("Processing request data for request %s.\n",
    request.RequestContext.RequestID)
    fmt.Printf("Body size = %d.\n", len(request.Body))

    fmt.Println("Headers:")
    for key, value := range request.Headers {
        fmt.Printf("    %s: %s\n", key, value)
    }

    return events.APIGatewayProxyResponse{Body: request.Body, StatusCode: 200}, nil
}
```

Crie o executável com o `go build` e crie um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Go com arquivos .zip \(p. 640\)](#).

Criar um microsserviço simples usando o Lambda e o API Gateway

Neste tutorial, você usará o console do Lambda para criar uma função do Lambda e um endpoint do Amazon API Gateway para acionar essa função. Você será capaz de chamar o endpoint com qualquer método (GET, POST, PATCH, etc.) para acionar sua função do Lambda. Quando o endpoint for chamado, toda a solicitação será transmitida para sua função do Lambda. A ação de sua função dependerá do método com o qual você chamar o endpoint:

- DELETE: excluir um item de uma tabela do DynamoDB
- GET: varrer tabela e retornar todos os itens
- POST: criar um item
- PUT: atualizar um item

Criar uma API usando o Amazon API Gateway

Siga as etapas nesta seção para criar uma nova função do Lambda e um endpoint do API Gateway para acioná-la:

Como criar uma API

1. Faça login no Console de Gerenciamento da AWS e abra o console do AWS Lambda.
2. Escolha Create Lambda function.
3. Selecione Use a blueprint (Usar um esquema).
4. Insira **microservice** na barra de pesquisa. Escolha o esquema microservice-http-endpoint.
5. Defina a função com as configurações a seguir.
 - Nome – **lambda-microservice**.
 - Role (Função): Create a new role from AWS policy templates.
 - Nome da função – **lambda-apigateway-role**.
 - Policy templates (Modelos de política): Simple microservice permissions.
 - API – Criar uma API.
 - Tipo de API – API de HTTP.
 - Security (Segurança): Open.

Escolha Create function.

Quando você concluir o assistente e criar sua função, o Lambda criará um recurso de proxy chamado **lambda-microservice** sob o nome da API que você selecionou. Para obter mais informações sobre recursos de proxy, consulte [Configurar a integração de proxy para um recurso de proxy](#).

Um recurso de proxy tem um tipo de integração **AWS_PROXY** e um método genérico **ANY**. O tipo de integração **AWS_PROXY** aplica um modelo de mapeamento padrão para transmitir toda a solicitação à função do Lambda e transforma a saída da função do Lambda em respostas HTTP. O método **ANY** define a mesma configuração de integração para todos os métodos compatíveis, incluindo **GET**, **POST**, **PATCH**, **DELETE** e outros.

Testar o envio de uma solicitação HTTP

Neste exercício, você usa o console para testar manualmente a função do Lambda. Além disso, você pode executar um comando `curl` para testar a experiência completa. Ou seja, [enviar uma solicitação HTTP](#) para o seu método de API e fazer com que o Amazon API Gateway invoque sua função do Lambda. Para concluir as etapas, certifique-se de ter criado uma tabela do DynamoDB chamada "MyTable". Para obter mais informações, consulte [Criar uma tabela do DynamoDB com uma transmissão habilitada \(p. 348\)](#).

Como testar a API

1. Com sua função do **lambda-microservice** ainda aberta no console do Lambda, escolha a guia Test (Testar).
2. Escolha New event (Novo evento).
3. Escolha o modelo Hello World.
4. Em Name (Nome), insira um nome para o evento de teste.
5. No painel de entrada de texto, substitua o texto existente pelo seguinte:

```
{  
  "httpMethod": "GET",  
  "queryStringParameters": {  
    "TableName": "MyTable"  
  }  
}
```

Esse comando **GET** examina sua tabela do DynamoDB e retorna todos os itens encontrados.

6. Depois de inserir o texto acima, escolha Test (Testar).

Verifique se o teste teve êxito. Você deverá ver a seguinte resposta da função:

```
{  
  "statusCode": "200",  
  "body": "{\"Items\":[],\"Count\":0,\"ScannedCount\":0}",  
  "headers": {  
    "Content-Type": "application/json"  
  }  
}
```

Modelo do AWS SAM para uma aplicação do API Gateway

Veja abaixo um modelo de exemplo do AWS SAM para a aplicação do Lambda do [tutorial \(p. 287\)](#).

Copie o texto abaixo para um arquivo e salve-o ao lado do pacote ZIP criado previamente. Observe que os valores dos parâmetros Handler e Runtime devem corresponder àqueles usados quando você criou a função na seção anterior.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Resources:  
  LambdaFunctionOverHttps:  
    Type: AWS::Serverless::Function  
    Properties:  
      Handler: index.handler  
      Runtime: nodejs12.x  
      Policies: AmazonDynamoDBFullAccess  
      Events:  
        HttpPost:  
          Type: Api  
          Properties:  
            Path: '/DynamoDBOperations/DynamoDBManager'  
            Method: post
```

Para obter informações sobre como empacotar e implantar o aplicativo sem servidor usando os comandos de empacotamento e implantação, consulte [Implantar aplicativos sem servidor](#) no Guia do desenvolvedor do AWS Serverless Application Model.

Usar o AWS Lambda com o AWS CloudTrail

O AWS CloudTrail é um serviço que fornece um registro das ações realizadas por um usuário, função ou serviço da AWS. O CloudTrail captura as chamadas de API como eventos. Para obter um registro contínuo de eventos em sua conta da AWS, crie uma trilha. Uma trilha permite que o CloudTrail entregue arquivos de log de eventos a um bucket do Amazon S3.

Você pode aproveitar o recurso de notificação do bucket do Amazon S3 e direcionar o Amazon S3 para publicar eventos criados por objetos no AWS Lambda. Sempre que o CloudTrail gravar logs no bucket do S3, o Amazon S3 poderá invocar a função do Lambda passando o evento criado por objeto do Amazon S3 como um parâmetro. O evento do S3 fornece informações, incluindo o nome do bucket e o nome da chave do objeto de log que o CloudTrail criou. O código da função do Lambda pode ler o objeto de log e processar os registros de acesso realizados pelo CloudTrail. Por exemplo, você pode escrever um código de função do Lambda para ser notificado sobre a chamada de API específica realizada em sua conta.

Nesse cenário, o CloudTrail grava logs de acesso no bucket do S3. Quanto ao AWS Lambda, o Amazon S3 é a origem do evento, portanto, o Amazon S3 publica eventos no AWS Lambda e invoca a função do Lambda.

Example Log do CloudTrail

```
{
  "Records": [
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "Root",
        "principalId": "123456789012",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "access-key-id",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2015-01-24T22:41:54Z"
          }
        }
      },
      "eventTime": "2015-01-24T23:26:50Z",
      "eventSource": "sns.amazonaws.com",
      "eventName": "CreateTopic",
      "awsRegion": "us-east-2",
      "sourceIPAddress": "205.251.233.176",
      "userAgent": "console.amazonaws.com",
      "requestParameters": {
        "name": "dropmeplease"
      },
      "responseElements": {
        "topicArn": "arn:aws:sns:us-east-2:123456789012:exampletopic"
      },
      "requestID": "3fdb7834-9079-557e-8ef2-350abc03536b",
      "eventID": "17b46459-dada-4278-b8e2-5a4ca9ff1a9c",
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    },
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "Root",
        "principalId": "123456789012",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "access-key-id",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2015-01-24T22:41:54Z"
          }
        }
      }
    }
  ]
}
```

```
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext":{
            "attributes":{
                "mfaAuthenticated":"false",
                "creationDate":"2015-01-24T22:41:54Z"
            }
        },
        "eventTime":"2015-01-24T23:27:02Z",
        "eventSource":"sns.amazonaws.com",
        "eventName":"GetTopicAttributes",
        "awsRegion":"us-east-2",
        "sourceIPAddress":"205.251.233.176",
        "userAgent":"console.amazonaws.com",
        "requestParameters":{
            "topicArn":"arn:aws:sns:us-east-2:123456789012:exampletopic"
        },
        "responseElements":null,
        "requestID":"4a0388f7-a0af-5df9-9587-c5c98c29cbe",
        "eventID":"ec5bb073-8fa1-4d45-b03c-f07b9fc9ea18",
        "eventType":"AwsApiCall",
        "recipientAccountId":"123456789012"
    }
]
```

Para obter informações detalhadas sobre como configurar o Amazon S3 como fonte de eventos, consulte [Usar o AWS Lambda com o Amazon S3 \(p. 431\)](#).

Tópicos

- [Registrar em log chamadas de API do AWS Lambda com o AWS CloudTrail \(p. 305\)](#)
- [Tutorial: acionar uma função do Lambda com eventos do AWS CloudTrail \(p. 308\)](#)
- [Código de exemplo da função do \(p. 313\)](#)

Registrar em log chamadas de API do AWS Lambda com o AWS CloudTrail

O AWS Lambda é integrado ao AWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, uma função ou um serviço da AWS no AWS Lambda. O CloudTrail captura as chamadas de API do AWS Lambda como eventos. As chamadas capturadas incluem as chamadas do console do AWS Lambda e as chamadas de código para as operações da API do AWS Lambda. Se você criar uma trilha, poderá habilitar a entrega contínua de eventos do CloudTrail para um bucket do Amazon S3, incluindo eventos para o AWS Lambda. Se você não configurar uma trilha, ainda poderá visualizar os eventos mais recentes no console do CloudTrail em Event history (Histórico de eventos). Usando as informações coletadas pelo CloudTrail, é possível determinar a solicitação feita para o AWS Lambda, o endereço IP no qual a solicitação foi feita, quem fez a solicitação, quando ela foi feita, além de detalhes adicionais.

Para saber mais sobre o CloudTrail, incluindo como configurá-lo e ativá-lo, consulte o [Manual do usuário do AWS CloudTrail](#).

Informações do AWS Lambda no CloudTrail

O CloudTrail é habilitado em sua conta da AWS quando ela é criada. Quando a atividade do evento compatível ocorrer no AWS Lambda, ela será registrada em um evento do CloudTrail juntamente com outros eventos de serviços da AWS no Event history (Histórico de eventos). Você pode visualizar, pesquisar e fazer download de eventos recentes em sua conta da AWS. Para obter mais informações, consulte [Visualizar eventos com o histórico de eventos do CloudTrail](#).

Para obter um registro contínuo de eventos na sua conta da AWS, incluindo eventos para o AWS Lambda, crie uma trilha. Uma trilha permite que o CloudTrail entregue arquivos de log a um bucket do Amazon S3. Por padrão, quando você cria uma trilha no console, ela é aplicada a todas as regiões da AWS. A trilha registra em log eventos de todas as regiões na partição da AWS e entrega os arquivos de log para o bucket do Amazon S3 especificado por você. Além disso, você pode configurar outros serviços da AWS para analisar mais profundamente e agir sobre os dados de eventos coletados nos logs do CloudTrail. Para obter mais informações, consulte:

- [Visão geral da criação de uma trilha](#)
- [Serviços e integrações compatíveis com o CloudTrail](#)
- [Configurar notificações do Amazon SNS para o CloudTrail](#)
- [Receiving CloudTrail log files from multiple regions](#) e [Receiving CloudTrail log files from multiple accounts](#)

O AWS Lambda é compatível com as seguintes ações como eventos nos arquivos de log do CloudTrail:

- [AddLayerVersionPermission \(p. 771\)](#)
- [AddPermission \(p. 775\)](#)
- [CreateEventSourceMapping \(p. 786\)](#)
- [CreateFunction \(p. 796\)](#)

(O parâmetro `ZipFile` é omitido dos logs do CloudTrail para `CreateFunction`.)

- [DeleteEventSourceMapping \(p. 812\)](#)
- [DeleteFunction \(p. 818\)](#)
- [GetEventSourceMapping \(p. 837\)](#)
- [GetFunction \(p. 842\)](#)
- [GetFunctionConfiguration \(p. 851\)](#)

- [GetLayerVersionPolicy \(p. 867\)](#)
- [GetPolicy \(p. 869\)](#)
- [ListEventSourceMappings \(p. 888\)](#)
- [listFunctions \(p. 894\)](#)
- [RemovePermission \(p. 942\)](#)
- [UpdateEventSourceMapping \(p. 956\)](#)
- [UpdateFunctionCode \(p. 965\)](#)

(O parâmetro `ZipFile` é omitido dos logs do CloudTrail para `UpdateFunctionCode`.)

- [UpdateFunctionConfiguration \(p. 974\)](#)

Cada entrada de log contém informações sobre quem gerou a solicitação. As informações sobre identidade do usuário no registro ajudam a determinar se a solicitação foi feita com credenciais de usuário raiz ou do IAM;, com credenciais de segurança temporárias para uma função ou um usuário federado ou por outro serviço da AWS. Para obter mais informações, consulte o campo `userIdentity` na [Referência de evento do CloudTrail](#).

Você pode armazenar os arquivos de log no bucket pelo tempo que desejar, mas também pode definir regras do ciclo de vida do Amazon S3 para arquivar ou excluir os arquivos de log automaticamente. Por padrão, os arquivos de log são criptografados usando-se Server-Side Encryption (SSE - Criptografia do lado do servidor) do Amazon S3.

É possível optar por ter as notificações do CloudTrail publicadas pelo Amazon SNS quando os arquivos de log novos forem entregues caso você queira agir rapidamente após a entrega do arquivo de log. Para obter mais informações, consulte [Configurar notificações do Amazon SNS para o CloudTrail](#).

Você também pode agregar arquivos de log do AWS Lambda de várias regiões da AWS e de várias contas da AWS em um único bucket do S3. Para obter mais informações, consulte [Trabalhar com arquivos de log do CloudTrail](#).

Noções básicas sobre entradas de arquivos de log do AWS Lambda

Os arquivos de log do CloudTrail podem conter uma ou mais entradas de log, e cada uma é composta por vários eventos em formato JSON. Uma entrada de log representa uma única solicitação de qualquer origem e inclui informações sobre a ação solicitada, quaisquer parâmetros, a data e hora da ação e assim por diante. Não é garantido que as entradas de log estejam em uma ordem específica. Ou seja, elas não são um rastreamento ordenado das chamadas de API públicas.

O exemplo a seguir mostra entradas de log do CloudTrail que demonstram as ações `GetFunction` e `DeleteFunction`.

```
{  
  "Records": [  
    {  
      "eventVersion": "1.03",  
      "userIdentity": {  
        "type": "IAMUser",  
        "principalId": "A1B2C3D4E5F6G7EXAMPLE",  
        "arn": "arn:aws:iam::999999999999:user/myUserName",  
        "accountId": "999999999999",  
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",  
        "userName": "myUserName"  
      },  
      "eventTime": "2015-03-18T19:03:36Z",  
      "eventSource": "AWS_Lambda",  
      "eventName": "DeleteFunction",  
      "awsRegion": "us-east-1",  
      "invokedFunctionArn": "arn:aws:lambda:us-east-1:999999999999:function:myLambda",  
      "awsRequestID": "12345678901234567890123456789012",  
      "logEvent": "2015-03-18T19:03:36.000Z myLambda myFunction 12345678901234567890123456789012: DeleteFunction: myFunction  
    }  
  ]  
}
```

```
        "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::999999999999:user/myUserName is not
authorized to perform: lambda:GetFunction on resource: arn:aws:lambda:us-
west-2:999999999999:function:other-acct-function",
        "requestParameters": null,
        "responseElements": null,
        "requestID": "7aebcd0f-cda1-11e4-aaa2-e356da31e4ff",
        "eventId": "e92a3e85-8ecd-4d23-8074-843aabfe89bf",
        "eventType": "AwsApiCall",
        "recipientAccountId": "999999999999"
    },
    {
        "eventVersion": "1.03",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "A1B2C3D4E5F6G7EXAMPLE",
            "arn": "arn:aws:iam::999999999999:user/myUserName",
            "accountId": "999999999999",
            "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
            "userName": "myUserName"
        },
        "eventTime": "2015-03-18T19:04:42Z",
        "eventSource": "lambda.amazonaws.com",
        "eventName": "DeleteFunction",
        "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": "999999999999"
    }
]
}
```

Note

`eventName` pode incluir informações como data e versão, como `"GetFunction20150331"`, mas ainda está se referindo à mesma API pública. Para obter mais informações, consulte [Serviços compatíveis com o histórico de eventos do CloudTrail](#) no Guia do usuário do AWS CloudTrail.

Usar o CloudTrail para acompanhar invocações de função

O CloudTrail também registra eventos de dados. Você pode ativar o registro de eventos de dados para registrar um evento toda vez que as funções do Lambda forem invocadas. Isso ajuda você a entender quais identidades estão invocando as funções e a frequência das invocações. Para obter mais informações sobre essa opção, consulte [Registrar em log eventos de dados para trilhas](#).

Usar o CloudTrail para solucionar problemas de fontes de eventos desabilitadas

Um evento de dados que pode ser encontrado é um evento `LambdaESMDisabled`. Há cinco categorias gerais de erro associadas a esse evento:

RESOURCE_NOT_FOUND

O recurso especificado na solicitação não existe.

FUNCTION_NOT_FOUND

A função anexada à fonte de evento não existe.

REGION_NAME_NOT_VALID

O nome de uma região fornecido para a função ou para a fonte de evento é inválido.

AUTHORIZATION_ERROR

As permissões não foram definidas ou estão configuradas incorretamente.

FUNCTION_IN_FAILED_STATE

O código da função não compila, encontrou uma exceção irrecuperável ou ocorreu uma implantação incorreta.

Esses erros são incluídos na mensagem de evento do CloudTrail na entidade `serviceEventDetails`.

Example `serviceEventDetails` Entidade

```
"serviceEventDetails":{  
    "ESMDisableReason":"Lambda Function not found"  
}
```

Tutorial: acionar uma função do Lambda com eventos do AWS CloudTrail

É possível configurar o Amazon S3 para publicar eventos no AWS Lambda quando o AWS CloudTrail armazenar logs de chamadas de API. A função do Lambda pode ler o objeto de log e processar os registros de acesso realizados pelo CloudTrail.

Use as instruções a seguir para criar uma função do Lambda que notifica você quando uma chamada de API específica é feita em sua conta. A função processa eventos de notificação do Amazon S3, lê logs de um bucket e publica alertas por meio de um tópico do Amazon SNS. Para este tutorial, você cria:

- Uma trilha do CloudTrail e um bucket do S3 no qual salvar logs.
- Um tópico do Amazon SNS para publicar notificações de alerta.
- Uma função de usuário do IAM com permissões para ler itens de um bucket do S3 e gravar logs no Amazon CloudWatch.
- Uma função Lambda que processa logs do CloudTrail e envia uma notificação sempre que um tópico do Amazon SNS é criado.

Requirements

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em [Conceitos básicos do Lambda \(p. 9\)](#) para criar sua primeira função do Lambda.

Antes de começar, verifique se você tem as seguintes ferramentas:

- [Node.js 12.x com npm](#).
- O shell Bash. Para Linux e macOS, isso está incluso por padrão. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

- A AWS CLI.

Etapa 1: criar uma trilha no CloudTrail

Quando você cria uma trilha, o CloudTrail registra as chamadas de API em arquivos de log e as armazena no Amazon S3. Um log do CloudTrail é uma matriz de eventos não ordenados no formato JSON. Para cada chamada a uma ação de API com suporte, o CloudTrail registra informações sobre a solicitação e a entidade que fez a chamada. Os eventos de log incluem o nome da ação, os parâmetros, valores de resposta e detalhes sobre o solicitante.

Para criar uma trilha

1. Abra a página [Trails \(Trilhas\) do console do CloudTrail](#).
2. Escolha Create Trail (Criar trilha).
3. Em Trail name (Nome da trilha), insira um nome.
4. Em S3 bucket (Bucket do S3), insira um nome.
5. Escolha Create (Criar).
6. Salve o nome do recurso da Amazon (ARN) do bucket para adicioná-lo à função de execução do IAM, que será criada posteriormente.

Etapa 2: criar um tópico do Amazon SNS

Crie um tópico do Amazon SNS para enviar uma notificação quando novos eventos de objeto tiverem ocorrido.

Para criar um tópico

1. Abra a página [Topics \(Tópicos\) no console do Amazon SNS](#).
2. Escolha Create topic.
3. Em Topic name (Nome do tópico), insira um nome.
4. Escolha Create topic.
5. Registre o ARN do tópico. Você precisará dele ao criar a função de execução do IAM e a função do Lambda.

Etapa 3: criar uma função de execução do IAM

Uma [função de execução \(p. 57\)](#) concede à sua função permissão para acessar recursos da AWS. Crie uma função de execução que conceda à função permissão para acessar o CloudWatch Logs, o Amazon S3 e o Amazon SNS.

Para criar uma função de execução

1. Abra a página [Roles \(Funções\) no console do IAM](#).
2. Selecione Create role.
3. Crie uma função com as seguintes propriedades:
 - Em Trusted entity (Entidade confiável), selecione Lambda.
 - Em Role name (Nome da função), insira **lambda-cloudtrail-role**.
 - Em Permissions (Permissões), crie uma política personalizada com as instruções a seguir. Substitua os valores destacados pelos nomes do seu bucket e do tópico.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:*"  
            ],  
            "Resource": "arn:aws:logs:*:*:*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Resource": "arn:aws:s3:::my-bucket/*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sns:Publish"  
            ],  
            "Resource": "arn:aws:sns:us-west-2:123456789012:my-topic"  
        }  
    ]  
}
```

4. Anote o ARN da função. Você precisará dele ao criar a função do Lambda.

Etapa 4: criar a função do Lambda

A função Lambda a seguir processa logs do CloudTrail e envia uma notificação por meio do Amazon SNS quando um tópico do Amazon SNS é criado.

Para criar a função

1. Crie uma pasta e atribua um nome que indique que ela é sua função do Lambda (por exemplo, *lambda-cloudtrail*).
2. Na pasta, crie um arquivo chamado `index.js`.
3. Cole o seguinte código em `index.js`. Substitua o ARN do tópico do Amazon SNS pelo ARN que o Amazon S3 criou quando você criou o tópico do Amazon SNS do.

```
var aws = require('aws-sdk');  
var zlib = require('zlib');  
var async = require('async');  
  
var EVENT_SOURCE_TO_TRACK = '/sns.amazonaws.com/';  
var EVENT_NAME_TO_TRACK = '/CreateTopic/';  
var DEFAULT_SNS_REGION = 'us-east-2';  
var SNS_TOPIC_ARN = 'arn:aws:sns:us-west-2:123456789012:my-topic';  
  
var s3 = new aws.S3();  
var sns = new aws.SNS({  
    apiVersion: '2010-03-31',  
    region: DEFAULT_SNS_REGION  
});  
  
exports.handler = function(event, context, callback) {  
    var srcBucket = event.Records[0].s3.bucket.name;  
    var srcKey = event.Records[0].s3.object.key;
```

```
async.waterfall([
    function fetchLogFromS3(next){
        console.log('Fetching compressed log from S3...');
        s3.getObject({
            Bucket: srcBucket,
            Key: srcKey
        },
        next);
    },
    function uncompressLog(response, next){
        console.log("Uncompressing log...");
        zlib.gunzip(response.Body, next);
    },
    function publishNotifications(jsonBuffer, next) {
        console.log('Filtering log...');
        var json = jsonBuffer.toString();
        console.log('CloudTrail JSON from S3:', json);
        var records;
        try {
            records = JSON.parse(json);
        } catch (err) {
            next('Unable to parse CloudTrail JSON: ' + err);
            return;
        }
        var matchingRecords = records
            .Records
            .filter(function(record) {
                return record.eventSource.match(EVENT_SOURCE_TO_TRACK)
                    && record.eventName.match(EVENT_NAME_TO_TRACK);
            });

        console.log('Publishing ' + matchingRecords.length + ' notification(s) in parallel...');
        async.each(
            matchingRecords,
            function(record, publishComplete) {
                console.log('Publishing notification: ', record);
                sns.publish({
                    Message:
                        'Alert... SNS topic created: \n TopicARN=' +
                    record.responseElements.topicArn + '\n\n' +
                        JSON.stringify(record),
                    TopicArn: SNS_TOPIC_ARN
                }, publishComplete);
            },
            next
        );
    },
], function (err) {
    if (err) {
        console.error('Failed to publish notifications: ', err);
    } else {
        console.log('Successfully published all notifications.');
    }
    callback(null,"message");
});
};
```

4. Na pasta `lambda-cloudtrail` execute o script a seguir. Isso cria um arquivo `package-lock.json` e uma pasta `node_modules`, que lida com todas as dependências.

```
npm install async
```

5. Execute o script a seguir para criar um pacote de implantação.

```
zip -r function.zip .
```

6. Crie uma função do Lambda chamada CloudTrailEventProcessing com o comando `create-function` executando o script a seguir. Faça as substituições indicadas.

```
aws lambda create-function --function-name CloudTrailEventProcessing \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs12.x --timeout
10 --memory-size 1024 \
--role arn:aws:iam::123456789012:role/lambda-cloudtrail-role
```

Etapa 5: adicionar permissões à política da função do Lambda

A política de recursos da função do Lambda precisa de permissões para permitir que o Amazon S3 invoque a função.

Como conceder ao Amazon S3 permissões para invocar a função

1. Execute o seguinte comando `add-permission`. Substitua o ARN e o ID da conta pelos seus.

```
aws lambda add-permission --function-name CloudTrailEventProcessing \
--statement-id Id-1 --action "lambda:InvokeFunction" --principal s3.amazonaws.com \
--source-arn arn:aws:s3:::my-bucket \
--source-account 123456789012
```

Esse comando concede ao principal do serviço Amazon S3 (`s3.amazonaws.com`) permissões para executar a ação `lambda:InvokeFunction`. As permissões de invocação serão concedidas ao Amazon S3 somente se as seguintes condições forem atendidas:

- O CloudTrail armazena um objeto de log no bucket especificado.
 - O bucket é de propriedade da conta especificada da AWS. Se o proprietário do bucket exclui um bucket, outra conta da AWS pode criar um bucket com o mesmo nome. Essa condição garante que somente a conta especificada da AWS possa invocar a função do Lambda.
2. Para visualizar a política de acesso da função do Lambda, execute o comando `get-policy` a seguir e substitua o nome da função.

```
aws lambda get-policy --function-name function-name
```

Etapa 6: configurar notificações em um bucket do Amazon S3

Para solicitar que o Amazon S3 publique eventos criados por objetos no Lambda, adicione uma configuração de notificação ao bucket do S3. Na página de configuração, você deve especificar o seguinte:

- Tipo de evento: quaisquer tipos de evento que criam objetos.
- Função Lambda — a função Lambda que você deseja que o invoque o Amazon S3 do.

Como configurar notificações

1. Abra o [console do Amazon S3](#).
2. Escolha o bucket de origem.
3. Escolha Properties (Propriedades).
4. Em Events (Eventos), configure uma notificação com as seguintes configurações:

- Nome – **lambda-trigger**
- Events (Eventos – **All object create events**)
- Send to (Enviar para – **Lambda function**)
- Lambda – **CloudTrailEventProcessing**

Quando o CloudTrail armazena logs no bucket, o Amazon S3 envia um evento para a função. O evento fornece informações, incluindo o nome do bucket e o nome da chave do objeto de log que o CloudTrail criou.

Código de exemplo da função do

O código de amostra está disponível para as seguintes linguagens.

Tópicos

- [Node.js \(p. 313\)](#)

Node.js

O exemplo a seguir processa os logs do CloudTrail e envia uma notificação quando um tópico do Amazon SNS foi criado.

Example index.js

```
var aws = require('aws-sdk');
var zlib = require('zlib');
var async = require('async');

var EVENT_SOURCE_TO_TRACK = /sns.amazonaws.com/;
var EVENT_NAME_TO_TRACK = /CreateTopic/;
var DEFAULT_SNS_REGION = 'us-west-2';
var SNS_TOPIC_ARN = 'The ARN of your SNS topic';

var s3 = new aws.S3();
var sns = new aws.SNS({
    apiVersion: '2010-03-31',
    region: DEFAULT_SNS_REGION
});

exports.handler = function(event, context, callback) {
    var srcBucket = event.Records[0].s3.bucket.name;
    var srcKey = event.Records[0].s3.object.key;

    async.waterfall([
        function fetchLogFromS3(next){
            console.log('Fetching compressed log from S3...');
            s3.getObject({
                Bucket: srcBucket,
                Key: srcKey
            },
            next);
        },
        function uncompressLog(response, next){
            console.log("Uncompressing log...");
            zlib.gunzip(response.Body, next);
        },
        function publishNotifications(jsonBuffer, next) {
            console.log('Filtering log...');
        }
    ],
    function(err, results) {
        if (err) {
            return callback(err);
        }
        // Publish notifications
        sns.publish({
            TopicArn: SNS_TOPIC_ARN,
            Message: jsonBuffer
        }, function(err, response) {
            if (err) {
                return callback(err);
            }
            callback(null, response);
        });
    });
}
```

```
var json = jsonBuffer.toString();
console.log('CloudTrail JSON from S3:', json);
var records;
try {
    records = JSON.parse(json);
} catch (err) {
    next('Unable to parse CloudTrail JSON: ' + err);
    return;
}
var matchingRecords = records
    .Records
    .filter(function(record) {
        return record.eventSource.match(EVENT_SOURCE_TO_TRACK)
            && record.eventName.match(EVENT_NAME_TO_TRACK);
    });
console.log('Publishing ' + matchingRecords.length + ' notification(s) in
parallel...');
async.each(
    matchingRecords,
    function(record, publishComplete) {
        console.log('Publishing notification: ', record);
        sns.publish({
            Message:
                'Alert... SNS topic created: \n TopicARN=' +
            record.responseElements.topicArn + '\n\n' +
                JSON.stringify(record),
            TopicArn: SNS_TOPIC_ARN
        }, publishComplete);
    },
    next
);
},
function (err) {
    if (err) {
        console.error('Failed to publish notifications: ', err);
    } else {
        console.log('Successfully published all notifications.');
    }
    callback(null,"message");
});
};
```

Compre o código do exemplo para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Node.js com arquivos .zip \(p. 517\)](#).

O uso do AWS Lambda Com Amazon CloudWatch Events

O [Amazon CloudWatch Events](#) ajuda a responder às alterações de estado nos recursos da AWS. Quando seus recursos mudam de estado, enviam eventos automaticamente em um fluxo de evento. Você pode criar regras que correspondam a eventos selecionados no fluxo e roteá-los para sua função de AWS Lambda para execução de uma ação. Por exemplo, é possível invocar uma função do AWS Lambda automaticamente para registrar em log o estado de uma [Instância do EC2](#) ou do [grupo de AutoScaling](#).

O CloudWatch Events invoca a função de forma assíncrona com um documento de evento que encapsula o evento de sua origem. O exemplo a seguir mostra um evento originado de um snapshot de banco de dados no Amazon Relational Database Service.

Example CloudWatch Events

```
{  
    "version": "0",  
    "id": "fe8d3c65-xmpl-c5c3-2c87-81584709a377",  
    "detail-type": "RDS DB Instance Event",  
    "source": "aws.rds",  
    "account": "123456789012",  
    "time": "2020-04-28T07:20:20Z",  
    "region": "us-east-2",  
    "resources": [  
        "arn:aws:rds:us-east-2:123456789012:db:rdz6xmpliljlb1"  
    ],  
    "detail": {  
        "EventCategories": [  
            "backup"  
        ],  
        "SourceType": "DB_INSTANCE",  
        "SourceArn": "arn:aws:rds:us-east-2:123456789012:db:rdz6xmpliljlb1",  
        "Date": "2020-04-28T07:20:20.112Z",  
        "Message": "Finished DB Instance backup",  
        "SourceIdentifier": "rdz6xmpliljlb1"  
    }  
}
```

Também é possível criar uma função do Lambda e direcionar o AWS Lambda para invocá-la em uma programação regular. É possível especificar uma taxa fixa (por exemplo, invocar uma função do Lambda a cada hora ou a cada 15 minutos) ou especificar uma expressão Cron.

Example Evento de mensagem do CloudWatch Events

```
{  
    "version": "0",  
    "account": "123456789012",  
    "region": "us-east-2",  
    "detail": {},  
    "detail-type": "Scheduled Event",  
    "source": "aws.events",  
    "time": "2019-03-01T01:23:45Z",  
    "id": "cdc73f9d-aea9-11e3-9d5a-835b769c0d9c",  
    "resources": [  
        "arn:aws:events:us-east-2:123456789012:rule/my-schedule"  
    ]  
}
```

Para configurar o CloudWatch Events para invocar a função

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Em Function overview (Visão geral da função), escolha Add trigger (Adicionar gatilho).
4. Defina o tipo de acionador como EventBridge (CloudWatch Events).
5. Em Rule (Regra), selecione Create a new rule (Criar uma regra).
6. Configure as opções restantes e selecione Add (Adicionar).

Para obter mais informações sobre programações de expressões, consulte [Programar expressões usando rate ou cron \(p. 320\)](#).

Cada conta da AWS pode ter até 100 fontes de eventos exclusivas do tipo de fonte CloudWatch Events - Schedule. Cada uma dessas pode ser a fonte de eventos para até cinco funções do Lambda. Ou seja, você pode ter até 500 funções do Lambda em execução em um cronograma na sua conta da AWS.

Tópicos

- [Tutorial: usar o AWS Lambda com eventos programados \(p. 316\)](#)
- [AWS SAMModelo do para um aplicativo do CloudWatch Events \(p. 319\)](#)
- [Programar expressões usando rate ou cron \(p. 320\)](#)

Tutorial: usar o AWS Lambda com eventos programados

Neste tutorial, você faz o seguinte:

- Crie uma função do lambda usando o esquema lambda-canary. Você configura a função do Lambda para execução a cada minuto. Observe que, se a função retornar um erro, o AWS Lambda registrará as métricas de erro no CloudWatch.
- Configure um alarme do CloudWatch na métrica `Errors` da sua função do Lambda para postar uma mensagem no seu tópico do Amazon SNS quando o AWS Lambda emitir métricas de erro para o CloudWatch. Você se inscreve nos tópicos do Amazon SNS para receber notificação por e-mail. Neste tutorial, você faz o seguinte para configurar isso:
 - Crie um tópico do Amazon SNS.
 - Registre-se no tópico de forma que possa receber notificações por e-mail quando uma nova mensagem for publicada no tópico.
 - No Amazon CloudWatch, defina um alarme sobre a métrica `Errors` da sua função do Lambda para publicar uma mensagem no tópico do SNS quando ocorrerem erros.

Prerequisites

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em [Conceitos básicos do Lambda \(p. 9\)](#) para criar sua primeira função do Lambda.

Criar uma função do Lambda

1. Faça login no AWS Management Console e abra o console AWS Lambda em <https://console.aws.amazon.com/lambda/>.
2. Escolha Create function.

3. Selecione Use a blueprint (Usar um esquema).
4. Insira **canary** na barra de pesquisa. Escolha o esquema lambda-canary e selecione Configure (Configurar).
5. Configure as definições a seguir.
 - Nome – **lambda-canary**.
 - Role (Função): Create a new role from AWS policy templates.
 - Nome da função – **lambda-apigateway-role**.
 - Policy templates (Modelos de política): Simple microservice permissions.
 - Rule (Regra): Crie uma nova regra.
 - Rule name (Nome da regra – **CheckWebsiteScheduledEvent**).
 - Rule description (Descrição da regra – **CheckWebsiteScheduledEvent trigger**).
 - Schedule expression (Expressão de programação – **rate(1 minute)**).
 - Enabled (Habilitado): Verdadeiro (marcado).
 - Variáveis de ambiente
 - site – **https://docs.aws.amazon.com/lambda/latest/dg/welcome.html**.
 - expected (esperado): **What is AWS Lambda?**
6. Escolha Create function.

O CloudWatch Events emite um evento a cada minuto, com base na expressão de programação. O evento aciona a função do Lambda, que verifica se a string esperada aparece na página especificada. Para obter mais informações sobre programações de expressões, consulte [Programar expressões usando rate ou cron \(p. 320\)](#).

Testar a função do Lambda

Teste a função com um evento de exemplo fornecido pelo console do Lambda.

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha a função lambda-canary.
3. Escolha Test (Testar).
4. Crie um evento usando o modelo de evento do CloudWatch Events.
5. Escolha Create event (Criar evento).
6. Escolha Invoke (Invocar).

A saída da execução da função é mostrada na parte superior da página.

Criar um tópico do Amazon SNS e inscrever-se nele

Crie um tópico do Amazon Simple Notification Service (Amazon SNS) para receber notificações quando a função canary retornar um erro.

Para criar um tópico

1. Abra o [console do Amazon SNS](#).
2. Escolha Create topic.
3. Crie um tópico com as configurações a seguir.
 - Nome – **lambda-canary-notifications**.
 - Display name (Nome de exibição – **Canary**).
4. Selecione Create subscription.

5. Crie uma inscrição com as configurações a seguir.

- Protocol (Protocolo – **Email**).
- Endpoint: seu endereço de e-mail.

O Amazon SNS envia um e-mail de Canary <no-reply@sns.amazonaws.com>, refletindo o nome do tópico que é fácil de lembrar. Use o link no e-mail para confirmar seu endereço.

Configurar um alarme

Configure um alarme no Amazon CloudWatch que monitore a função do Lambda e envie uma notificação em caso de falha.

Para criar um alarme

1. Abra o [console do CloudWatch](#).
2. Escolha Alarms.
3. Selecione Create alarm (Criar alarme).
4. Escolha Alarms.
5. Crie um alarme com as configurações a seguir.

- Metrics (Métricas): lambda-canary Errors.

Procure **lambda canary errors** para encontrar a métrica.

- Statistic (Estatística) – **Sum**.

Escolha a estatística no menu suspenso acima do gráfico de visualização.

- Nome – **lambda-canary-alarm**.
- Descrição – **Lambda canary alarm**.
- Threshold (Limite): Whenever Errors is ≥ 1 (Sempre que o erro for ≥ 1).
- Send notification to (Enviar notificação para – **lambda-canary-notifications**).

Testar o alarme

Atualize a configuração da função para fazer com que a função retorne um erro, o que aciona o alarme.

Como acionar um alarme

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha a função lambda-canary.
3. Role para baixo. Em Environment variables (Variáveis de ambiente), selecione Edit (Editar).
4. Defina expected (esperado) como **404**.
5. Escolha Save (Salvar).

Espere um minuto e verifique se há uma mensagem do Amazon SNS em seu e-mail.

Limpar recursos da

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua conta da AWS.

Como excluir a função do Lambda

1. Abra a página [Functions](#) (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Actions, Delete.
4. Escolha Delete.

Para excluir o alarme do CloudWatch

1. Abra a página [Alarms](#) (Alarmes) do console do CloudWatch.
2. Selecione o alarme que você criou.
3. Escolha Actions, Delete.
4. Escolha Delete.

Para excluir a inscrição do Amazon SNS

1. Abrir a [Página Assinaturas](#) No console do Amazon SNS.
2. Selecione a assinatura que você criou.
3. Escolha Delete (Excluir), Yes, Delete (Sim, excluir).

Para excluir o tópico do Amazon SNS

1. Abra a página [Topics](#) (Tópicos) no console do Amazon SNS.
2. Selecione o tópico que você criou.
3. Escolha Delete.
4. Digite **delete me** na caixa de texto.
5. Escolha Delete.

AWS SAM Modelo do para um aplicativo do CloudWatch Events

Você pode criar esse aplicativo usando [AWS SAM](#). Para saber mais sobre como criar modelos do AWS SAM, consulte [Noções básicas de modelos do AWS SAM](#) no Guia do desenvolvedor do AWS Serverless Application Model.

Veja abaixo um modelo de exemplo do AWS SAM para a aplicação do Lambda do [tutorial \(p. 316\)](#). Copie o texto abaixo para um arquivo .yaml e salve-o ao lado do pacote ZIP criado previamente. Observe que os valores dos parâmetros Handler e Runtime devem corresponder àqueles usados quando você criou a função na seção anterior.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Parameters:
  NotificationEmail:
    Type: String
Resources:
  CheckWebsitePeriodically:
    Type: AWS::Serverless::Function
```

```

Properties:
  Handler: LambdaFunctionOverHttps.handler
  Runtime: runtime
  Policies: AmazonDynamoDBFullAccess
  Events:
    CheckWebsiteScheduledEvent:
      Type: Schedule
      Properties:
        Schedule: rate(1 minute)

AlarmTopic:
  Type: AWS::SNS::Topic
  Properties:
    Subscription:
      - Protocol: email
        Endpoint: !Ref NotificationEmail

Alarm:
  Type: AWS::CloudWatch::Alarm
  Properties:
    AlarmActions:
      - !Ref AlarmTopic
    ComparisonOperator: GreaterThanOrEqualToThreshold
    Dimensions:
      - Name: FunctionName
        Value: !Ref CheckWebsitePeriodically
    EvaluationPeriods: 1
    MetricName: Errors
    Namespace: AWS/Lambda
    Period: 60
    Statistic: Sum
    Threshold: '1'

```

Para obter informações sobre como empacotar e implantar o aplicativo sem servidor usando os comandos de empacotamento e implantação, consulte [Implantar aplicativos sem servidor](#) no Guia do desenvolvedor do AWS Serverless Application Model.

Programar expressões usando rate ou cron

O AWS Lambda oferece suporte às expressões cron e rate padrão para frequências de até uma vez por minuto. As expressões de taxa do CloudWatch Events têm o formato a seguir

```
rate(Value Unit)
```

Em que **Valor** é um inteiro positivo e **Unidade** pode ser minuto(s), hora(s) ou dia(s). Para um valor singular, a unidade deverá ser singular (por exemplo, `rate(1 day)`); caso contrário, plural (por exemplo, `rate(5 days)`).

Exemplos de expressão de taxa

Frequência	Expressão
A cada 5 minutos	<code>rate(5 minutes)</code>
A cada hora	<code>rate(1 hour)</code>
A cada sete dias	<code>rate(7 days)</code>

As expressões cron têm o formato a seguir.

```
cron(Minutes Hours Day-of-month Month Day-of-week Year)
```

Exemplos de expressão cron

Frequência	Expressão
10h15 (UTC) a cada dia	<code>cron(15 10 * * ? *)</code>
18h00 de segunda-feira a sexta-feira	<code>cron(0 18 ? * MON-FRI *)</code>
8h00 no primeiro dia do mês	<code>cron(0 8 1 * ? *)</code>
A cada 10 min em dias da semana	<code>cron(0/10 * ? * MON-FRI *)</code>
A cada 5 minutos entre 8h00 e 17h55 em dias da semana	<code>cron(0/5 8-17 ? * MON-FRI *)</code>
9h00 na primeira segunda-feira de cada mês	<code>cron(0 9 ? * 2#1 *)</code>

Observe o seguinte:

- Se você estiver usando o console do Lambda, não inclua o prefixo `cron` na expressão.
- Um dos valores de dia do mês ou dia da semana deverá ser um ponto de interrogação (?).

Para obter mais informações, consulte [Programar expressões para regras](#) no Manual do usuário do EventBridge.

Usar o Lambda com CloudWatch Logs

Você pode usar uma função do Lambda para monitorar e analisar logs de uma transmissão de log do Amazon CloudWatch Logs. Crie [inscrições](#) para um ou mais fluxos de log para invocar uma função quando os logs forem criados ou corresponderem a um padrão opcional. Use a função para enviar uma notificação ou manter o log em um banco de dados ou armazenamento.

O CloudWatch Logs invoca a função de forma assíncrona com um evento que contém dados de log. O valor do campo de dados é um arquivo .gzip codificado em base64.

Example Evento de mensagem do CloudWatch Logs

```
{  
  "awslogs": {  
    "data":  
      "ewogICAgIm1lc3NhZ2VUeXB1IjogIkRBVEFFTUVTU0FHRSIsCiAgICAib3duZXIIioAiMTIzNDU2Nzg5MDEyIiwKICAgICJs2dHc  
  }  
}
```

Quando decodificados e descompactados, os dados de log compõem um documento JSON com a seguinte estrutura:

Example Dados de mensagem do CloudWatch Logs (decodificados)

```
{  
  "messageType": "DATA_MESSAGE",  
  "owner": "123456789012",  
  "logGroup": "/aws/lambda/echo-nodejs",  
  "logStream": "2019/03/13/[$LATEST]94fa867e5374431291a7fc14e2f56ae7",  
  "subscriptionFilters": [  
    "LambdaStream_cloudwatchlogs-node"  
  ],  
  "logEvents": [  
    {  
      "id": "34622316099697884706540976068822859012661220141643892546",  
      "timestamp": 1552518348220,  
      "message": "REPORT RequestId: 6234bffe-149a-b642-81ff-2e8e376d8aff\\tDuration:  
46.84 ms\\tBilled Duration: 47 ms \\tMemory Size: 192 MB\\tMax Memory Used: 72 MB\\t\\n"  
    }  
  ]  
}
```

Para uma aplicação de exemplo que usa o CloudWatch Logs como acionador, consulte [Aplicativo de exemplo de processador de erros para o AWS Lambda \(p. 500\)](#).

Usar o AWS Lambda com o AWS CloudFormation

Em um modelo do AWS CloudFormation, você pode especificar uma função do Lambda como o destino de um recurso personalizado. Use recursos personalizados para processar parâmetros, recuperar valores de configurações ou chamar outros serviços da AWS durante eventos de ciclo de vida da pilha.

O exemplo a seguir invoca uma função que foi definida em outro lugar do modelo.

Example – Definição de recurso personalizado

```
Resources:  
  primerinvoke:  
    Type: AWS::CloudFormation::CustomResource  
    Version: "1.0"  
    Properties:  
      ServiceToken: !GetAtt primer.Arn  
      FunctionName: !Ref randomerror
```

O token do serviço é o nome de recurso da Amazon (ARN) da função que o AWS CloudFormation invoca quando você cria, atualiza ou exclui a pilha. Você também pode incluir propriedades, como `FunctionName`, que o AWS CloudFormation transmite à sua função como está.

O AWS CloudFormation invoca a função do Lambda [de forma assíncrona \(p. 161\)](#) com um evento que inclui um URL de retorno de chamada.

Example – Evento de mensagem do AWS CloudFormation

```
{  
  "RequestType": "Create",  
  "ServiceToken": "arn:aws:lambda:us-east-2:123456789012:function:lambda-error-processor-primer-14ROR2T3JKU66",  
  "ResponseURL": "https://cloudformation-custom-resource-response-useast2.s3-us-east-2.amazonaws.com/arn%3Aaws%3Acloudformation%3Aus-east-2%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-2:123456789012: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-2:123456789012:function:lambda-error-processor-primer-14ROR2T3JKU66",  
    "FunctionName": "lambda-error-processor-randomerror-ZWUC391MQAJK"  
  }  
}
```

A função é responsável por retornar uma resposta ao URL de retorno que indica êxito ou falha. Para sintaxe de resposta completa, consulte [Objetos de resposta de recursos personalizados](#).

Example – Resposta de recursos personalizados do AWS CloudFormation

```
{  
  "Status": "SUCCESS",  
  "PhysicalResourceId": "2019/04/18/[ $LATEST ]b3d1bfc65f19ec610654e4d9b9de47a0",  
  "StackId": "arn:aws:cloudformation:us-east-2:123456789012:stack/lambda-error-processor/1134083a-2608-1e91-9897-022501a2c456",
```

```
        "RequestId": "5d478078-13e9-baf0-464a-7ef285ecc786",
        "LogicalResourceId": "primerinvoke"
    }
```

O AWS CloudFormation fornece uma biblioteca chamada `cfn-response` que lida com o envio da resposta. Se definir a função dentro de um modelo, você poderá exigir a biblioteca pelo nome. O AWS CloudFormation adiciona a biblioteca ao pacote de implantação que cria para a função.

O exemplo a seguir invoca uma segunda função. Se a chamada tiver êxito, a função enviará uma resposta de êxito ao AWS CloudFormation e a atualização da pilha continuará. O modelo usa o tipo de recurso [AWS::Serverless::Function](#) fornecido pelo AWS Serverless Application Model.

Example [error-processor/template.yml](#) – Função do recurso personalizado

```
Transform: 'AWS::Serverless-2016-10-31'
Resources:
  primer:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs12.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 a função que o recurso personalizado invocar não estiver definida em um modelo, você poderá obter o código fonte para o `cfn-response` no [módulo cfn-response](#) no Manual do usuário do AWS CloudFormation .

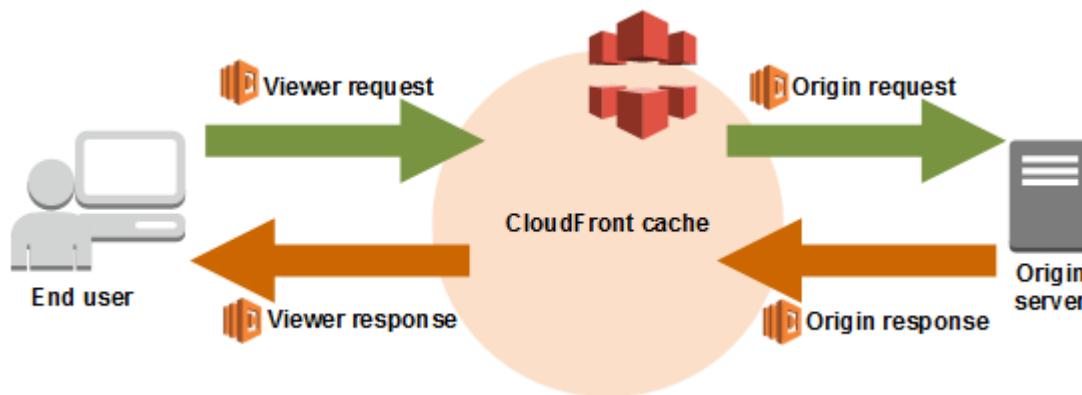
Para obter um aplicativo de exemplo que usa um recurso personalizado para garantir que o grupo de logs de uma função seja criado antes de outro recurso que dependa dele, consulte [Aplicativo de exemplo de processador de erros para o AWS Lambda \(p. 500\)](#).

Para obter mais informações sobre recursos personalizados, consulte [Recursos personalizados no AWS CloudFormation Guia do usuário](#) do.

Como usar o AWS Lambda com o Lambda@Edge do CloudFront

O Lambda@Edge permite executar funções do Lambda em Node.js e em Python para personalizar o conteúdo oferecido pelo CloudFront, executando as funções nos locais da AWS mais próximos do visualizador. As funções executadas em resposta a eventos do CloudFront, sem provisionamento nem gerenciamento de servidores. Você pode usar funções do Lambda para alterar as solicitações e as respostas do CloudFront nos seguintes pontos:

- Depois que o CloudFront receber uma solicitação de um visualizador (solicitação do visualizador)
- Antes do CloudFront encaminhar a solicitação para a origem (solicitação da origem)
- Depois que o CloudFront receber a resposta da origem (resposta da origem)
- Antes que o CloudFront encaminhe a resposta ao visualizador (resposta do visualizador)



Note

O Lambda@Edge oferece suporte a um conjunto limitado de tempos de execução e recursos. Para obter detalhes, consulte [Requisitos e restrições sobre funções do Lambda](#) no guia do desenvolvedor do Amazon CloudFront.

Você também pode gerar respostas aos visualizadores sem nunca enviar a solicitação para a origem.

Example Evento do CloudFront

```
{  
  "Records": [  
    {  
      "cf": {  
        "config": {  
          "distributionId": "EDFDVBD6EXAMPLE"  
        },  
        "request": {  
          "clientIp": "2001:0db8:85a3:0:0:8a2e:0370:7334",  
          "method": "GET",  
          "uri": "/picture.jpg",  
          "headers": {  
            "host": [  
              {  
                "key": "Host",  
                "value": "d111111abcdef8.cloudfront.net"  
              }  
            ]  
          }  
        }  
      }  
    }  
  ]  
}
```

```
        "user-agent": [
            {
                "key": "User-Agent",
                "value": "curl/7.51.0"
            }
        ]
    }
}
```

Com o Lambda@Edge é possível criar diversas soluções, por exemplo:

- Inspecione os cookies para reescrever URLs para diferentes versões de um site para teste A/B.
- Envie objetos diferentes para seus usuários com base no cabeçalho `User-Agent`, que contém informações sobre o dispositivo que enviou a solicitação. Por exemplo, você pode enviar imagens em diferentes resoluções aos usuários, dependendo de seus dispositivos.
- Inspecione cabeçalhos ou tokens autorizados, inserindo um cabeçalho correspondente e permitindo controle de acesso antes de encaminhar uma solicitação para a origem.
- Adicione, exclua e modifique cabeçalhos e reescreva o caminho do URL para direcionar os usuários para diferentes objetos no cache.
- Gere novas respostas HTTP para executar coisas como redirecionar usuários não autenticados para páginas de login ou criar páginas da Web estáticas e entregá-las direto do ponto de presença. Para obter mais informações, consulte [Usar funções do Lambda para gerar respostas HTTP a solicitações de visualizador e origem](#) no Guia do desenvolvedor do Amazon CloudFront.

Para obter mais informações sobre como usar o Lambda@Edge, consulte [Uso do CloudFront com o Lambda@Edge](#).

Usar o AWS Lambda com o AWS CodeCommit

Você pode criar um acionador para um repositório do AWS CodeCommit de modo que eventos no repositório chamem uma função do Lambda. Por exemplo, você pode invocar uma função do Lambda quando uma ramificação ou etiqueta é criada ou quando um push é feito para uma ramificação existente.

Example AWS CodeCommitEvento de mensagem do

```
{  
    "Records": [  
        {  
            "awsRegion": "us-east-2",  
            "codecommit": {  
                "references": [  
                    {  
                        "commit": "5e493c6f3067653f3d04eca608b4901eb227078",  
                        "ref": "refs/heads/master"  
                    }  
                ]  
            },  
            "eventId": "31ade2c7-f889-47c5-a937-1cf99e2790e9",  
            "eventName": "ReferenceChanges",  
            "eventPartNumber": 1,  
            "eventSource": "aws:codecommit",  
            "eventSourceARN": "arn:aws:codecommit:us-east-2:123456789012:lambda-pipeline-  
repo",  
            "eventTime": "2019-03-12T20:58:25.400+0000",  
            "eventTotalParts": 1,  
            "eventTriggerConfigId": "0d17d6a4-efeb-46f3-b3ab-a63741badeb8",  
            "eventTriggerName": "index.handler",  
            "eventVersion": "1.0",  
            "userIdentityARN": "arn:aws:iam::123456789012:user/intern"  
        }  
    ]  
}
```

Para obter mais informações, consulte [Gerenciar triggers para um repositório do AWS CodeCommit](#).

Usar o AWS Lambda com o AWS CodePipeline

AWS CodePipeline O é um serviço que permite criar pipelines de entrega contínua para aplicações que são executadas na AWS. Você pode criar um pipeline para implantar sua aplicação do Lambda. Também pode configurar um pipeline para invocar uma função do Lambda para executar uma tarefa quando o pipeline é executado. Quando você [cria uma aplicação do Lambda \(p. 192\)](#) no console do Lambda, o Lambda cria um pipeline que inclui estágios de origem, compilação e implantação.

O CodePipeline invoca a função de forma assíncrona com um evento que contém detalhes sobre o trabalho. O exemplo a seguir mostra um evento de um pipeline que invocou uma função chamada `my-function`.

Example CodePipeline

```
{  
    "CodePipeline.job": {  
        "id": "c0d76431-b0e7-xmpl-97e3-e8ee786eb6f6",  
        "accountId": "123456789012",  
        "data": {  
            "actionConfiguration": {  
                "configuration": {  
                    "FunctionName": "my-function",  
                    "UserParameters": "{\"KEY\": \"VALUE\"}"  
                }  
            },  
            "inputArtifacts": [  
                {  
                    "name": "my-pipeline-SourceArtifact",  
                    "revision": "e0c7xmpl2308ca3071aa7bab414de234ab52eea",  
                    "location": {  
                        "type": "S3",  
                        "s3Location": {  
                            "bucketName": "us-west-2-123456789012-my-pipeline",  
                            "objectKey": "my-pipeline/test-api-2/TdOSFRV"  
                        }  
                    }  
                }  
            ],  
            "outputArtifacts": [  
                {  
                    "name": "invokeOutput",  
                    "revision": null,  
                    "location": {  
                        "type": "S3",  
                        "s3Location": {  
                            "bucketName": "us-west-2-123456789012-my-pipeline",  
                            "objectKey": "my-pipeline/invokeOutp/D0YHsJn"  
                        }  
                    }  
                }  
            ],  
            "artifactCredentials": {  
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE",  
                "secretAccessKey": "6CGtmAa3lzWtV7a...",  
                "sessionToken": "IQoJb3JpZ2luX2VjEA...",  
                "expirationTime": 1575493418000  
            }  
        }  
    }  
}
```

Para concluir o trabalho, a função deve chamar a API do CodePipeline para sinalizar êxito ou falha. O exemplo de função Node.js a seguir usa a operação `PutJobSuccessResult` para sinalizar êxito. Ela obtém o ID de trabalho para a chamada de API do objeto de evento.

Example index.js

```
var AWS = require('aws-sdk')
var codepipeline = new AWS.CodePipeline()

exports.handler = async (event) => {
    console.log(JSON.stringify(event, null, 2))
    var jobId = event["CodePipeline.job"].id
    var params = {
        jobId: jobId
    }
    return codepipeline.putJobSuccessResult(params).promise()
}
```

Para a invocação assíncrona, o Lambda coloca em fila a mensagem e [tenta novamente \(p. 176\)](#) caso a sua função retorne um erro. Configure sua função com um [destino \(p. 164\)](#) para manter os eventos que a função não puder processar.

Para obter um tutorial sobre como configurar um pipeline para invocar uma função do Lambda, consulte [Invocar uma função do AWS Lambda em um pipeline](#) no Manual do usuário do AWS CodePipeline.

Você pode usar o AWS CodePipeline para criar um pipeline de entrega contínua para o seu aplicativo do Lambda. O CodePipeline combina o controle, a compilação e a implantação de recursos da origem para criar um pipeline que é executado sempre que você faz uma alteração no código-fonte da aplicação.

Para obter um método alternativo de criação de um pipeline com o AWS Serverless Application Model e o AWS CloudFormation, assista a [Automate your serverless application deployments](#) no canal do YouTube da Amazon Web Services.

Permissions

Para invocar uma função, um pipeline do CodePipeline precisa de permissão para usar as seguintes operações de API:

- [listFunctions \(p. 894\)](#)
- [InvokeFunction \(p. 875\)](#)

A função de serviço de pipeline padrão inclui essas permissões.

Para concluir um trabalho, a função precisa das seguintes permissões em sua [função de execução \(p. 57\)](#).

- `codepipeline:PutJobSuccessResult`
- `codepipeline:PutJobFailureResult`

Essas permissões estão incluídas na política gerenciada [AWSCodePipelineCustomActionAccess](#).

Usar o AWS Lambda com o Amazon Cognito

O recurso de eventos do Amazon Cognito permite executar funções do Lambda em resposta a eventos no Amazon Cognito. Por exemplo, você pode invocar uma função do Lambda para os eventos Sync Trigger, que são publicados toda vez que um conjunto de dados é sincronizado. Para saber mais e obter um exemplo prático, consulte [Introducing Amazon Cognito Events: Sync Triggers](#) no blog Mobile Development (Desenvolvimento para dispositivos móveis).

Example Evento de mensagem do Amazon Cognito

```
{  
    "datasetName": "datasetName",  
    "eventType": "SyncTrigger",  
    "region": "us-east-1",  
    "identityId": "identityId",  
    "datasetRecords": {  
        "SampleKey2": {  
            "newValue": "newValue2",  
            "oldValue": "oldValue2",  
            "op": "replace"  
        },  
        "SampleKey1": {  
            "newValue": "newValue1",  
            "oldValue": "oldValue1",  
            "op": "replace"  
        }  
    },  
    "identityPoolId": "identityPoolId",  
    "version": 2  
}
```

Você configura o mapeamento de fonte do evento usando a configuração de assinatura de eventos do Amazon Cognito. Para obter informações sobre o mapeamento de origem do evento e um evento de exemplo, consulte [Amazon Cognito events](#) no Amazon Cognito Developer Guide.

Usar o AWS Lambda com o AWS Config

Você pode usar as funções de AWS Lambda para avaliar se as configurações de recursos da AWS são compatíveis com suas regras personalizadas do Config. À medida que os recursos são criados, excluídos ou alterados, o AWS Config registra essas alterações e envia as informações para suas funções do Lambda. Em seguida, suas funções do Lambda avaliam as alterações e relatam os resultados para o AWS Config. Em seguida, você pode usar o AWS Config para avaliar a conformidade geral dos recursos: você pode saber quais recursos não estão em conformidade, e quais atributos da configuração são a causa da não conformidade.

Example AWS ConfigEvento de mensagem do

```
{  
    "invokingEvent": "{\"configurationItem\":{\"configurationItemCaptureTime\": \"2016-02-17T01:36:34.043Z\", \"awsAccountId\": \"000000000000\", \"configurationItemStatus\": \"OK\", \"resourceId\": \"i-0000000000\", \"ARN\": \"arn:aws:ec2:us-east-1:000000000000:instance/i-00000000\", \"awsRegion\": \"us-east-1\", \"availabilityZone\": \"us-east-1a\", \"resourceType\": \"AWS::EC2::Instance\", \"tags\": {\"Foo\": \"Bar\"}, \"relationships\": [{\"resourceId\": \"eipalloc-00000000\", \"resourceType\": \"AWS::EC2::EIP\", \"name\": \"Is attached to ElasticIp\"}], \"configuration\": {\"foo\": \"bar\"}, \"messageType\": \"ConfigurationItemChangeNotification\"},  
    \"ruleParameters\": {\"myParameterKey\": \"myParameterValue\"},  
    \"resultToken\": \"myResultToken\",  
    \"eventLeftScope\": false,  
    \"executionRoleArn\": \"arn:aws:iam::012345678912:role/config-role\",  
    \"configRuleArn\": \"arn:aws:config:us-east-1:012345678912:config-rule/config-rule-0123456\",  
    \"configRuleName\": \"change-triggered-config-rule\",  
    \"configRuleId\": \"config-rule-0123456\",  
    \"accountId\": \"012345678912\",  
    \"version\": \"1.0\"\n}
```

Para obter mais informações, consulte [Avaliar recursos com regras do AWS Config](#).

Usar o Lambda com o Amazon Connect

Você pode usar uma função do Lambda para processar solicitações do Amazon Connect.

O Amazon Connect invoca a sua função do Lambda de forma síncrona com um evento que contém o corpo da solicitação e os metadados.

Example Evento de solicitação do Amazon Connect

```
{  
    "Details": {  
        "ContactData": {  
            "Attributes": {},  
            "Channel": "VOICE",  
            "ContactId": "4a573372-1f28-4e26-b97b-XXXXXXXXXXXX",  
            "CustomerEndpoint": {  
                "Address": "+1234567890",  
                "Type": "TELEPHONE_NUMBER"  
            },  
            "InitialContactId": "4a573372-1f28-4e26-b97b-XXXXXXXXXXXX",  
            "InitiationMethod": "INBOUND | OUTBOUND | TRANSFER | CALLBACK",  
            "InstanceARN": "arn:aws:connect:aws-region:1234567890:instance/  
c8c0e68d-2200-4265-82c0-XXXXXXXXXX",  
            "PreviousContactId": "4a573372-1f28-4e26-b97b-XXXXXXXXXXXX",  
            "Queue": {  
                "ARN": "arn:aws:connect:eu-west-2:111111111111:instance/cccccccc-bbbb-dddd-  
eeee-ffffffffffff/queue/aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee",  
                "Name": "PasswordReset"  
            },  
            "SystemEndpoint": {  
                "Address": "+1234567890",  
                "Type": "TELEPHONE_NUMBER"  
            }  
        },  
        "Parameters": {  
            "sentAttributeKey": "sentAttributeValue"  
        }  
    },  
    "Name": "ContactFlowEvent"  
}
```

Para obter informações sobre como usar o Amazon Connect com o Lambda, consulte [Invocar funções do Lambda](#) no Guia do administrador do Amazon Connect.

Usar o AWS Lambda com o Amazon DynamoDB

É possível usar uma função do AWS Lambda para processar registros em um [Amazon DynamoDB Stream](#). Com o DynamoDB Streams, você pode acionar uma função do Lambda para executar o trabalho adicional cada vez que uma tabela do DynamoDB é atualizada.

O Lambda lê registros da transmissão e invoca sua função [de maneira síncrona \(p. 158\)](#) com um evento que contém registros de transmissão. O Lambda lê registros em lotes e invoca sua função para processar registros do lote.

Example Evento de registro do DynamoDB Streams

```
{  
    "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": "eventsourcearn",  
            "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"  
                    }  
                }  
            }  
        }  
    ]  
}
```

```
        "Id": {  
            "N": "101"  
        }  
    },  
    "StreamViewType": "NEW_AND_OLD_IMAGES"  
},  
"awsRegion": "us-west-2",  
"eventName": "MODIFY",  
"eventSourceARN": sourcearn,  
"eventSource": "aws:dynamodb"  
}
```

O Lambda sonda os fragmentos em sua transmissão do DynamoDB em busca de registros a uma taxa básica de 4 vezes por segundo. Quando os registros estão disponíveis, o Lambda invoca a função e aguarda o resultado. Se o processamento for bem-sucedido, o Lambda continua a sondagem até que ela receba mais registros.

Por padrão, o Lambda invoca sua função assim que os registros estão disponíveis na transmissão. Se o lote que o Lambda lê da transmissão tiver apenas um registro nele, o Lambda envia apenas um registro para a função. Para evitar invocar a função com um número pequeno de registros, você pode instruir à origem dos eventos para fazer o buffer dos registros por até cinco minutos, configurando uma janela de lote. Antes de invocar a função, o Lambda continua a ler registros da transmissão até coletar um lote inteiro, ou até que a janela de lote expire.

Se a sua função retornar um erro, o Lambda tentará executar novamente o lote até que o processamento seja bem-sucedido ou os dados expirem. Para evitar fragmentos paralisados, você pode configurar o mapeamento de fontes de eventos para tentar novamente com um tamanho de lote menor, limitar o número de tentativas ou descartar registros que são muito antigos. Para manter eventos descartados, você pode configurar o mapeamento de fontes de eventos para enviar detalhes sobre lotes com falha para uma fila do SQS ou para um tópico do SNS.

Você também pode aumentar a simultaneidade processando vários lotes de cada fragmento em paralelo. O Lambda pode processar até 10 lotes em cada fragmento simultaneamente. Se você aumentar o número de lotes simultâneos por fragmento, o Lambda ainda garante o processamento em ordem no nível de chave de partição.

Configure `ParallelizationFactor` para processar um fragmento de um fluxo de dados do Kinesis ou do DynamoDB com mais de uma invocação do Lambda simultaneamente. Você pode especificar o número de lotes simultâneos que o Lambda pesquisa de um fragmento por meio de um fator de paralelização de 1 (padrão) a 10. Por exemplo, quando `ParallelizationFactor` estiver definido como 2, você poderá ter 200 invocações simultâneas do Lambda no máximo para processar 100 fragmentos de dados do Kinesis. Isso ajuda a aumentar a taxa de transferência de processamento quando o volume de dados é volátil e o valor de `IteratorAge` é alto. Observe que o fator de paralelização não funcionará se você estiver usando a agregação do Kinesis. Para obter mais informações, consulte [New AWS Lambda scaling controls for Kinesis and DynamoDB event sources](#).

Seções

- [Permissões da função de execução \(p. 335\)](#)
- [Configurar um stream como fonte de eventos \(p. 335\)](#)
- [APIs de mapeamento da fonte de eventos \(p. 336\)](#)
- [Tratamento de erros \(p. 338\)](#)
- [Métricas do Amazon CloudWatch \(p. 339\)](#)
- [Janelas de tempo \(p. 339\)](#)
- [Gerar relatórios de falhas de itens de lote \(p. 343\)](#)
- [Tutorial: Usar o AWS Lambda com o Amazon DynamoDB Streams \(p. 345\)](#)
- [Código de exemplo da função do \(p. 350\)](#)
- [Modelo do AWS SAM para uma aplicação do DynamoDB \(p. 353\)](#)

Permissões da função de execução

O Lambda precisa das permissões a seguir para gerenciar recursos relacionados à sua transmissão do DynamoDB. Adicione-as à função de execução da sua função.

- [dynamodb:DescribeStream](#)
- [dynamodb:GetRecords](#)
- [dynamodb:GetShardIterator](#)
- [dynamodb>ListStreams](#)
- [dynamodb>ListShards](#)

A política gerenciada [AWSLambdaDynamoDBExecutionRole](#) inclui essas permissões. Para obter mais informações, consulte [AWS Lambda Função de execução do \(p. 57\)](#).

Para enviar registros de lotes com falha para uma fila ou um tópico, sua função precisa de permissões adicionais. Cada serviço de destino requer uma permissão diferente, como se segue:

- Amazon SQS – [sq:SendMessage](#)
- Amazon SNS – [sns:Publish](#)

Configurar um stream como fonte de eventos

Crie um mapeamento de fontes de eventos para orientar o Lambda a enviar registros de sua transmissão para uma função do Lambda. É possível criar vários mapeamentos de origem de evento para processar os mesmos dados com várias funções do Lambda ou processar itens de vários fluxos com uma única função.

Para configurar sua função para leitura no DynamoDB Streams no console do Lambda, crie um acionador do DynamoDB.

Para criar um trigger

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Em Function overview (Visão geral da função), escolha Add trigger (Adicionar gatilho).
4. Escolha um tipo de acionador.
5. Configure as opções necessárias e escolha Add (Adicionar).

O Lambda oferece suporte às opções de fontes de eventos do DynamoDB a seguir.

Opções de fonte do evento

- DynamoDB table (Tabela do DynamoDB): a tabela do DynamoDB da qual os registros serão lidos.
- Batch size (Tamanho do lote): o número de registros a serem enviados para a função em cada lote, até 10.000. O Lambda transmite todos os registros no batch para a função em uma única chamada, enquanto o tamanho total dos eventos não exceder o [limite de carga útil \(p. 53\)](#) para invocação síncrona (6 MB).
- Batch window (Janela de lote): especifique o máximo de tempo para reunir registros antes de invocar a função, em segundos.
- Starting position (Posição inicial): processe apenas registros novos ou todos os registros existentes.
 - Latest (Mais recente): processe novos registros adicionados ao fluxo.
 - Trim horizon (Redução horizontal): processe todos os registros na transmissão.

Depois de processar todos os registros existentes, a função é capturada e continua a processar novos registros.

- Destino na falha: uma fila do SQS ou um tópico do SNS para registros que não podem ser processados. Quando o Lambda descarta um lote de registros porque ele é muito antigo ou esgotou todas as tentativas, ele envia detalhes sobre o lote para a fila ou tópico.
- Retry attempts (Tentativas de repetição): o número máximo de vezes que o Lambda tenta novamente quando a função retorna um erro. Isso não se aplica a erros de serviço ou controles em que o lote não atingiu a função.
- Idade máxima do registro: a idade máxima de um registro que o Lambda envia para sua função.
- Dividir lote por erro: quando a função retornar um erro, o lote é dividido em dois antes de uma nova tentativa.
- Lotes simultâneos por fragmento: processa vários lotes do mesmo fragmento simultaneamente.
- Enabled (Habilitado): defina como verdadeiro para habilitar o mapeamento de fontes de eventos. Defina como falso para interromper o processamento de registros. O Lambda monitora o último registro processado e retoma o processamento a partir desse ponto quando o mapeamento é habilitado novamente.

Note

Você não é cobrado por chamadas da API GetRecords invocadas pelo Lambda como parte de açãoadores do DynamoDB.

Para gerenciar a configuração da fonte do evento posteriormente, escolha o gatilho no designer.

APIs de mapeamento da fonte de eventos

Para gerenciar uma origem de evento com a [AWS CLI](#) ou o [AWS SDK](#), você pode usar as seguintes operações da API:

- [CreateEventSourceMapping \(p. 786\)](#)
- [ListEventSourceMappings \(p. 888\)](#)
- [GetEventSourceMapping \(p. 837\)](#)
- [UpdateEventSourceMapping \(p. 956\)](#)
- [DeleteEventSourceMapping \(p. 812\)](#)

O exemplo a seguir usa a AWS CLI para mapear uma função chamada `my-function` para uma transmissão do DynamoDB especificada pelo nome do recurso da Amazon (ARN), com um tamanho de lote de 500.

```
aws lambda create-event-source-mapping --function-name my-function --batch-size 500 --  
starting-position LATEST \  
--event-source-arn arn:aws:dynamodb:us-east-2:123456789012:table/my-table/  
stream/2019-06-10T19:26:16.525
```

Você deve ver a saída a seguir:

```
{  
    "UUID": "14e0db71-5d35-4eb5-b481-8945cf9d10c2",  
    "BatchSize": 500,  
    "MaximumBatchingWindowInSeconds": 0,  
    "ParallelizationFactor": 1,  
    "EventSourceArn": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/  
stream/2019-06-10T19:26:16.525",
```

```
"FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
"LastModified": 1560209851.963,
"LastProcessingResult": "No records processed",
"State": "Creating",
"StateTransitionReason": "User action",
"DestinationConfig": {},
"MaximumRecordAgeInSeconds": 604800,
"BisectBatchOnFunctionError": false,
"MaximumRetryAttempts": 10000
}
```

Configure opções adicionais para personalizar como os lotes são processados e especificar quando descartar registros que não podem ser processados. O exemplo a seguir atualiza um mapeamento de fontes de eventos para enviar um registro de falha para uma fila do SQS após duas tentativas de repetição, ou se os registros tiverem mais de uma hora.

```
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:sqs:us-
east-2:123456789012:dlq"}}'
```

Você deve ver este resultado:

```
{
  "UUID": "f89f8514-cdd9-4602-9e1f-01a5b77d449b",
  "BatchSize": 100,
  "MaximumBatchingWindowInSeconds": 0,
  "ParallelizationFactor": 1,
  "EventSourceArn": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2019-06-10T19:26:16.525",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
  "LastModified": 1573243620.0,
  "LastProcessingResult": "PROBLEM: Function call failed",
  "State": "Updating",
  "StateTransitionReason": "User action",
  "DestinationConfig": {},
  "MaximumRecordAgeInSeconds": 604800,
  "BisectBatchOnFunctionError": false,
  "MaximumRetryAttempts": 10000
}
```

As configurações atualizadas são aplicadas de forma assíncrona e não são refletidas na saída até que o processo seja concluído. Use o comando `get-event-source-mapping` para visualizar o status atual.

```
aws lambda get-event-source-mapping --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b
```

Você deve ver este resultado:

```
{
  "UUID": "f89f8514-cdd9-4602-9e1f-01a5b77d449b",
  "BatchSize": 100,
  "MaximumBatchingWindowInSeconds": 0,
  "ParallelizationFactor": 1,
  "EventSourceArn": "arn:aws:dynamodb:us-east-2:123456789012:table/my-table/
stream/2019-06-10T19:26:16.525",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
  "LastModified": 1573244760.0,
  "LastProcessingResult": "PROBLEM: Function call failed",
  "State": "Enabled",
  "StateTransitionReason": "User action",
```

```
"DestinationConfig": {
    "OnFailure": {
        "Destination": "arn:aws:sqs:us-east-2:123456789012:dlq"
    }
},
"MaximumRecordAgeInSeconds": 3600,
"BisectBatchOnFunctionError": false,
"MaximumRetryAttempts": 2
}
```

Para processar vários lotes simultaneamente, use a opção `--parallelization-factor`.

```
aws lambda update-event-source-mapping --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b11284 \
--parallelization-factor 5
```

Tratamento de erros

O mapeamento de fontes de eventos que lê registros da transmissão do DynamoDB chama sua função de forma síncrona e faz novas tentativas ao encontrar erros. Se a função for limitada ou o serviço Lambda retornar um erro sem invocar a função, o Lambda tentará novamente até que os registros expirem ou excedam a idade máxima configurada no mapeamento de fontes de eventos.

Se a função receber os registros, mas retornar um erro, o Lambda tentará novamente até que os registros no lote expirem, excedam a idade máxima ou atinjam a cota de repetição configurada. Para erros de função, você também pode configurar o mapeamento de fontes de eventos para dividir um lote com falha em dois lotes. Tentar novamente com lotes menores isola registros inválidos e resolve problemas de tempo limite. A divisão de um lote não é considerada para a cota de novas tentativas.

Se as medidas de tratamento de erros falharem, o Lambda descartará os registros e continuará processando lotes provenientes da transmissão. Com as configurações padrão, isso significa que um registro inválido pode bloquear o processamento no fragmento afetado por até um dia. Para evitar isso, configure o mapeamento de fontes de eventos da sua função com um número razoável de tentativas e uma idade máxima de registro que se adapte ao seu caso de uso.

Para reter um registro dos eventos descartados, configure um destino para eventos com falha. O Lambda envia um documento para a fila de destino ou o tópico com detalhes sobre o lote.

Como configurar um destino para registros de eventos com falha

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
 2. Escolha uma função.
 3. Em Function overview (Visão geral da função), escolha Add destination (Adicionar destino).
 4. Em Source (Origem), escolha Stream invocation (Chamada de fluxo).
 5. Para Stream (Fluxo), escolha um fluxo mapeado para a função.
 6. Em Destination type (Tipo de destino), escolha o tipo de recurso que recebe o registro da invocação.
 7. Em Destination (Destino), escolha um recurso.
 8. Escolha Save (Salvar).

O exemplo a seguir mostra um registro de invocação de uma transmissão do DynamoDB.

Example Registro de invocaco

```
{  
    "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"
    }
}
```

Você pode usar essas informações para recuperar os registros afetados da transmissão para solução de problemas. Os registros reais não estão incluídos, portanto, você deve processar esses registros e recuperá-los da transmissão antes que eles expirem e sejam perdidos.

Métricas do Amazon CloudWatch

O Lambda emite a métrica `IteratorAge` quando a sua função termina de processar um lote de registros. A métrica indica a idade do último registro no lote quando o processamento foi concluído. Se a sua função estiver processando novos eventos, você poderá usar a idade do iterador para estimar a latência entre quando um registro é adicionado e quando a função o processa.

Uma tendência crescente na idade do iterador pode indicar problemas com sua função. Para obter mais informações, consulte [Trabalhar com métricas de função do AWS Lambda \(p. 717\)](#).

Janelas de tempo

As funções do Lambda podem executar aplicações de processamento contínuo de transmissões. Um stream representa dados não vinculados que fluem continuamente por meio de sua aplicação. Para analisar as informações dessa entrada de atualização contínua, você pode vincular os registros incluídos usando uma janela definida em termos de tempo.

As janelas de tumbling são janelas de tempo distintas que abrem e fecham em intervalos regulares. Por padrão, as invocações do Lambda são sem estado. Não é possível usá-las para processar dados ao longo de várias invocações contínuas sem um banco de dados externo. No entanto, com as janelas de tumbling, você pode manter seu estado em todas as invocações. Esse estado contém o resultado agregado das mensagens previamente processadas para a janela atual. Seu estado pode ter no máximo 1 MB por fragmento. Se exceder esse tamanho, o Lambda encerra a janela antes.

Cada registro de um stream pertence a uma janela específica. Um registro é processado apenas uma vez, quando o Lambda processa a janela à qual o registro pertence. Em cada janela, você pode executar cálculos, como uma soma ou média, no nível da [chave de partição](#) dentro de um fragmento.

Agregação e processamento

Sua função gerenciada pelo usuário é chamada tanto para agregação quanto para processamento dos resultados finais dessa agregação. O Lambda agrupa todos os registros recebidos na janela. Você pode

receber esses registros em vários lotes, cada um como uma invocação separada. Cada invocação recebe um estado. Assim, ao usar janelas de tumbling, sua resposta de função do Lambda deve conter uma propriedade de state. Se a resposta não contiver uma propriedade de state, o Lambda considerará esta uma invocação com falha. Para satisfazer essa condição, a função pode retornar um objeto do TimeWindowEventResponse, que tem a seguinte forma JSON:

Example TimeWindowEventResponse Valores de

```
{  
    "state": {  
        "1": 282,  
        "2": 715  
    },  
    "batchItemFailures": []  
}
```

Note

Para funções Java, recomendamos o uso de um `Map<String, String>` para representar o estado.

No final da janela, a sinalização `isFinalInvokeForWindow` é definida como `true` para indicar que esse é o estado final e que está pronto para processamento. Após o processamento, a janela é concluída e sua invocação final é concluída e, em seguida, o estado é descartado.

No final da janela, o Lambda usa o processamento final para ações sobre os resultados da agregação. Seu processamento final é invocado de forma síncrona. Após a invocação bem-sucedida, sua função define os pontos de verificação no número da sequência e o processamento de streams continua. Se a invocação não for bem-sucedida, sua função do Lambda suspenderá o processamento adicional até uma chamada bem-sucedida.

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"  
            },  
            "approximateCreationDateTime": "1970-01-01T00:00:00.000Z"  
        }  
    ]  
}
```

```

        "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
}
```

Configuration

Você pode configurar janelas em cascata ao criar ou atualizar um [mapeamento de fonte de eventos \(p. 170\)](#). Para configurar uma janela em cascata, especifique a janela em segundos. O comando de exemplo da AWS Command Line Interface (AWS CLI) a seguir cria um mapeamento de fonte de eventos em streaming com uma janela em cascata de 120 segundos. A função do Lambda definida para agregação e processamento é chamada de `tumbling-window-example-function`.

```
aws lambda create-event-source-mapping --event-source-arn arn:aws:dynamodb:us-east-1:123456789012:stream/lambda-stream --function-name "arn:aws:lambda:us-east-1:123456789018:function:tumbling-window-example-function" --region us-east-1 --starting-position TRIM_HORIZON --tumbling-window-in-seconds 120
```

O Lambda determina os limites da janela em cascata com base no horário em que os registros foram inseridos no stream. Todos os registros têm um carimbo de data/hora aproximado disponível que o Lambda usa para determinar os limites.

As agregações de janelas em cascata não são compatíveis com refragmentação. Quando o fragmento termina, o Lambda considera a janela como fechada e os fragmentos filhos iniciam suas próprias janelas em um novo estado.

As janelas em cascata são totalmente compatíveis com as políticas `maxRetryAttempts` e `maxRecordAge`.

Example Handler.py: agregação e processamento

A função do Python a seguir demonstra como agregar e, em seguida, processar seu estado final:

```
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}
```

Gerar relatórios de falhas de itens de lote

Ao consumir e processar dados de transmissão de uma fonte de eventos, o Lambda definirá checkpoints por padrão no número mais elevado na sequência de um lote somente quando o lote for um sucesso total. O Lambda trata todos os outros resultados como uma falha completa e tenta processar novamente o lote até o limite de novas tentativas. Para permitir sucessos parciais durante o processamento de lotes de um stream, ative `ReportBatchItemFailures`. Permitir sucessos parciais pode ajudar a reduzir o número de novas tentativas em um registro, embora não impeça totalmente a possibilidade de novas tentativas em um registro bem-sucedido.

Para ativar `ReportBatchItemFailures`, inclua o valor de enum `ReportBatchItemFailures` na lista `FunctionResponseTypes`. Essa lista indica quais tipos de resposta estão habilitados para sua função. Você pode configurar essa lista ao criar ou atualizar um [mapeamento de fonte de eventos \(p. 170\)](#).

Sintaxe do relatório

Ao configurar relatórios sobre falhas de itens de lote, a classe `StreamsEventResponse` é retornada com uma lista de falhas de itens de lote. É possível usar um objeto `StreamsEventResponse` para retornar o número sequencial do primeiro registro com falha no lote. Você também pode criar sua própria classe personalizada usando a sintaxe de resposta correta. A seguinte estrutura JSON mostra a sintaxe de resposta necessária:

```
{  
  "batchItemFailures": [  
    {  
      "itemIdentifier": "<id>"  
    }  
  ]  
}
```

Condições de sucesso e falha

O Lambda trata um lote como um sucesso completo se você retornar qualquer um destes:

- Uma lista de `batchItemFailure` vazia
- Uma lista de `batchItemFailure` nula
- Uma vazi `EventResponse`
- Uma nul `EventResponse`

O Lambda trata um lote como uma falha absoluta se você retornar qualquer um dos seguintes:

- Uma string vazi `itemIdentifier`
- Uma nul `itemIdentifier`
- Um `itemIdentifier` com um nome de chave inválido

O Lambda faz novas tentativas após falhas com base na sua estratégia de repetição.

Dividir um lote

Se a invocação falhar e `BisectBatchOnFunctionError` estiver ativado, o lote será dividido independentemente da configuração de `ReportBatchItemFailures`.

Quando uma resposta de sucesso parcial do lote é recebida e tanto `BisectBatchOnFunctionError` quanto `ReportBatchItemFailures` estão ativados, o lote é dividido no número de sequência retornado e o Lambda tenta novamente apenas os registros restantes.

Java

Example Handler.java: retornar um novo StreamsEventResponse()

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessDynamodbRecords implements RequestHandler<DynamodbEvent,
    Serializable> {

    @Override
    public Serializable handleRequest(DynamodbEvent input, Context context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
        ArrayList<*>();
        String curRecordSequenceNumber = "";

        for (DynamodbEvent.DynamodbEventRecord dynamodbEventRecord :
            input.getRecords()) {
            try {
                //Process your record
                DynamodbEvent.Record dynamodbRecord =
                dynamodbEventRecord.getDynamodb();
                curRecordSequenceNumber = dynamodbRecord.getSequenceNumber();

            } catch (Exception e) {
                //Return failed record's sequence number
                batchItemFailures.add(new
                StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
            }
        }

        return new StreamsEventResponse(batchItemFailures);
    }
}
```

Python

Example Handler.py: retornar batchItemFailures[]

```
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":[]}
```

Tutorial: Usar o AWS Lambda com o Amazon DynamoDB Streams

Neste tutorial, você cria uma função do Lambda para consumir eventos do Amazon DynamoDB Streams.

Prerequisites

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em [Conceitos básicos do Lambda \(p. 9\)](#) para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, você precisa de um terminal de linha de comando ou de um shell para executar os comandos. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

Você deve ver a saída a seguir:

```
aws-cli/2.0.57 Python/3.7.4 Darwin/19.6.0 exe/x86_64
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

Criar a função de execução

Crie a [função de execução \(p. 57\)](#) que dá à sua função permissão para acessar recursos do AWS.

Para criar uma função de execução

1. Abra a [página Roles](#) (Funções) no console do IAM.
2. Selecione Create role.
3. Crie uma função com as seguintes propriedades.
 - Entidade confiável: Lambda.
 - Permissions (Permissões): AWSLambdaDynamoDBExecutionRole.
 - Nome da função – **lambda-dynamodb-role**.

O AWSLambdaDynamoDBExecutionRole tem as permissões necessárias para a função ler itens do DynamoDB e gravar logs no CloudWatch Logs.

Criar a função

O código de amostra a seguir recebe uma entrada de evento do DynamoDB e processa as mensagens que ela contém. Na ilustração, o código grava alguns dos dados de entrada no CloudWatch Logs.

Note

Para o código de amostra em outras linguagens, consulte [Código de exemplo da função do \(p. 350\)](#).

Example index.js

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(function(record) {
        console.log(record.eventID);
        console.log(record.eventName);
        console.log('DynamoDB Record: %j', record.dynamodb);
    });
    callback(null, "message");
};
```

Para criar a função

1. Copie o código de amostra em um arquivo chamado `index.js`.
2. Crie um pacote de implantação.

```
zip function.zip index.js
```

3. Crie uma função do Lambda com o comando `create-function`.

```
aws lambda create-function --function-name ProcessDynamoDBRecords \
--zip-file file://function.zip --handler index.handler --runtime nodejs12.x \
--role arn:aws:iam::123456789012:role/lambda-dynamodb-role
```

Testar a função do Lambda

Nesta etapa, você invoca sua função do Lambda manualmente usando o comando `invoke` da CLI do AWS Lambda e o seguinte exemplo de evento do DynamoDB.

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"
    }
}
```

Execute o seguinte comando `invoke`.

```
aws lambda invoke --function-name ProcessDynamoDBRecords --payload file://input.txt
outfile.txt
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

A função retorna a string `message` no corpo da resposta.

Verifique a saída no arquivo `outfile.txt`.

Criar uma tabela do DynamoDB com uma transmissão habilitada

Criar uma tabela do Amazon DynamoDB com uma transmissão habilitada.

Como criar uma tabela do DynamoDB

1. Abra o [console do DynamoDB](#).
2. Escolha Create table.
3. Crie uma tabela com as configurações a seguir.
 - Table name (Nome da tabela – `lambda-dynamodb-stream`)
 - Primary key (Chave primária): `id` (string)
4. Escolha Create (Criar).

Como habilitar fluxos

1. Abra o [console do DynamoDB](#).
2. Escolha Tables (Tabelas).
3. Escolha a tabela `lambda-dynamodb-stream`.
4. Em Overview (Visão geral), escolha Manage stream (Gerenciar fluxo).
5. Escolha Enable (Habilitar).

Anote o ARN do fluxo. Você precisará dele na próxima etapa quando for associar a transmissão à função do Lambda. Para obter mais informações sobre como habilitar fluxos, consulte [Capturing table activity with DynamoDB Streams](#) (Capturar atividades de tabelas com o DynamoDB Streams).

Adicionar uma fonte de eventos no AWS Lambda

Crie um mapeamento da fonte do evento no AWS Lambda. Este mapeamento de fontes de eventos associa a transmissão do DynamoDB à função do Lambda. Depois de criar esse mapeamento da fonte do evento, o AWS Lambda começa a sondar o fluxo.

Execute o seguinte comando AWS CLI da `create-event-source-mapping`. Depois que o comando for executado, anote o UUID. Você precisará deste UUID para se referir ao mapeamento da fonte do evento em todos os comandos, por exemplo, ao excluir o mapeamento da fonte do evento.

```
aws lambda create-event-source-mapping --function-name ProcessDynamoDBRecords \
```

```
--batch-size 100 --starting-position LATEST --event-source DynamoDB-stream-arn
```

Isso cria um mapeamento entre a transmissão do DynamoDB especificado e a função do Lambda. Você pode associar uma transmissão do DynamoDB a várias funções do Lambda e associar a mesma função do Lambda a várias transmissões. No entanto, as funções do Lambda compartilharão a taxa de transferência de leitura para os fluxos dos quais compartilham.

Você pode obter a lista de mapeamentos de fontes de eventos executando o comando a seguir.

```
aws lambda list-event-source-mappings
```

Esta lista retorna todos os mapeamentos de fontes de eventos que você criou, e para cada mapeamento mostra o `LastProcessingResult`, entre outras coisas. Este campo será usado para fornecer uma mensagem informativa, se houver problemas. Valores como `No records processed` (indica que o AWS Lambda não iniciou a sondagem ou de que não há registros no fluxo) e `OK` (indica que o AWS Lambda foi bem-sucedido na leitura dos registros do fluxo e invocou a função do Lambda) indicam que não há problemas. Se houver problemas, você receberá uma mensagem de erro.

Se você tiver muitos mapeamentos da fonte do evento, use o parâmetro de nome da função para refinar os resultados.

```
aws lambda list-event-source-mappings --function-name ProcessDynamoDBRecords
```

Testar a configuração

Teste a experiência completa. À medida que você executa atualizações nas tabelas, o DynamoDB grava registros de eventos na transmissão. Quando o AWS Lambda sonda o stream, ele detecta novos registros no stream e executa a função do Lambda em seu nome, passando os eventos para a função.

1. No console do DynamoDB, adicione, atualize ou exclua itens da tabela. O DynamoDB grava registros dessas ações na transmissão.
2. O AWS Lambda sonda a transmissão e, ao detectar atualizações nela, invoca a função do Lambda passando os dados do evento que ele encontrar na transmissão.
3. A função é executada e cria logs no Amazon CloudWatch. Você pode verificar os logs relatados no console do Amazon CloudWatch.

Limpando recursos da

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua conta da AWS.

Como excluir a função do Lambda

1. Abra a página [Functions](#) (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Actions, Delete.
4. Escolha Delete.

Para excluir a função de execução

1. Abra a página [Roles](#) (Funções) no console do IAM.
2. Selecione a função de execução que você criou.

3. Selecione Delete role (Excluir função).
4. Escolha Sim, excluir.

Para excluir uma tabela do DynamoDB

1. Abra a [página Tables](#) (Tabelas) no console do DynamoDB.
2. Selecione a tabela que você criou.
3. Escolha Delete.
4. Digite **delete** na caixa de texto.
5. Escolha Delete.

Código de exemplo da função do

O código de amostra está disponível para as seguintes linguagens.

Tópicos

- [Node.js \(p. 350\)](#)
- [Java 11 \(p. 350\)](#)
- [C# \(p. 351\)](#)
- [Python 3 \(p. 352\)](#)
- [Go \(p. 352\)](#)

Node.js

O exemplo a seguir processa mensagens do DynamoDB e registra seu conteúdo.

Example ProcessDynamoDBStream.js

```
console.log('Loading function');

exports.lambda_handler = function(event, context, callback) {
    console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(function(record) {
        console.log(record.eventID);
        console.log(record.eventName);
        console.log('DynamoDB Record: %j', record.dynamodb);
    });
    callback(null, "message");
};
```

Compacta o código do exemplo para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Node.js com arquivos .zip \(p. 517\)](#).

Java 11

O exemplo a seguir processa mensagens do DynamoDB e registra seu conteúdo. O `handleRequest` é o manipulador que o AWS Lambda invoca e fornece dados do evento. O manipulador usa a classe predefinida `DynamodbEvent`, que é definida na biblioteca `aws-lambda-java-events`.

Example DDBEventProcessor.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 com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;

public class DDBEventProcessor implements
    RequestHandler<DynamodbEvent, String> {

    public String handleRequest(DynamodbEvent ddbEvent, Context context) {
        for (DynamodbStreamRecord record : ddbEvent.getRecords()){
            System.out.println(record.getEventID());
            System.out.println(record.geteventName());
            System.out.println(record.getDynamodb().toString());

        }
        return "Successfully processed " + ddbEvent.getRecords().size() + " records.";
    }
}
```

Se o manipulador obtém um retorno normal sem exceções, o Lambda considera que os lotes de entrada de registros foram processados com êxito e começa a ler novos registros no fluxo. Se o manipulador gera uma exceção, o Lambda considera que os lotes de entrada de registros não foram processados e invoca novamente a função com o mesmo lote de registros.

Dependencies

- aws-lambda-java-core
- aws-lambda-java-events

Crie o código com as dependências da biblioteca do Lambda para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Java com arquivos .zip ou JAR \(p. 599\)](#).

C#

O exemplo a seguir processa mensagens do DynamoDB e registra seu conteúdo. O `ProcessDynamoEvent` é o manipulador que o AWS Lambda invoca e fornece dados do evento. O manipulador usa a classe predefinida `DynamoDbEvent`, que é definida na biblioteca `Amazon.Lambda.DynamoDBEvents`.

Example ProcessingDynamoDBStreams.cs

```
using System;
using System.IO;
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.DynamoDBEvents;

using Amazon.Lambda.Serialization.Json;

namespace DynamoDBStreams
{
    public class DdbSample
    {
        private static readonly JsonSerializer _jsonSerializer = new JsonSerializer();

        public void ProcessDynamoEvent(DynamoDBEvent dynamoEvent)
        {
            Console.WriteLine($"Beginning to process {dynamoEvent.Records.Count} records...");
        }
    }
}
```

```
foreach (var record in dynamoEvent.Records)
{
    Console.WriteLine($"Event ID: {record.EventID}");
    Console.WriteLine($"Event Name: {record.EventName}");

    string streamRecordJson = SerializeObject(record.Dynamodb);
    Console.WriteLine($"DynamoDB Record:");
    Console.WriteLine(streamRecordJson);
}

Console.WriteLine("Stream processing complete.");
}

private string SerializeObject(object streamRecord)
{
    using (var ms = new MemoryStream())
    {
        _jsonSerializer.Serialize(streamRecord, ms);
        return Encoding.UTF8.GetString(ms.ToArray());
    }
}
}
```

Substitua o `Program.cs` em um projeto do .NET Core pelo exemplo acima. Para obter instruções, consulte [Implantar funções do Lambda em C# com arquivos .zip \(p. 668\)](#).

Python 3

O exemplo a seguir processa mensagens do DynamoDB registrando seu conteúdo.

Example `ProcessDynamoDBStream.py`

```
from __future__ import print_function

def lambda_handler(event, context):
    for record in event['Records']:
        print(record['eventID'])
        print(record['eventName'])
    print('Successfully processed %s records.' % str(len(event['Records'])))
```

Compacte o código do exemplo para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Python com arquivos .zip \(p. 543\)](#).

Go

O exemplo a seguir processa mensagens do DynamoDB registrando seu conteúdo.

Example

```
import (
    "strings"

    "github.com/aws/aws-lambda-go/events"
)

func handleRequest(ctx context.Context, e events.DynamoDBEvent) {
    for _, record := range e.Records {
```

```
    fmt.Printf("Processing request data for event ID %s, type %s.\n", record.EventID,
record.EventName)

    // Print new values for attributes of type String
    for name, value := range record.Change.NewImage {
        if value.DataType() == events.DataTypeString {
            fmt.Printf("Attribute name: %s, value: %s\n", name, value.String())
        }
    }
}
```

Crie o executável com o `go build` e crie um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Go com arquivos .zip](#) (p. 640).

Modelo do AWS SAM para uma aplicação do DynamoDB

Você pode criar esse aplicativo usando [AWS SAM](#). Para saber mais sobre como criar modelos do AWS SAM, consulte [Noções básicas de modelos do AWS SAM](#) no Guia do desenvolvedor do AWS Serverless Application Model.

Veja abaixo um modelo de exemplo do AWS SAM para o [aplicativo do tutorial \(p. 345\)](#). Copie o texto abaixo para um arquivo `.yaml` e salve-o ao lado do pacote ZIP criado previamente. Observe que os valores dos parâmetros `Handler` e `Runtime` devem corresponder àqueles usados quando você criou a função na seção anterior.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  ProcessDynamoDBStream:
    Type: AWS::Serverless::Function
    Properties:
      Handler: handler
      Runtime: runtime
      Policies: AWSLambdaDynamoDBExecutionRole
      Events:
        Stream:
          Type: DynamoDB
          Properties:
            Stream: !GetAtt DynamoDBTable.StreamArn
            BatchSize: 100
            StartingPosition: TRIM_HORIZON

  DynamoDBTable:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        -AttributeName: id
        AttributeType: S
      KeySchema:
        -AttributeName: id
        KeyType: HASH
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5
      StreamSpecification:
        StreamViewType: NEW_IMAGE
```

Para obter informações sobre como empacotar e implantar o aplicativo sem servidor usando os comandos de empacotamento e implantação, consulte [Implantar aplicativos sem servidor](#) no Guia do desenvolvedor do AWS Serverless Application Model.

O uso do AWS Lambda com o Amazon EC2

Você pode usar o AWS Lambda para processar eventos de ciclo de vida do Amazon Elastic Compute Cloud e gerenciar recursos do Amazon EC2. O Amazon EC2 envia eventos ao Amazon CloudWatch Events para eventos de ciclo de vida, como quando uma instância muda de estado, quando um snapshot de volume do Amazon Elastic Block Store é concluído ou quando uma instância spot está programada para ser encerrada. Configure o CloudWatch Events para encaminhar esses eventos para uma função do Lambda para processamento.

O CloudWatch Events invoca sua função do Lambda de forma assíncrona com o documento de evento no Amazon EC2.

Example evento do ciclo de vida da instância

```
{  
    "version": "0",  
    "id": "b6ba298a-7732-2226-xmpl-976312c1a050",  
    "detail-type": "EC2 Instance State-change Notification",  
    "source": "aws.ec2",  
    "account": "123456798012",  
    "time": "2019-10-02T17:59:30Z",  
    "region": "us-east-2",  
    "resources": [  
        "arn:aws:ec2:us-east-2:123456798012:instance/i-0c314xmplcd5b8173"  
    ],  
    "detail": {  
        "instance-id": "i-0c314xmplcd5b8173",  
        "state": "running"  
    }  
}
```

Para obter detalhes sobre como configurar eventos no CloudWatch Events, consulte [O uso do AWS LambdaCom Amazon CloudWatch Events \(p. 315\)](#). Para obter uma função de exemplo que processa notificações de snapshot do Amazon EBS, consulte [Amazon CloudWatch Events para Amazon EBS](#) no Manual do usuário do Amazon EC2 para instâncias do Linux.

Você também pode usar o AWS SDK para gerenciar instâncias e outros recursos com a API do Amazon EC2. Para obter um tutorial com um aplicativo de exemplo em C#, consulte [Tutorial: usar o AWS SDK for .NET para gerenciar instâncias spot do Amazon EC2 \(p. 356\)](#).

Permissions

Para processar eventos de ciclo de vida do Amazon EC2, o CloudWatch Events precisa de permissão para invocar sua função. Essa permissão vem da [política baseada em recursos \(p. 62\)](#) da função. Se você usar o console do CloudWatch Events para configurar um gatilho de evento, o console atualizará a política baseada em recursos em seu nome. Caso contrário, adicione uma instrução como a seguinte:

Example instrução de política baseada em recursos para notificações de ciclo de vida do Amazon EC2

```
{  
    "Sid": "ec2-events",  
    "Effect": "Allow",  
    "Principal": {  
        "Service": "events.amazonaws.com"  
    },  
    "Action": "lambda:InvokeFunction",  
    "Resource": "  
        arn:aws:lambda:  
        us-east-2:  
        123456798012:  
        function:  
        myLambdaFunction  
    "
```

```
"Resource": "arn:aws:lambda:us-east-2:12456789012:function:my-function",
"Condition": {
    "ArnLike": {
        "AWS:SourceArn": "arn:aws:events:us-east-2:12456789012:rule/*"
    }
}
```

Para adicionar uma instrução, use o comando `add-permission` da AWS CLI.

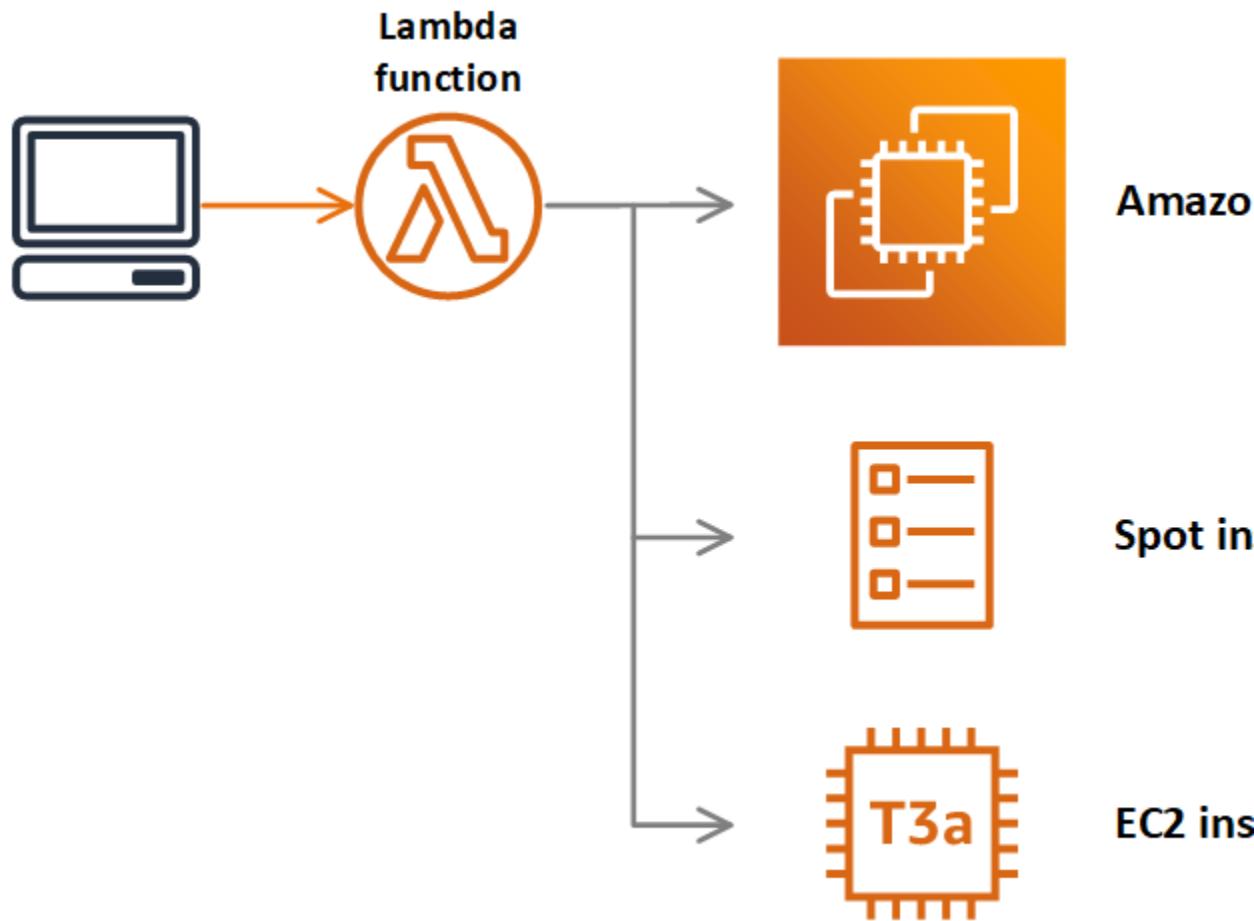
```
aws lambda add-permission --action lambda:InvokeFunction --statement-id ec2-events \
--principal events.amazonaws.com --function-name my-function --source-arn
'arn:aws:events:us-east-2:12456789012:rule/*'
```

Se sua função usar o AWS SDK para gerenciar recursos do Amazon EC2, adicione permissões do Amazon EC2 à [função de execução \(p. 57\)](#) da função.

Tutorial: usar o AWS SDK for .NET para gerenciar instâncias spot do Amazon EC2

Você pode usar o AWS SDK for .NET para gerenciar instâncias spot do Amazon EC2 com código C#. O SDK permite que você use a API do Amazon EC2 para criar solicitações de instância spot, determinar quando a solicitação é atendida, excluir solicitações e identificar as instâncias criadas.

Este tutorial fornece código que executa essas tarefas e uma aplicação de exemplo que você pode executar localmente ou na AWS. Ele inclui um projeto de exemplo que você pode implantar no tempo de execução do .NET Core 2.1 do AWS Lambda.



Para obter mais informações sobre o uso de instâncias spot e as práticas recomendadas, consulte [Instâncias spot](#) no guia do usuário do Amazon EC2.

Prerequisites

Para concluir as etapas a seguir, você precisa de um terminal de linha de comando ou de um shell para executar os comandos. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

Você deve ver a saída a seguir:

```
aws-cli/2.0.57 Python/3.7.4 Darwin/19.6.0 exe/x86_64
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

Este tutorial usa o código do repositório GitHub do guia do desenvolvedor. O repositório também contém scripts auxiliares e arquivos de configuração que são necessários para seguir seus procedimentos. Clone o repositório em github.com/awsdocs/aws-lambda-developer-guide.

Para usar o código de exemplo, você precisa das seguintes ferramentas:

- AWS CLI: para implantar a aplicação de exemplo na AWS, instale a [AWS CLI](#). A AWS CLI também fornece credenciais para o código de exemplo quando você o executa localmente.
- CLI do .NET Core: para executar e testar o código localmente, instale o [.NET Core SDK 2.1](#).
- Ferramenta global do .NET Core do Lambda: para criar o pacote de implantação para o Lambda, instale a [ferramenta global do .NET Core](#) com a CLI .NET Core.

```
dotnet tool install -g Amazon.Lambda.Tools
```

O código neste tutorial gerencia solicitações spot que executam instâncias do Amazon EC2. Para executar o código localmente, você precisa de credenciais de SDK com permissão para usar as seguintes APIs.

- `ec2:RequestSpotInstance`
- `ec2:GetSpotRequestState`
- `ec2:CancelSpotRequest`
- `ec2:TerminateInstances`

Para executar a aplicação de exemplo na AWS, você precisa de [permissão para usar o Lambda \(p. 56\)](#) e os serviços a seguir.

- [AWS CloudFormation \(definição de preço\)](#)
- [Amazon Elastic Compute Cloud\(Definição de preço do\)](#)

Aplicam-se taxas padrão para cada serviço.

Revisar o código

Localize o projeto de amostra no repositório guia em [sample-apps/ec2-spot](#). Este diretório contém código de função do Lambda, testes, arquivos de projeto, scripts e um modelo do AWS CloudFormation.

A classe `Function` inclui um método `FunctionHandler` que chama outros métodos para criar solicitações spot, verificar seu status e limpar. Ele cria um cliente do Amazon EC2 com o AWS SDK for .NET em um construtor estático para permitir que ele seja usado em toda a classe.

Example `Function.cs: FunctionHandler`

```
using Amazon.EC2;
...
    public class Function
    {
        private static AmazonEC2Client ec2Client;

        static Function()
        {
            AWSSDKHandler.RegisterXRayForAllServices();
            ec2Client = new AmazonEC2Client();
        }

        public async Task<string> FunctionHandler(Dictionary<string, string> input,
ILambdaContext context)
        {
            // More AMI IDs: amazon-linux-2/release-notes/

```

```
// us-east-2 HVM EBS-Backed 64-bit Amazon Linux 2
string ami = "ami-09d9edae5eb90d556";
string sg = "default";
InstanceType type = InstanceType.T3aNano;
string price = "0.003";
int count = 1;
var requestSpotInstances = await RequestSpotInstance(ami, sg, type, price,
count);
var spotRequestId =
requestSpotInstances.SpotInstanceRequests[0].SpotInstanceRequestId;
```

O método `RequestSpotInstance` cria uma solicitação de instância spot.

Example Function.cs: RequestSpotInstance

```
using Amazon;
using Amazon.Util;
using Amazon.EC2;
using Amazon.EC2.Model;
...
public async Task<RequestSpotInstancesResponse> RequestSpotInstance(
    string amiId,
    string securityGroupName,
    InstanceType instanceType,
    string spotPrice,
    int instanceCount)
{
    var request = new RequestSpotInstancesRequest();

    var launchSpecification = new LaunchSpecification();
    launchSpecification.ImageId = amiId;
    launchSpecification.InstanceType = instanceType;
    launchSpecification.SecurityGroups.Add(securityGroupName);

    request.SpotPrice = spotPrice;
    request.InstanceCount = instanceCount;
    request.LaunchSpecification = launchSpecification;

    RequestSpotInstancesResponse response = await
ec2Client.RequestSpotInstancesAsync(request);

    return response;
}
...
```

Em seguida, é preciso esperar até que a solicitação spot alcance o estado `Active` antes de continuar para a última etapa. Para determinar o estado da solicitação spot, use o método `DescribeSpotInstanceRequests` para obter o estado do ID da solicitação spot a ser monitorado.

```
public async Task<SpotInstanceRequest> GetSpotRequest(string spotRequestId)
{
    var request = new DescribeSpotInstanceRequestsRequest();
    request.SpotInstanceRequestIds.Add(spotRequestId);

    var describeResponse = await ec2Client.DescribeSpotInstanceRequestsAsync(request);

    return describeResponse.SpotInstanceRequests[0];
}
```

A etapa final é limpar suas solicitações e instâncias. É importante cancelar todas as solicitações pendentes e encerrar as instâncias. Simplesmente cancelar as solicitações não encerrará as instâncias, o que

significa que você continuará a ser cobrado por elas. Se você encerrar suas instâncias suas solicitações spot poderão ser canceladas, mas há algumas situações, por exemplo, se você usar solicitações persistentes, nas quais encerrar suas instâncias não basta para impedir que a solicitação seja atendida novamente. Portanto, é uma prática recomendada cancelar todas as solicitações ativas e encerrar as instâncias em execução.

Para cancelar uma solicitação spot, você usa o método [CancelSpotInstanceRequests](#). O exemplo a seguir demonstra como cancelar uma solicitação spot.

```
public async Task CancelSpotRequest(string spotRequestId)
{
    Console.WriteLine("Canceling request " + spotRequestId);
    var cancelRequest = new CancelSpotInstanceRequestsRequest();
    cancelRequest.SpotInstanceRequestIds.Add(spotRequestId);

    await ec2Client.CancelSpotInstanceRequestsAsync(cancelRequest);
}
```

Para encerrar uma instância, você usa o método [TerminateInstances](#).

```
public async Task TerminateSpotInstance(string instanceId)
{
    Console.WriteLine("Terminating instance " + instanceId);
    var terminateRequest = new TerminateInstancesRequest();
    terminateRequest.InstanceIds = new List<string>() { instanceId };
    try
    {
        var terminateResponse = await ec2Client.TerminateInstancesAsync(terminateRequest);
    }
    catch (AmazonEC2Exception ex)
    {
        // Check the ErrorCode to see if the instance does not exist.
        if ("InvalidInstanceID.NotFound" == ex.ErrorCode)
        {
            Console.WriteLine("Instance {0} does not exist.", instanceId);
        }
        else
        {
            // The exception was thrown for another reason, so re-throw the exception.
            throw;
        }
    }
}
```

Executar o código localmente

Execute o código em sua máquina local para criar uma solicitação de instância spot. Depois que a solicitação é atendida, o código exclui a solicitação e encerra a instância.

Para executar o código do aplicativo

1. Navegue até o diretório `ec2Spot.Tests`.

```
cd test/ec2Spot.Tests
```

2. Use a CLI .NET para executar os testes de unidade do projeto.

```
dotnet test
```

Você deve ver a saída a seguir:

```
Starting test execution, please wait...
sir-x5tgs5ij
open
open
open
open
open
active
Canceling request sir-x5tgs5ij
Terminating instance i-0b3fdff0e12e0897e
Complete

Test Run Successful.
Total tests: 1
    Passed: 1
Total time: 7.6060 Seconds
```

O teste de unidade chama o método FunctionHandler para criar uma solicitação de instância spot, monitorá-la e limpar. Ele é implementado no framework de teste [xUnit.net](#).

Implantar o aplicativo

Execute o código no Lambda como um ponto de partida para criar uma aplicação sem servidor.

Para implantar e testar o aplicativo

1. Defina sua região como us-east-2.

```
export AWS_DEFAULT_REGION=us-east-2
```

2. Crie um bucket para artefatos de implantação.

```
./create-bucket.sh
```

Você deve ver a saída a seguir:

```
make_bucket: lambda-artifacts-63d5cbbf18fa5ecc
```

3. Crie um pacote de implantação e implante o aplicativo.

```
./deploy.sh
```

Você deve ver a saída a seguir:

```
Amazon Lambda Tools for .NET Core applications (3.3.0)
Project Home: https://github.com/aws/aws-extensions-for-dotnet-cli, https://github.com/
aws/aws-lambda-dotnet

Executing publish command
...
Created publish archive (ec2spot.zip)
Lambda project successfully packaged: ec2spot.zip
Uploading to ebd38e401ce7d676d05d22b76f0209 1305107 / 1305107.0 (100.00%)
Successfully packaged artifacts and wrote output template to file out.yaml.
```

```
Run the following command to deploy the packaged template
aws cloudformation deploy --template-file out.yaml --stack-name <YOUR STACK NAME>
```

```
Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - ec2-spot
```

4. Abra a página [Applications](#) (Aplicações) do console do Lambda.

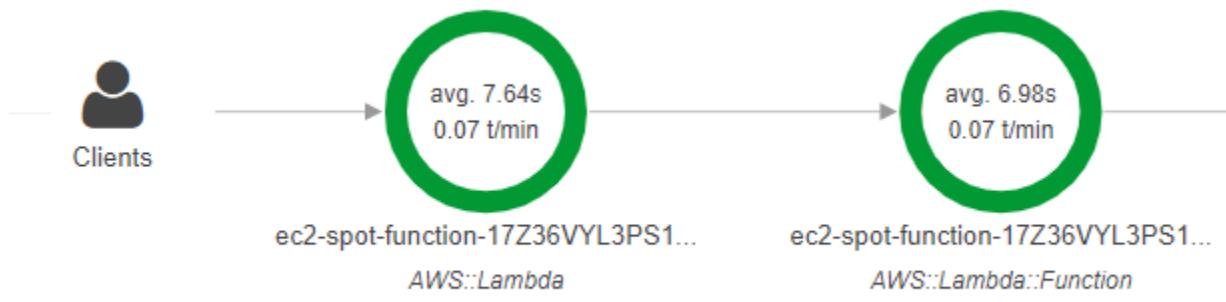
The screenshot shows the AWS Lambda console interface for the 'ec2-spot' stack. At the top, there are three tabs: 'Overview' (which is selected), 'Deployments', and 'Monitoring'. Below the tabs, a 'Getting started' section is displayed. Under the 'Resources' heading, there are two entries:

Logical ID	Physical ID	Type
function	ec2-spot-function-17Z36VYL3PS14	Lambda Function
role	ec2-spot-role-1TDCWJ2ZNNF1M	IAM Role

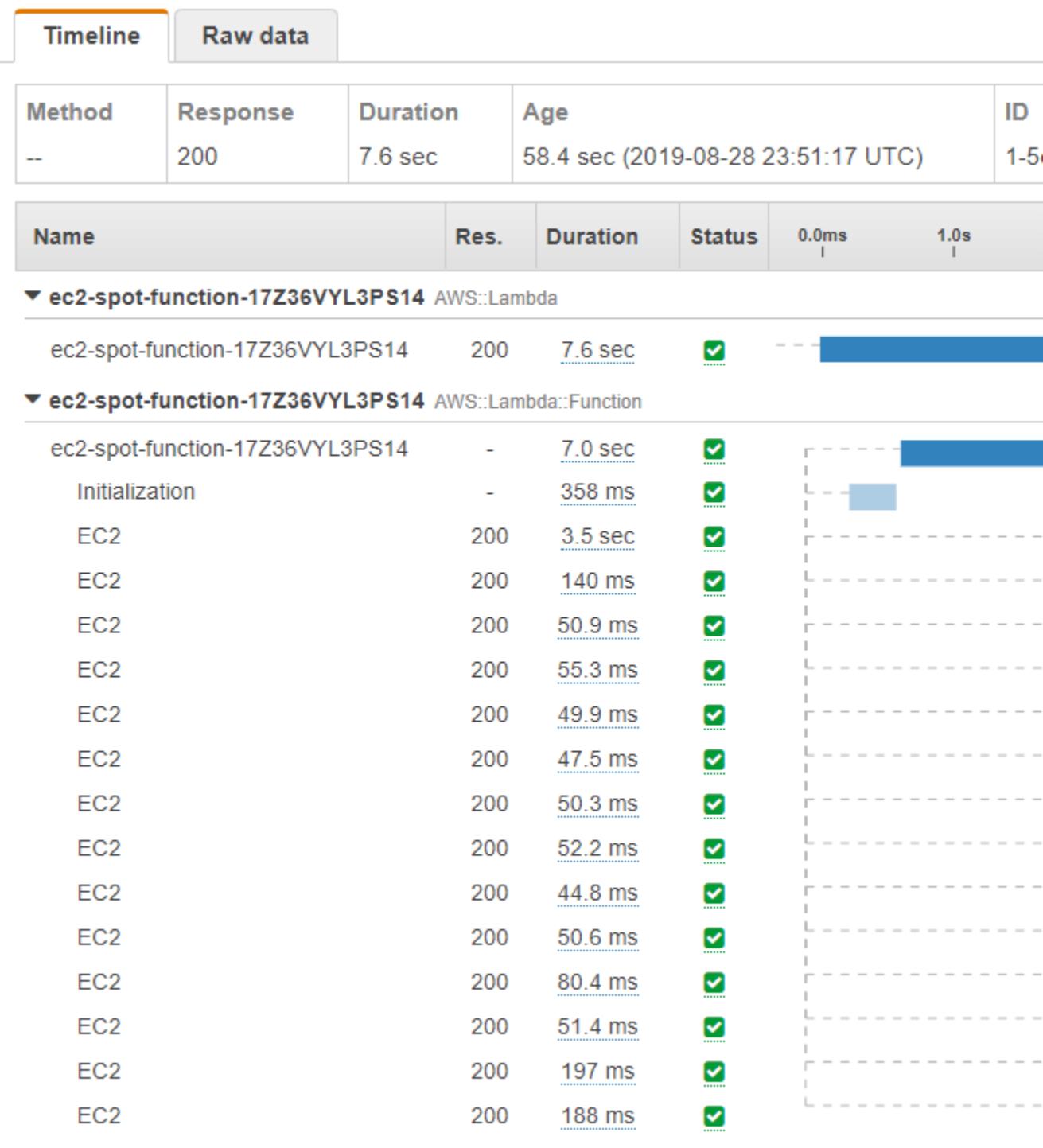
5. Em Resources (Recursos), escolha function (função).
6. Escolha Test (Testar) e crie um evento de teste a partir do modelo padrão.
7. Escolha Test (Testar) novamente para invocar a função.

Exiba os logs e informações de rastreamento para ver o ID de solicitação spot e a sequência de chamadas para o Amazon EC2.

Para exibir o mapa de serviço, abra a [página Service map](#) (Mapa de serviço) no console do X-Ray.



Escolha um nó no mapa de serviço e escolha View traces (Visualizar rastreamentos) para ver uma lista de rastreamentos. Escolha um rastreamento na lista para ver a linha do tempo das chamadas feitas pela função para o Amazon EC2.



Limpar

O código fornecido neste tutorial foi projetado para criar e excluir solicitações de instância spot e para encerrar as instâncias executadas. No entanto, se ocorrer um erro, as solicitações e instâncias podem não ser limpas automaticamente. Exiba as solicitações spot e as instâncias no console do Amazon EC2.

Para confirmar que os recursos do Amazon EC2 estão limpos

1. Abra a [página Spot Requests](#) (Solicitações spot) no console do Amazon EC2.
2. Verifique se o estado das solicitações é Cancelled (Cancelado).
3. Escolha o ID da instância na coluna Capacity (Capacidade) para exibir a instância.
4. Verifique se o estado das instâncias é Terminated (Encerrado) ou Shutting down (Desligando).

Para limpar a função de exemplo e os recursos de suporte, exclua a pilha do AWS CloudFormation e o bucket de artefatos que você criou.

```
./cleanup.sh
```

Você deve ver a saída a seguir:

```
Delete deployment artifacts and bucket (lambda-artifacts-63d5cbbf18fa5ecc)?y
delete: s3://lambda-artifacts-63d5cbbf18fa5ecc/ebd38e401cedd7d676d05d22b76f0209
remove_bucket: lambda-artifacts-63d5cbbf18fa5ecc
```

O grupo de log da função não é excluído automaticamente. Você pode excluí-lo no[Console CloudWatch Logs](#). Rastreamentos no X-Ray expiram após algumas semanas e são excluídos automaticamente.

Tutorial: configurar uma função do Lambda para acessar o Amazon ElastiCache em uma Amazon VPC

Neste tutorial, você faz o seguinte:

- Crie um cluster do Amazon ElastiCache na Amazon Virtual Private Cloud padrão. Para obter mais informações sobre o Amazon ElastiCache, consulte [Amazon ElastiCache](#).
- Crie uma função do Lambda para acessar o cluster do ElastiCache. Ao criar a função do Lambda, você fornece os IDs de sub-rede da sua Amazon VPC e grupo de segurança de VPC para permitir que a função do Lambda acesse os recursos na sua VPC. Para ilustração neste tutorial, a função do Lambda gera um UUID, o grava em cache e o recupera do cache.
- invoque a função do Lambda e verifique se ela acessou o cluster do ElastiCache na sua VPC.

Para obter detalhes sobre o uso do Lambda com a Amazon VPC, consulte [Configurar uma função do Lambda para acessar recursos em uma VPC \(p. 124\)](#).

Prerequisites

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em [Conceitos básicos do Lambda \(p. 9\)](#) para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, você precisa de um terminal de linha de comando ou de um shell para executar os comandos. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

Você deve ver a saída a seguir:

```
aws-cli/2.0.57 Python/3.7.4 Darwin/19.6.0 exe/x86_64
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

Criar a função de execução

Crie a [função de execução \(p. 57\)](#) que dá à sua função permissão para acessar recursos do AWS.

Para criar uma função de execução

1. Abra a [página Roles](#) (Funções) no console do IAM.
2. Selecione Create role.
3. Crie uma função com as seguintes propriedades.
 - Entidade confiável: Lambda.

- Permissions (Permissões): AWSLambdaVPCAccessExecutionRole.
- Nome da função – **lambda-vpc-role**.

A AWSLambdaVPCAccessExecutionRole tem as permissões que a função precisa para gerenciar conexões de rede com uma VPC.

Criar um cluster do ElastiCache

Crie um cluster do ElastiCache na sua VPC padrão.

1. Execute o seguinte comando da AWS CLI para criar um cluster Memcached.

```
aws elasticache create-cache-cluster --cache-cluster-id ClusterForLambdaTest --cache-node-type cache.t3.medium --engine memcached --num-cache-nodes 1 --security-group-ids sg-0123a1b123456c1de
```

Você pode examinar o security group do VPC padrão no console do VPC em Security Groups. Sua função Lambda de exemplo adicionará e recuperará um item desse cluster.

2. Anote o endpoint de configuração para o cluster de cache que você iniciou. Você pode obter isso no console do Amazon ElastiCache. Você poderá especificar esse valor no código da sua função do Lambda, na próxima seção.

Criar um pacote de implantação

O exemplo a seguir do código Python lê e grava um item no seu cluster do ElastiCache.

Example app.py

```
from __future__ import print_function
import time
import uuid
import sys
import socket
import elasticache_auto_discovery
from pymemcache.client.hash import HashClient

#elasticache settings
elasticache_config_endpoint = "your-elastichache-cluster-endpoint:port"
nodes = elasticache_auto_discovery.discover(elasticache_config_endpoint)
nodes = map(lambda x: (x[1], int(x[2])), nodes)
memcache_client = HashClient(nodes)

def handler(event, context):
    """
    This function puts into memcache and get from it.
    Memcache is hosted using elasticache
    """

    #Create a random UUID... this will be the sample element we add to the cache.
    uuid_inserted = uuid.uuid4().hex
    #Put the UUID to the cache.
    memcache_client.set('uuid', uuid_inserted)
    #Get item (UUID) from the cache.
    uuid_obtained = memcache_client.get('uuid')
    if uuid_obtained.decode("utf-8") == uuid_inserted:
        # this print should go to the CloudWatch Logs and Lambda console.
        print ("Success: Fetched value %s from memcache" %(uuid_inserted))
```

```
else:  
    raise Exception("Value is not the same as we put :(. Expected %s got %s"  
%(uuid_inserted, uuid_obtained))  
  
return "Fetched value from memcache: " + uuid_obtained.decode("utf-8")
```

Dependencies

- [pymemcache](#): o código da função do Lambda usa essa biblioteca para criar um objeto `HashClient` para definir e obter itens do cache de memória.
- [elasticache-auto-discovery](#): a função do Lambda usa essa biblioteca para obter os nós no seu cluster do Amazon ElastiCache.

Instale dependências com o Pip e crie um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Python com arquivos .zip \(p. 543\)](#).

Criar a função do Lambda

Crie a função do Lambda com o comando `create-function`.

```
aws lambda create-function --function-name AccessMemCache --timeout 30 --memory-size 1024 \  
--zip-file fileb://function.zip --handler app.handler --runtime python3.8 \  
--role arn:aws:iam::123456789012:role/lambda-vpc-role \  
--vpc-config SubnetIds=subnet-0532bb6758ce7c71f,subnet-  
d6b7fda068036e11f,SecurityGroupIds=sg-0897d5f549934c2fb
```

Você pode encontrar os IDs de sub-rede e o ID do security group padrão do seu VPC no console do VPC.

Testar a função do Lambda

Nesta etapa, você invoca a função do Lambda manualmente usando o comando `invoke`. Quando a função do Lambda for executada, ela gerará um UUID e o gravará no cluster do ElastiCache que você especificou no seu código do Lambda. A função Lambda, então, recupera o item do cache.

1. Invoque a função do Lambda com o comando `invoke`.

```
aws lambda invoke --function-name AccessMemCache output.txt
```

2. Verifique se a função do Lambda foi executada com êxito, da seguinte forma:

- Analise o arquivo `output.txt`.
- Analisar os resultados no console do AWS Lambda.
- Verifique os resultados no CloudWatch Logs.

Agora que você criou uma função do Lambda que acessa um cluster do ElastiCache na sua VPC, a função já poderá ser invocada em resposta a eventos. Para obter informações sobre como configurar fontes de eventos e exemplos, consulte [Usar o AWS Lambda com outros serviços \(p. 277\)](#).

Limpar recursos

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua conta da AWS.

Como excluir a função do Lambda

1. Abra a página [Functions](#) (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Actions (Ações), Delete (Excluir).
4. Escolha Delete.

Para excluir a função de execução

1. Abra a página [Roles](#) (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Selecione Delete role (Excluir função).
4. Escolha Sim, excluir.

Para excluir o cluster do ElastiCache

1. Abrir o[Memcached](#)do console do ElastiCache.
2. Selecione o cluster que você criou.
3. Escolha Actions (Ações), Delete (Excluir).
4. Escolha Delete.

Usar o AWS Lambda com um平衡ador de carga da aplicação

Você pode usar uma função do Lambda para processar solicitações de um balanceador de carga da aplicação. O Elastic Load Balancing é compatível com funções do Lambda como destino para um balanceador de carga da aplicação. Use regras de load balancer para rotear solicitações HTTP para uma função, com base no caminho ou em valores de cabeçalho. Processe a solicitação e retorne uma resposta HTTP de sua função do Lambda.

O Elastic Load Balancing invoca a função do Lambda de forma síncrona com um evento que contém o corpo da solicitação e os metadados.

Example Evento de solicitação do balanceador de carga da aplicação

```
{  
    "requestContext": {  
        "elb": {  
            "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:123456789012:targetgroup/lambdapgZ5rsrHC2Fjr/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-123456789012.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  
}
```

Sua função processa o evento e retorna um documento de resposta para o balanceador de carga no JSON. O Elastic Load Balancing converte o documento em uma resposta de sucesso ou de erro de HTTP e o devolve ao usuário.

Example formato do documento de resposta

```
{  
    "statusCode": 200,  
    "statusDescription": "200 OK",  
    "isBase64Encoded": false,  
    "headers": {  
        "Content-Type": "text/html"  
    },  
    "body": "

Success!

"  
}
```

```
        "body": "<h1>Hello from Lambda!</h1>"  
    }
```

Para configurar um balanceador de carga da aplicação como um acionador de função, conceda ao Elastic Load Balancing permissão para executar a função, crie um grupo de destino que roteie as solicitações para a função e adicione uma regra ao balanceador de carga que envie solicitações para o grupo de destino.

Use o comando `add-permission` para adicionar uma instrução de permissão à política baseada em recursos de sua função.

```
aws lambda add-permission --function-name alb-function \  
--statement-id load-balancer --action "lambda:InvokeFunction" \  
--principal elasticloadbalancing.amazonaws.com
```

Você deve ver a saída a seguir:

```
{  
    "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\""}  
}
```

Para obter instruções sobre como configurar o listener e um grupo de destino do balanceador de carga da aplicação, consulte [Funções do Lambda como destino](#) no Guia do usuário para平衡adores de carga da aplicação.

Uso do Amazon EFS com o Lambda

O Lambda se integra ao Amazon Elastic File System (Amazon EFS) para oferecer acesso seguro e compartilhado ao sistema de arquivos para aplicativos do Lambda. É possível configurar funções para montar um sistema de arquivos durante a inicialização com o protocolo NFS na rede local em uma VPC. O Lambda gerencia a conexão e criptografa todo o tráfego de e para o sistema de arquivos.

O sistema de arquivos e a função do Lambda devem estar na mesma região. Uma função do Lambda em uma conta pode montar um sistema de arquivos em uma conta diferente. Nesse cenário, você configura um emparelhamento de VPCs entre a VPC da função e a VPC do sistema de arquivos.

Note

Para configurar uma função para se conectar a um sistema de arquivos, consulte [Configurar o acesso ao sistema de arquivos para funções do Lambda \(p. 139\)](#).

O Amazon EFS oferece suporte ao [bloqueio de arquivos](#) para evitar a corrupção se várias funções tentarem gravar no mesmo sistema de arquivos ao mesmo tempo. O bloqueio do Amazon EFS segue o protocolo NFS v4.1 para bloqueios recomendados, além de permitir que seus aplicativos usem bloqueios de arquivo inteiro ou de intervalos de bytes.

O Amazon EFS fornece opções para personalizar o sistema de arquivos com base na necessidade da aplicação de manter a alta performance em escala. Há três principais fatores a serem considerados: o número de conexões, a taxa de transferência (em MiB por segundo) e as IOPS.

Quotas

Para obter detalhes sobre cotas e limites do sistema de arquivos, consulte [Cotas para sistemas de arquivos do Amazon EFS](#) no Manual do usuário do Amazon Elastic File System.

Para evitar problemas de escalabilidade, taxa de transferência e IOPS, monitore as [métricas](#) que o Amazon EFS envia para o Amazon CloudWatch. Para obter uma visão geral do monitoramento no Amazon EFS, consulte [Monitoramento do EFS](#) no Guia do Usuário do Amazon Elastic File System.

Seções

- [Connections \(p. 372\)](#)
- [Throughput \(p. 373\)](#)
- [IOPS \(p. 373\)](#)

Connections

O Amazon EFS oferece suporte para até 25.000 conexões por sistema de arquivos. Durante a inicialização, cada instância de uma função cria uma única conexão com o sistema de arquivos que persiste entre invocações. Isso significa que você pode alcançar a simultaneidade de 25.000 em uma ou mais funções conectadas a um sistema de arquivos. Para limitar o número de conexões criadas por uma função, use a [simultaneidade reservada \(p. 113\)](#).

No entanto, quando você faz alterações no código ou na configuração da função em escala, ocorre um aumento temporário no número de instâncias da função além da simultaneidade atual. O Lambda provisão novas instâncias para lidar com novas solicitações e existe um atraso antes que as instâncias antigas fechem suas conexões com o sistema de arquivos. Para evitar atingir o limite máximo de conexões durante uma implantação, use [implantações contínuas \(p. 206\)](#). Com implantações contínuas, você gradualmente muda o tráfego para a nova versão sempre que fizer uma alteração.

Se você se conectar ao mesmo sistema de arquivos de outros serviços, como o Amazon EC2, esteja ciente também do comportamento de escalabilidade das conexões no Amazon EFS. Um sistema de

arquivos oferece suporte para a criação de até 3.000 conexões em uma intermitência, após a qual oferece suporte para 500 novas conexões por minuto. Isso corresponde ao comportamento da [escalabilidade de intermitência \(p. 32\)](#) no Lambda, que se aplica a todas as funções em uma região. Mas se você estiver criando conexões fora do Lambda, as funções podem não ser capazes de escalar na velocidade total.

Para monitorar e acionar um alarme nas conexões, use a métrica `ClientConnections`.

Throughput

Em escala, também é possível exceder a taxa de transferência máxima de um sistema de arquivos. No modo intermitente (padrão), um sistema de arquivos tem uma baixa taxa de transferência de linha de base que é dimensionada linearmente com o seu tamanho. Para permitir intermitências de atividade, o sistema de arquivos recebe créditos de intermitência que permitem o uso de 100 MiB/s ou mais de taxa de transferência. Os créditos são acumulados continuamente e são gastos em cada operação de leitura e gravação. Se o sistema de arquivos ficar sem créditos, ele limitará as operações de leitura e gravação que ultrapassarem a taxa de transferência de linha de base, o que pode fazer com que as invocações atinjam o tempo limite.

Note

Se você usar a [simultaneidade provisionada \(p. 113\)](#), a função poderá consumir créditos de intermitência mesmo quando estiver ociosa. Com a simultaneidade provisionada, o Lambda inicializa as instâncias da função antes dela ser invocada e recicla as instâncias em intervalos regulares. Se você usar arquivos em um sistema de arquivos anexado durante a inicialização, essa atividade poderá usar todos os créditos de intermitência.

Para monitorar e acionar um alarme na taxa de transferência, use a métrica `BurstCreditBalance`. Ela deverá aumentar quando a simultaneidade da função estiver baixa e diminuir quando ela estiver alta. Se ela sempre diminuir ou não acumular durante a baixa atividade o suficiente para cobrir o tráfego de pico, talvez seja necessário limitar a simultaneidade da função ou ativar a [taxa de transferência provisionada](#).

IOPS

As operações de entrada/saída por segundo (IOPS) é uma medida do número de operações de leitura e gravação processadas pelo sistema de arquivos. No modo de uso geral, as IOPS são limitadas a favor da menor latência, o que é benéfico para a maioria dos aplicativos.

Para monitorar e criar alarmes em IOPS no modo de uso geral, use a métrica `PercentIOLimit`. Se essa métrica atingir 100%, a função poderá atingir o tempo limite aguardando a conclusão das operações de leitura e gravação.

Usar o AWS Lambda com o AWS IoT

O AWS IoT fornece comunicação segura entre dispositivos conectados à Internet (como sensores) e a Nuvem AWS. Isso permite que você colete, armazene e analise dados de telemetria de vários dispositivos.

É possível criar regras de AWS IoT para que seus dispositivos interajam com os serviços da AWS. O [Mecanismo de regras](#) do AWS IoT fornece uma linguagem baseada em SQL para selecionar dados de cargas de mensagem e enviar os dados para outros serviços, como o Amazon S3, o Amazon DynamoDB e o AWS Lambda. Defina uma regra para invocar uma função do Lambda quando quiser invocar outro serviço da AWS ou um serviço de terceiros.

Quando uma mensagem de entrada da IoT aciona a regra, o AWS IoT invoca a função do Lambda de forma assíncrona (p. 161) e passa os dados da mensagem da IoT para a função.

O exemplo a seguir mostra uma leitura de umidade do sensor de uma estufa. Os valores de linha e pos identificam a localização do sensor. Esse evento de exemplo é baseado no tipo de estufa nos [Tutoriais de regras da AWS IoT](#).

Example AWS IoTEvent de mensagem do

```
{  
    "row" : "10",  
    "pos" : "23",  
    "moisture" : "75"  
}
```

Para a invocação assíncrona, o Lambda coloca em fila a mensagem e [tenta novamente](#) (p. 176) caso a sua função retorne um erro. Configure sua função com um [destino](#) (p. 164) para manter os eventos que a função não puder processar.

É necessário conceder permissão para que o serviço da AWS IoT invoque sua função do Lambda. Use o comando `add-permission` para adicionar uma instrução de permissão à política baseada em recursos de sua função.

```
aws lambda add-permission --function-name my-function \  
--statement-id iot-events --action "lambda:InvokeFunction" --principal iot.amazonaws.com
```

Você deve ver a saída a seguir:

```
{  
    "Statement": "{\"Sid\":\"iot-events\",\"Effect\":\"Allow\",\"Principal\":\"  
    \\"Service\":[\"iot.amazonaws.com\"],\"Action\":\"lambda:InvokeFunction\",\"Resource\":"  
    \"arn:aws:lambda:us-west-2:123456789012:function:my-function\""}  
}
```

Para obter mais informações sobre como usar o Lambda com o AWS IoT, consulte [Criar uma regra do AWS Lambda](#).

Usar o AWS Lambda com o AWS IoT Events

AWS IoT Events monitora as entradas de vários sensores e aplicativos IoT para reconhecer padrões de eventos. Em seguida, ele toma as ações apropriadas quando ocorrem eventos. O AWS IoT Events recebe as entradas como cargas JSON de muitas fontes. O AWS IoT Events oferece suporte a eventos simples (onde cada entrada aciona um evento) e eventos complexos (onde várias entradas devem ocorrer para acionar o evento).

Para usar o AWS IoT Events, você define um modelo de detector, que é um modelo de máquina de estado do seu equipamento ou processo. Além dos estados, você define entradas e eventos para o modelo. Você também define as ações a serem executadas quando ocorre um evento. Use uma função do Lambda para uma ação quando quiser invocar outro serviço da AWS (como Amazon Connect) ou executar ações em uma aplicação externa (como sua aplicação de planejamento de recursos empresariais, ERP).

Quando o evento ocorre, o AWS IoT Events invoca a função do Lambda de forma assíncrona. Ele fornece informações sobre o modelo do detector e o evento que desencadeou a ação. O exemplo de mensagem de exemplo a seguir é baseado nas definições descritas no [exemplo detalhado simples](#) do AWS IoT Events.

Example AWS IoT Events Event message

```
{  
  "event": ":{  
    "eventName": "myChargedEvent",  
    "eventTime": 1567797571647,  
    "payload":{  
      "detector":{  
        "detectorModelName": "AWS_IoTEvents_Hello_World1567793458261",  
        "detectorModelVersion": "4",  
        "keyValue": "100009"  
      },  
      "eventTriggerDetails":{  
        "triggerType": "Message",  
        "inputName": "AWS_IoTEvents_HelloWorld_VoltageInput",  
        "messageId": "64c75a34-068b-4a1d-ae58-c16215dc4efd"  
      },  
      "actionExecutionId": "49f0f32f-1209-38a7-8a76-d6ca49dd0bc4",  
      "state":{  
        "variables": {},  
        "stateName": "Charged",  
        "timers": {}  
      }  
    }  
  }  
}
```

O evento que é passado para a função do Lambda inclui os seguintes campos:

- **eventName**: o nome desse evento no modelo do detector.
- **eventTime**: a hora em que o evento ocorreu.
- **detector**: o nome e a versão do modelo do detector.
- **eventTriggerDetails**: uma descrição da entrada que acionou o evento.
- **actionExecutionId**: o identificador de execução exclusivo da ação.
- **state**: o estado do modelo do detector quando o evento ocorreu.
 - **stateName**: o nome do estado no modelo do detector.
 - **timers**: quaisquer temporizadores que são definidos neste estado.

- **variables**: quaisquer valores de variável que são definidos neste estado.

É necessário conceder permissão para que o serviço da AWS IoT Events invoque sua função do Lambda. Use o comando `add-permission` para adicionar uma instrução de permissão à política baseada em recursos de sua função.

```
aws lambda add-permission --function-name my-function \
--statement-id iot-events --action "lambda:InvokeFunction" --principal
iotevents.amazonaws.com
```

Você deve ver a saída a seguir:

```
{
  "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\"}"
}
```

Para obter mais informações sobre como usar o Lambda com o AWS IoT Events, consulte [Using AWS IoT Events with other services](#).

Usar o Lambda com o Apache Kafka autogerenciado

O Lambda suporta [Apache Kafka](#) como um [Origem do evento \(p. 170\)](#). O Apache Kafka é uma plataforma de streaming de eventos de código aberto que suporta cargas de trabalho, como pipelines de dados e análises de streaming.

Você pode usar o AWS serviço Kafka gerenciado pela Amazon Managed Streaming for Apache Kafka (Amazon MSK) ou um cluster Kafka autogerenciado. Para obter detalhes sobre o uso do Lambda com o Amazon MSK, consulte [Usar o Lambda com o Amazon MSK \(p. 419\)](#).

Este tópico descreve como usar o Lambda com um cluster Kafka autogerenciado. Dentro da terminologia, um cluster autogerenciado inclui não-AWS hospedou clusters Kafka. Por exemplo, você pode hospedar seu cluster Kafka com um provedor de nuvem, como [CloudKarafka](#). Você também pode usar outras opções de hospedagem da AWS para seu cluster. Para obter mais informações, consulte [Práticas recomendadas para execução do Apache Kafka na AWS](#) no blog sobre Big Data da AWS.

O Apache Kafka como uma fonte de eventos opera de forma semelhante ao uso do Amazon Simple Queue Service (Amazon SQS) ou do Amazon Kinesis. O Lambda pesquisa internamente por novas mensagens da origem do evento e, em seguida, chama de forma síncrona a função do Lambda de destino. O Lambda lê as mensagens em lotes e fornece estas para a sua função como uma carga de eventos. O tamanho máximo do lote é configurável. (O valor padrão é de 100 mensagens.)

Para obter um exemplo de como usar o Kafka autogerenciado como uma fonte de eventos, consulte [Usar o Apache Kafka autohospedado como uma fonte de eventos para o AWS Lambda](#) no AWS Compute blog.

O Lambda envia o lote de mensagens no parâmetro de evento quando ele invoca sua função do Lambda. O payload do evento contém uma matriz de mensagens. Cada item da matriz contém detalhes do tópico do Kafka e do identificador de partição do Kafka, juntamente com um carimbo de data/hora e uma mensagem codificada em base64.

```
{  
  "eventSource": "aws:SelfManagedKafka",  
  "bootstrapServers": "b-2.demo-cluster-1.albcde.c1.kafka.us-east-1.amazonaws.com:9092,b-1.demo-cluster-1.albcde.c1.kafka.us-east-1.amazonaws.com:9092",  
  "records": [  
    "mytopic-0": [  
      {  
        "topic": "mytopic",  
        "partition": "0",  
        "offset": 15,  
        "timestamp": 1545084650987,  
        "timestampType": "CREATE_TIME",  
        "value": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",  
        "headers": [  
          {"headerKey": [  
            104,  
            101,  
            97,  
            100,  
            101,  
            114,  
            86,  
            97,  
            108,  
            117,  
            101
```

```
        ]
      }
    ]
}
```

Tópicos

- [Gerenciar acesso e permissões \(p. 378\)](#)
- [Autenticação Kafka \(p. 379\)](#)
- [Configuração de rede \(p. 380\)](#)
- [Adicionar um cluster do Kafka como uma origem de evento \(p. 380\)](#)
- [Usar um cluster do Kafka como uma fonte de eventos \(p. 382\)](#)
- [Dimensionamento automático da origem do evento Kafka \(p. 383\)](#)
- [Operações de API de origem de eventos \(p. 383\)](#)
- [Erros de mapeamento da fonte de eventos \(p. 383\)](#)
- [Parâmetros de configuração da fonte do \(p. 384\)](#)

Gerenciar acesso e permissões

Para que o Lambda faça uma pesquisa no tópico do Apache Kafka e atualize outros recursos de cluster, sua função do Lambda, bem como seu AWS Identity and Access Management Usuários e funções do (IAM) — devem ter as permissões a seguir.

Permissões de função do Lambda necessárias

Para criar e armazenar logs em um grupo de logs no Amazon CloudWatch Logs, sua função do Lambda deve ter as seguintes permissões em sua [função de execução \(p. 57\)](#):

- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:PutLogEvents`

Permissões de função do Lambda opcionais

Sua função do Lambda pode precisar de permissão para descrever seu segredo do AWS Secrets Manager ou sua [chave gerenciada pelo cliente \(CMK\)](#) do AWS Key Management Service (AWS KMS), ou para acessar sua Virtual Private Cloud (VPC).

Secrets Manager e AWS KMS permissions

Se os seus usuários do Apache Kafka acessarem seus agentes do Kafka pela internet, você deverá especificar um segredo do Secrets Manager. Sua função do Lambda pode precisar de permissão para descrever seu segredo do Secrets Manager ou descriptografar sua CMK gerenciada pelo cliente do AWS KMS. Para acessar esses recursos, a [função de execução \(p. 57\)](#) da função precisa ter as seguintes permissões:

- `secretsmanager:GetSecretValue`
- `kms:Decrypt`

Permissões da VPC

Se somente usuários dentro de uma VPC puderem acessar seu cluster Apache Kafka autogerenciado, sua função do Lambda deverá ter permissão para acessar seus recursos do Amazon Virtual Private Cloud (Amazon VPC). Esses recursos incluem sua VPC, sub-redes, security groups e interfaces de rede. Para acessar esses recursos, a [função de execução \(p. 57\)](#) da função precisa ter as seguintes permissões:

- [ec2:CreateNetworkInterface](#)
- [ec2:DescribeNetworkInterfaces](#)
- [ec2:DescribeVpcs](#)
- [ec2:DeleteNetworkInterface](#)
- [ec2:DescribeSubnets](#)
- [ec2:DescribeSecurityGroups](#)

Adicionar permissões à sua função de execução

Para acessar outros serviços da AWS que seu cluster autogerenciado do Apache Kafka usa, o Lambda utiliza as políticas de permissão que você define na [função de execução \(p. 57\)](#) da função do Lambda.

Por padrão, o Lambda não pode executar as ações obrigatórias ou opcionais para um cluster do Apache Kafka autogerenciado. Você deve criar e definir essas ações em uma [política de confiança do IAM](#) e, em seguida, associar a política à sua função de execução. Este exemplo mostra como criar uma política que permita que o Lambda acesse seus recursos da Amazon VPC.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:CreateNetworkInterface",  
                "ec2:DescribeNetworkInterfaces",  
                "ec2:DescribeVpcs",  
                "ec2:DeleteNetworkInterface",  
                "ec2:DescribeSubnets",  
                "ec2:DescribeSecurityGroups"  
            ],  
            "Resource": "arn:aws:ec2:us-east-1:01234567890:instance/my-instance-name"  
        }  
    ]  
}
```

Para obter informações sobre como criar um documento de política JSON no console do IAM, consulte [Criar políticas na guia JSON](#) no Guia do usuário do IAM.

Adicionar usuários a uma política do IAM

Por padrão, usuários e funções do IAM não têm permissão para executar [operações de API da fonte de eventos \(p. 383\)](#). Para conceder acesso a usuários em sua organização ou conta, talvez seja necessário criar uma política baseada em identidade. Para obter mais informações, consulte [Controlar o acesso aos recursos da AWS usando políticas](#) no Manual do usuário do IAM.

Autenticação Kafka

O Lambda suporta vários métodos para autenticar com seu cluster Apache Kafka autogerenciado. Certifique-se de configurar o cluster Kafka para utilizar um dos seguintes métodos de autenticação suportados pelo Lambda:

- VPC

Se apenas os usuários do Kafka em sua VPC acessarem seus agentes do Kafka, você deverá configurar a fonte do evento com o acesso da VPC.

- SASL/SCRAM

O Lambda é compatível com autenticação SASL/SCRAM (Simple Authentication e Security Layer/Salted Challenge Response Authentication Mechanism) com criptografia TLS. O Lambda envia as credenciais criptografadas para autenticar com o cluster. Como as credenciais são criptografadas, a conexão com o cluster não precisa ser criptografada. Para obter mais informações sobre a autenticação do SASL/SCRAM, consulte [RFC 5802](#).

- SASL/PLAIN

O Lambda é compatível com autenticação SASL/PLAIN com criptografia TLS. Com a autenticação SASL/PLAIN, as credenciais são enviadas como texto não criptografado (não criptografado) para o servidor. Como as credenciais são texto não criptografado, a conexão com o servidor deve usar criptografia TLS.

Para autenticação SASL, você deve armazenar o nome de usuário e senha como um segredo no Secrets Manager. Para obter mais informações, consulte [Tutorial: criar e recuperar um segredo](#) no Manual do usuário do AWS Secrets Manager.

Configuração de rede

Se você configurar o acesso da Amazon VPC aos seus corretores do Kafka, o Lambda deverá ter acesso aos recursos da Amazon VPC.

O Lambda deve ter acesso aos recursos do Amazon Virtual Private Cloud (Amazon VPC) associados ao cluster do Kafka. Recomendamos que você implante AWS PrivateLink [VPC endpoints](#) para o Lambda e AWS Security Token Service(AWS STS). Se a autenticação for necessária, implante também um endpoint da VPC para o Secrets Manager.

Alternativamente, certifique-se de que a VPC associada ao cluster do Kafka inclua um gateway NAT por sub-rede pública. Para obter mais informações, consulte [Acesso aos serviços e à Internet para funções conectadas à VPC \(p. 129\)](#).

Seus grupos de segurança da Amazon VPC devem ser configurados com as seguintes regras (no mínimo):

- Regras de entrada: permitir todo o tráfego em todas as portas do grupo de segurança especificado como a fonte de eventos.
- Regras de saída: permitir todo o tráfego em todas as portas para todos os destinos.

Para obter mais informações sobre a configuração da rede, consulte [Configurar o AWS Lambda com um cluster Apache Kafka em uma VPC](#) no AWS Computar blog.

Adicionar um cluster do Kafka como uma origem de evento

Para criar um [Mapeamento da origem do \(p. 170\)](#), adicione seu cluster Kafka como uma função do Lambda [Trigger do \(p. 18\)](#). Usando o console do Lambda, uma [AWSSDK](#), ou o [AWS Command Line Interface\(AWS CLI\)](#).

Esta seção descreve como criar um mapeamento de origens de eventos do usando o console do Lambda e a AWS CLI.

Prerequisites

- Um cluster Apache Kafka autogerenciado. O Lambda suporta o Apache Kafka versão 0.10.0.0 e posterior.
- Uma função de execução do Lambda com permissão para acessar os recursos da AWS que seu cluster Kafka autogerenciado usa.

Adicionando um cluster Kafka autogerenciado (console)

Siga estas etapas para adicionar seu cluster autogerenciado do Apache Kafka e um tópico do Kafka como um acionador para sua função do Lambda.

Para adicionar um acionador do Apache Kafka à sua função do Lambda (console)

1. Abra a página [Functions](#) (Funções) no console do Lambda.
2. Escolha o nome da função do Lambda.
3. Em Function overview (Visão geral da função), escolha Add trigger (Adicionar gatilho).
4. Em Trigger configuration (Configuração do acionador), faça o seguinte:
 - a. Selecione o Apache Kafka Tipo de gatilho.
 - b. para o Servidores de bootstrap, insira o endereço de host e par de portas de um corretor Kafka em seu cluster e escolha Adicionar. Repita para cada agente da Kafka no cluster.
 - c. para o Nome do tópico, insira o nome do tópico do Kafka usado para armazenar registros no cluster.
 - d. (Opcional) Para Tamanho do lote, insira o número máximo de registros a serem recebidos em um único lote.
 - e. (Opcional) Para Posição inicial, escolha Mais recente para começar a ler o fluxo do registro mais recente. Ou escolha Horizonte de corte para iniciar no registro mais antigo disponível.
5. Sob Método de autenticação, escolha o protocolo de acesso ou autenticação dos corretores Kafka em seu cluster. Se apenas os usuários dentro de sua VPC acessarem seus corretores do Kafka, você deverá configurar o acesso da VPC. Se os usuários acessarem seus agentes do Kafka pela internet, você deverá configurar a autenticação SASL.
 - Para configurar o acesso da VPC, escolha a VPC para o seu cluster Kafka e, em seguida, escolha Sub-redes VPC e Security groups de VPC.
 - Para configurar a autenticação SASL, em Chave secreta, escolha Adicionar, em seguida, faça o seguinte:
 - a. Escolha o tipo de chave. Se o seu corretor Kafka usa texto simples SASL, escolha BASIC_AUTH. Caso contrário, escolha uma das SASL_SCRAM opções.
 - b. Escolha o nome da chave secreta do Secrets Manager que contém as credenciais do cluster do Kafka.
6. Para criar o acionador, selecione Add (Adicionar).

Adicionando um cluster Kafka autogerenciado (AWS CLI)

Use os comandos de exemplo da AWS CLI a seguir para criar e visualizar um acionador autogerenciado do Apache Kafka para sua função do Lambda.

Usar SASL/SCRAM

Se os usuários do Kafka acessarem seus agentes do Kafka pela Internet, você deverá especificar seu segredo do Secrets Manager criado para autenticação SASL/SCRAM. O exemplo a seguir usa

[acreate-event-source-mapping](#) AWS CLI para mapear uma função do Lambda chamada my-kafka-function para um tópico do Kafka chamado AWSKafkaTopic.

```
aws lambda create-event-source-mapping --topics AWSKafkaTopic
  --source-access-configuration
  Type=SASL_SCRAM_512_AUTH,URI=arn:aws:secretsmanager:us-east-1:01234567890:secret:MyBrokerSecretName
  --function-name arn:aws:lambda:us-east-1:01234567890:function:my-kafka-function
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS": ["abc3.xyz.com:9092", "abc2.xyz.com:9092"]}}'
```

Para obter mais informações, consulte a documentação de referência da API.

Usar uma VPC

Se apenas os usuários do Kafka em sua VPC acessarem seus agentes do Kafka, você deverá especificar sua VPC, suas sub-redes e seu grupo de segurança da VPC. O exemplo a seguir usa [acreate-event-source-mapping](#) AWS CLI para mapear uma função do Lambda chamada my-kafka-function para um tópico do Kafka chamado 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-1:01234567890:function:my-kafka-function
  --self-managed-event-source '{"Endpoints":{"KAFKA_BOOTSTRAP_SERVERS": ["abc3.xyz.com:9092", "abc2.xyz.com:9092"]}}'
```

Para obter mais informações, consulte a documentação de referência da API.

Visualizar o status usando a AWS CLI

O exemplo a seguir usa o comando [get-event-source-mapping](#) da AWS CLI para descrever o status do mapeamento de fontes de eventos que você criou.

```
aws lambda get-event-source-mapping
  --uuid dh38738e-992b-343a-1077-3478934hjkfd7
```

Usar um cluster do Kafka como uma fonte de eventos

Quando você adiciona seu cluster Apache Kafka como um gatilho para sua função do Lambda, o cluster é usado como um [Origem do evento \(p. 170\)](#).

O Lambda lê os dados de eventos dos tópicos do Kafka que você especifica em [Topics CreateEventSourceMapping \(p. 786\)](#) com base na posição inicial especificada em [CreateEventSourceMapping \(p. 786\)](#) `StartingPosition`. Após o processamento bem-sucedido, seu tópico do Kafka é confirmado no cluster do Kafka.

Se você especificar `LATEST` como a posição inicial, o Lambda começa a ler a partir da mensagem mais recente em cada partição pertencente ao tópico. Como pode haver algum atraso após a configuração do disparador antes do Lambda começar a ler as mensagens, o Lambda não lê nenhuma mensagem produzida durante esta janela.

O Lambda processa registros de uma ou mais partições de tópico do Kafka que você especifica e envia uma carga JSON para sua função do Lambda. Quando mais registros estiverem disponíveis, o Lambda continuará processando-os em batches, com base no valor especificado em [CreateEventSourceMapping \(p. 786\)](#) >BatchSize, até que a função alcance o tópico.

Se a função do Lambda retornar um erro para qualquer uma das mensagens em um lote, o Lambda tentará novamente todo o lote de mensagens até que o processamento seja bem-sucedido ou as mensagens expiram.

A quantidade máxima de tempo que o Lambda permite que uma função seja executada antes de interrompê-la é de 14 minutos.

Dimensionamento automático da origem do evento Kafka

Quando você cria inicialmente um Apache Kafka[Origem do evento \(p. 170\)](#), o Lambda aloca um consumidor para processar todas as partições no tópico do Kafka. O Lambda aumenta ou diminui automaticamente o número de consumidores, com base na workload. Para preservar a ordenação de mensagens em cada partição, o número máximo de consumidores é um consumidor por partição no tópico.

A cada 15 minutos, o Lambda avalia o atraso de compensação do consumidor de todas as partições no tópico. Se o atraso for muito alto, a partição está recebendo mensagens mais rápido do que o Lambda pode processá-las. Se necessário, o Lambda adiciona ou remove os consumidores do tópico.

Se sua função alvo do Lambda estiver sobrecarregada, o Lambda reduz o número de consumidores. Essa ação reduz a workload na função, reduzindo o número de mensagens que os consumidores podem recuperar e enviar para a função.

Para monitorar a taxa de transferência do tópico do Kafka, você pode visualizar as métricas de consumo do Apache Kafka, como `consumer_lageconsumer_offset`. Para verificar quantas invocações de função ocorrem em paralelo, você também pode monitorar o[Métricas de simultaneidade \(p. 719\)](#) Para a função do.

Operações de API de origem de eventos

Quando você adiciona o cluster do Kafka como uma [fonte de eventos \(p. 170\)](#) para sua função do Lambda usando o console do Lambda, um AWS SDK ou a AWS CLI, o Lambda usa APIs para processar sua solicitação.

Para gerenciar uma origem de evento com a [AWS CLI](#) ou o [AWS SDK](#), você pode usar as seguintes operações da API:

- [CreateEventSourceMapping \(p. 786\)](#)
- [ListEventSourceMappings \(p. 888\)](#)
- [GetEventSourceMapping \(p. 837\)](#)
- [UpdateEventSourceMapping \(p. 956\)](#)
- [DeleteEventSourceMapping \(p. 812\)](#)

Erros de mapeamento da fonte de eventos

Quando você adicionar seu cluster do Apache Kafka como uma [fonte de eventos \(p. 170\)](#) para sua função do Lambda, se ela encontrar um erro, o consumidor do Kafka interrompe o processamento de registros. Os consumidores de uma partição de tópico são aqueles que se inscrevem, leem e processam seus registros.

Seus outros consumidores do Kafka podem continuar processando registros, desde que não encontrem o mesmo erro.

Para determinar a causa de um consumidor parado, verifique o campo `StateTransitionReason` na resposta do `EventSourceMapping`. A lista a seguir descreve os erros de origem do evento que você pode receber:

ESM_CONFIG_NOT_VALID

A configuração do mapeamento da fonte de eventos não é válida.

EVENT_SOURCE_AUTHN_ERROR

O Lambda não conseguiu autenticar a fonte de eventos.

EVENT_SOURCE_AUTHZ_ERROR

O Lambda não tem as permissões necessárias para acessar a fonte de eventos.

FUNCTION_CONFIG_NOT_VALID

A configuração da função do não é válida.

Note

Se os registros de eventos do Lambda excederem o limite de tamanho permitido de 6 MB, eles poderão ficar sem processamento.

Parâmetros de configuração da fonte do

Todos os tipos de origem de evento Lambda compartilham o mesmo[CreateEventSourceMapping \(p. 786\)](#) e [UpdateEventSourceMapping \(p. 956\)](#) Operações da API. No entanto, apenas alguns dos parâmetros se aplicam ao Apache Kafka.

Parâmetros da fonte de eventos que se aplicam ao Apache Kafka autogerenciado

Parâmetro	Obrigatório	Padrão	Observações
BatchSize	N	100	Máximo: 10.000.
Enabled	N	Enabled	
FunctionName	Y		
SelfManageEventSource	Y		Lista de corretores Kafka. Pode definir apenas em Criar
SourceAccessConfiguration	N	Nenhuma credencial	Informações da VPC ou credenciais de autenticação para o cluster Para SASL_PLAIN, defina como BASIC_AUTH
StartingPosition	Y		TRIM_HORIZON Pode definir apenas em Criar

Parâmetro	Obrigatório	Padrão	Observações
Tópicos	Y		Nome do tópico Pode definir apenas em Criar

Usar o AWS Lambda com o Amazon Kinesis Data Firehose

O Amazon Kinesis Data Firehose captura, transforma e carrega dados de transmissão em serviços downstream, como o Kinesis Data Analytics ou o Amazon S3. Você pode escrever funções do Lambda para solicitar processamento personalizado adicional dos dados antes que eles sejam enviados downstream.

Example Evento de mensagem do Amazon Kinesis Data Firehose

```
{
  "invocationId": "invoked123",
  "deliveryStreamArn": "aws:lambda:events",
  "region": "us-west-2",
  "records": [
    {
      "data": "SGVsbG8gV29ybGQ=",
      "recordId": "record1",
      "approximateArrivalTimestamp": 1510772160000,
      "kinesisRecordMetadata": {
        "shardId": "shardId-000000000000",
        "partitionKey": "4d1ad2b9-24f8-4b9d-a088-76e9947c317a",
        "approximateArrivalTimestamp": "2012-04-23T18:25:43.511Z",
        "sequenceNumber": "49546986683135544286507457936321625675700192471156785154",
        "subsequenceNumber": ""
      }
    },
    {
      "data": "SGVsbG8gV29ybGQ=",
      "recordId": "record2",
      "approximateArrivalTimestamp": 151077216000,
      "kinesisRecordMetadata": {
        "shardId": "shardId-000000000001",
        "partitionKey": "4d1ad2b9-24f8-4b9d-a088-76e9947c318a",
        "approximateArrivalTimestamp": "2012-04-23T19:25:43.511Z",
        "sequenceNumber": "49546986683135544286507457936321625675700192471156785155",
        "subsequenceNumber": ""
      }
    }
  ]
}
```

Para obter mais informações, consulte [Transformação de dados do Amazon Kinesis Data Firehose](#) no Guia do desenvolvedor do Kinesis Data Firehose.

Usar o AWS Lambda com o Amazon Kinesis

Você pode usar uma função do AWS Lambda para processar registros em uma [transmissão de dados do Amazon Kinesis](#).

Uma transmissão de dados do Kinesis é um conjunto de [fragmentos](#). Cada fragmento contém uma sequência de registros de dados. Um consumidor é um aplicativo que processa os dados de um stream de dados do Kinesis. É possível mapear uma função do Lambda para um consumidor de taxa de transferência compartilhada (iterador padrão) ou para um consumidor de taxa de transferência dedicada com [distribuição avançada](#).

Para iteradores padrão, o Lambda sonda cada fragmento no stream do Kinesis em busca de registros usando o protocolo HTTP. O mapeamento da origem do evento compartilha a taxa de transferência de leitura com outros consumidores do fragmento.

Para minimizar a latência e maximizar a taxa de transferência de leitura, é possível criar um consumidor de stream de dados com distribuição avançada. Os consumidores de fluxo obtêm uma conexão dedicada para cada estilhaço que não afeta outros aplicativos que fazem leitura do fluxo. A taxa de transferência dedicada pode ajudar se você tiver muitos aplicativos lendo os mesmos dados, ou se você estiver reprocessando um fluxo com grandes registros. O Kinesis envia registros ao Lambda por meio de HTTP/2.

Para obter detalhes sobre transmissões de dados do Kinesis, consulte [Ler dados do Amazon Kinesis Data Streams](#).

O Lambda lê registros do fluxo de dados e invoca sua função [de maneira síncrona \(p. 158\)](#) com um evento que contém registros de transmissão. O Lambda lê registros em lotes e invoca sua função para processar registros do lote. Cada lote contém registros de um único fragmento/fluxo de dados.

Example Evento de registro do Kinesis

```
{  
    "Records": [  
        {  
            "kinesis": {  
                "kinesisSchemaVersion": "1.0",  
                "partitionKey": "1",  
                "sequenceNumber":  
"4959033827149025608559692538361571095921575989136588898",  
                "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":  
"4959033827149025608559692540925702759324208523137515618",  
                "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"
    }
]
```

Por padrão, o Lambda invoca sua função assim que os registros estão disponíveis na transmissão. Se o lote que o Lambda lê da transmissão tiver apenas um registro nele, o Lambda envia apenas um registro para a função. Para evitar invocar a função com um número pequeno de registros, você pode instruir à origem dos eventos para fazer o buffer dos registros por até cinco minutos, configurando uma janela de lote. Antes de invocar a função, o Lambda continua a ler registros da transmissão até coletar um lote inteiro, ou até que a janela de lote expire.

Se a sua função retornar um erro, o Lambda tentará executar novamente o lote até que o processamento seja bem-sucedido ou os dados expirem. Para evitar fragmentos paralisados, você pode configurar o mapeamento de fontes de eventos para tentar novamente com um tamanho de lote menor, limitar o número de tentativas ou descartar registros que são muito antigos. Para manter eventos descartados, você pode configurar o mapeamento de fontes de eventos para enviar detalhes sobre lotes com falha para uma fila do SQS ou para um tópico do SNS.

Você também pode aumentar a simultaneidade processando vários lotes de cada fragmento em paralelo. O Lambda pode processar até 10 lotes em cada fragmento simultaneamente. Se você aumentar o número de lotes simultâneos por fragmento, o Lambda ainda garante o processamento em ordem no nível de chave de partição.

Configure `ParallelizationFactor` para processar um fragmento de um fluxo de dados do Kinesis ou do DynamoDB com mais de uma invocação do Lambda simultaneamente. Você pode especificar o número de lotes simultâneos que o Lambda pesquisa de um fragmento por meio de um fator de paralelização de 1 (padrão) a 10. Por exemplo, quando `ParallelizationFactor` estiver definido como 2, você poderá ter 200 invocações simultâneas do Lambda no máximo para processar 100 fragmentos de dados do Kinesis. Isso ajuda a aumentar a taxa de transferência de processamento quando o volume de dados é volátil e o valor de `IteratorAge` é alto. Observe que o fator de paralelização não funcionará se você estiver usando a agregação do Kinesis. Para obter mais informações, consulte [New AWS Lambda scaling controls for Kinesis and DynamoDB event sources](#).

Seções

- [Configurar o stream de dados e a função \(p. 389\)](#)
- [Permissões da função de execução \(p. 389\)](#)
- [Configurar um stream como fonte de eventos \(p. 390\)](#)
- [API do mapeamento da fonte de eventos \(p. 391\)](#)
- [Tratamento de erros \(p. 393\)](#)
- [Métricas do Amazon CloudWatch \(p. 394\)](#)
- [Janelas de tempo \(p. 394\)](#)
- [Gerar relatórios de falhas de itens de lote \(p. 397\)](#)
- [Tutorial: Usar o AWS Lambda com o Amazon Kinesis \(p. 399\)](#)
- [Código de exemplo da função do \(p. 403\)](#)
- [Modelo do AWS SAM para uma aplicação do Kinesis \(p. 406\)](#)

Configurar o stream de dados e a função

Sua função do Lambda é uma aplicação de consumidor para seu fluxo de dados. Ele processa um lote de registros por vez de cada estilhaço. Você pode mapear uma função do Lambda para um fluxo de dados (iterador padrão) ou para um consumidor de um fluxo ([distribuição avançada](#)).

Para os iteradores padrão, o Lambda sonda cada fragmento em seu fluxo do Kinesis para identificar registros a uma taxa básica de uma vez por segundo. Quando houver mais registros disponíveis, o Lambda mantém lotes de processamento até que a função alcance o fluxo. O mapeamento da origem do evento compartilha a taxa de transferência de leitura com outros consumidores do fragmento.

Para minimizar a latência e maximizar a taxa de transferência de leitura, crie um consumidor de stream de dados com distribuição avançada. Os consumidores da distribuição avançada obtêm uma conexão dedicada para cada fragmento que não afeta outros aplicativos que fazem leitura do stream. Os consumidores de fluxos usam HTTP/2 para reduzir a latência, enviando os registros para o Lambda por meio de uma conexão de longa duração e compactando cabeçalhos de solicitação. Você pode criar um consumidor de fluxo com a API [RegisterStreamConsumer](#) do Kinesis.

```
aws kinesis register-stream-consumer --consumer-name con1 \
--stream-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream
```

Você deve ver a saída a seguir:

```
{
  "Consumer": {
    "ConsumerName": "con1",
    "ConsumerARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream/
consumer/con1:1540591608",
    "ConsumerStatus": "CREATING",
    "ConsumerCreationTimestamp": 1540591608.0
  }
}
```

Para aumentar a velocidade com que sua função processa registros, adicione fragmentos ao fluxo de dados. O Lambda processa registros em cada fragmento sequencialmente. Ele deixará de processar registros adicionais em um estilhaço se sua função retornar um erro. Com mais estilhaços, há mais lotes sendo processados de uma só vez, o que diminui o impacto de erros na simultaneidade.

Se a função não conseguir se expandir para lidar com o número total de lotes simultâneos, [solicite um aumento de cota](#) (p. 53) ou [reserve simultaneidade](#) (p. 113) para a função.

Permissões da função de execução

O Lambda precisa das permissões a seguir para gerenciar recursos relacionados ao seu fluxo de dados do Kinesis. Adicione-as à função de execução (p. 57) da sua função.

- [kinesis:DescribeStream](#)
- [kinesis:DescribeStreamSummary](#)
- [kinesis:GetRecords](#)
- [kinesis:GetShardIterator](#)
- [kinesis>ListShards](#)
- [kinesis>ListStreams](#)
- [kinesis:SubscribeToShard](#)

A política gerenciada `AWSLambdaKinesisExecutionRole` inclui essas permissões. Para obter mais informações, consulte [AWS LambdaFunção de execução do \(p. 57\)](#).

Para enviar registros de lotes com falha para uma fila ou um tópico, sua função precisa de permissões adicionais. Cada serviço de destino requer uma permissão diferente, como se segue:

- Amazon SQS – [sq:SendMessage](#)
- Amazon SNS – [sns:Publish](#)

Configurar um stream como fonte de eventos

Crie um mapeamento de fontes de eventos para orientar o Lambda a enviar registros de seu fluxo de dados para uma função do Lambda. É possível criar vários mapeamentos de origem de eventos para processar os mesmos dados com várias funções do Lambda ou processar itens de vários fluxos de dados com uma única função. Ao processar itens de vários fluxos de dados, cada lote conterá somente registros de um único fragmento/transmissão.

Para configurar sua função para leitura a partir do Kinesis no console do Lambda, crie um acionador do Kinesis.

Para criar um trigger

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Em Function overview (Visão geral da função), escolha Add trigger (Adicionar gatilho).
4. Escolha um tipo de acionador.
5. Configure as opções necessárias e escolha Add (Adicionar).

O Lambda oferece suporte às seguintes opções das fontes de eventos do Kinesis.

Opções de fonte do evento

- Kinesis stream (Fluxo do Kinesis): a transmissão do Kinesis a partir da qual a leitura de registros é realizada.
- Consumer (Consumidor) (opcional): use um consumidor de transmissão para ler a partir da transmissão por uma conexão dedicada.
- Batch size (Tamanho do lote): o número de registros a serem enviados para a função em cada lote, até 10.000. O Lambda transmite todos os registros no batch para a função em uma única chamada, enquanto o tamanho total dos eventos não exceder o [limite de carga útil \(p. 53\)](#) para invocação síncrona (6 MB).
- Batch window (Janela de lote): especifique o máximo de tempo para reunir registros antes de invocar a função, em segundos.
- Starting position (Posição inicial): processe apenas novos registros, todos os registros existentes ou registros criados após uma determinada data.
 - Latest (Mais recente): processe novos registros adicionados ao fluxo.
 - Trim horizon (Redução horizontal): processe todos os registros na transmissão.
 - At timestamp (Na data e hora): processe registros a partir de uma hora específica.

Depois de processar todos os registros existentes, a função é capturada e continua a processar novos registros.

- Destino na falha: uma fila do SQS ou um tópico do SNS para registros que não podem ser processados. Quando o Lambda descarta um lote de registros porque ele é muito antigo ou esgotou todas as tentativas, ele envia detalhes sobre o lote para a fila ou tópico.

- Retry attempts (Tentativas de repetição): o número máximo de vezes que o Lambda tenta novamente quando a função retorna um erro. Isso não se aplica a erros de serviço ou controles em que o lote não atingiu a função.
- Idade máxima do registro: a idade máxima de um registro que o Lambda envia para sua função.
- Dividir lote por erro: quando a função retornar um erro, o lote é dividido em dois antes de uma nova tentativa.
- Lotes simultâneos por fragmento: processa vários lotes do mesmo fragmento simultaneamente.
- Enabled (Habilitado): defina como verdadeiro para habilitar o mapeamento de fontes de eventos. Defina como falso para interromper o processamento de registros. O Lambda monitora o último registro processado e retoma o processamento a partir desse ponto quando habilitado novamente.

Note

O Kinesis cobra por cada fragmento e, para distribuição avançada, dados lidos da transmissão. Para obter detalhes de preço, consulte [Preço do Amazon Kinesis](#).

Para gerenciar a configuração da fonte do evento posteriormente, escolha o gatilho no designer.

API do mapeamento da fonte de eventos

Para gerenciar uma origem de evento com a [AWS CLI](#) ou o [AWS SDK](#), você pode usar as seguintes operações da API:

- [CreateEventSourceMapping \(p. 786\)](#)
- [ListEventSourceMappings \(p. 888\)](#)
- [GetEventSourceMapping \(p. 837\)](#)
- [UpdateEventSourceMapping \(p. 956\)](#)
- [DeleteEventSourceMapping \(p. 812\)](#)

Para criar o mapeamento de origem de eventos com a AWS CLI, use o comando `create-event-source-mapping`. O exemplo a seguir usa a AWS CLI para mapear uma função chamada `my-function` para um fluxo de dados do Kinesis. O streaming de dados é especificado por um nome de recurso da Amazon (ARN), com um tamanho de lote de 500, a partir do time stamp no horário do Unix.

```
aws lambda create-event-source-mapping --function-name my-function \
--batch-size 500 --starting-position AT_TIMESTAMP --starting-position-timestamp 1541139109
 \
--event-source-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream
```

Você deve ver a saída a seguir:

```
{
  "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
  "BatchSize": 500,
  "MaximumBatchingWindowInSeconds": 0,
  "ParallelizationFactor": 1,
  "EventSourceArn": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
  "LastModified": 1541139209.351,
  "LastProcessingResult": "No records processed",
  "State": "Creating",
  "StateTransitionReason": "User action",
```

```
"DestinationConfig": {},  
"MaximumRecordAgeInSeconds": 604800,  
"BisectBatchOnFunctionError": false,  
"MaximumRetryAttempts": 10000  
}
```

Para usar um consumidor, especifique o ARN do consumidor em vez do ARN do fluxo.

Configure opções adicionais para personalizar como os lotes são processados e especificar quando descartar registros que não podem ser processados. O exemplo a seguir atualiza um mapeamento de fontes de eventos para enviar um registro de falha para uma fila do SQS após duas tentativas de repetição, ou se os registros tiverem mais de uma hora.

```
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:sqs:us-  
east-2:123456789012:dlq"}}'
```

Você deve ver este resultado:

```
{  
    "UUID": "f89f8514-cdd9-4602-9e1f-01a5b77d449b",  
    "BatchSize": 100,  
    "MaximumBatchingWindowInSeconds": 0,  
    "ParallelizationFactor": 1,  
    "EventSourceArn": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream",  
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",  
    "LastModified": 1573243620.0,  
    "LastProcessingResult": "PROBLEM: Function call failed",  
    "State": "Updating",  
    "StateTransitionReason": "User action",  
    "DestinationConfig": {},  
    "MaximumRecordAgeInSeconds": 604800,  
    "BisectBatchOnFunctionError": false,  
    "MaximumRetryAttempts": 10000  
}
```

As configurações atualizadas são aplicadas de forma assíncrona e não são refletidas na saída até que o processo seja concluído. Use o comando `get-event-source-mapping` para visualizar o status atual.

```
aws lambda get-event-source-mapping --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b
```

Você deve ver este resultado:

```
{  
    "UUID": "f89f8514-cdd9-4602-9e1f-01a5b77d449b",  
    "BatchSize": 100,  
    "MaximumBatchingWindowInSeconds": 0,  
    "ParallelizationFactor": 1,  
    "EventSourceArn": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream",  
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",  
    "LastModified": 1573244760.0,  
    "LastProcessingResult": "PROBLEM: Function call failed",  
    "State": "Enabled",  
    "StateTransitionReason": "User action",  
    "DestinationConfig": {  
        "OnFailure": {  
            "Destination": "arn:aws:sqs:us-east-2:123456789012:dlq"  
        }  
    }  
}
```

```
},
"MaximumRecordAgeInSeconds": 3600,
"BisectBatchOnFunctionError": false,
"MaximumRetryAttempts": 2
}
```

Para processar vários lotes simultaneamente, use a opção `--parallelization-factor`.

```
aws lambda update-event-source-mapping --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b11284 \
--parallelization-factor 5
```

Tratamento de erros

O mapeamento de fontes de eventos que lê registros da transmissão do Kinesis chama sua função de forma síncrona e tenta novamente em caso de erros. Se a função for limitada ou o serviço Lambda retornar um erro sem invocar a função, o Lambda tentará novamente até que os registros expirem ou excedam a idade máxima configurada no mapeamento de fontes de eventos.

Se a função receber os registros, mas retornar um erro, o Lambda tentará novamente até que os registros no lote expirem, excedam a idade máxima ou atinjam a cota de repetição configurada. Para erros de função, você também pode configurar o mapeamento de fontes de eventos para dividir um lote com falha em dois lotes. Tentar novamente com lotes menores isola registros inválidos e resolve problemas de tempo limite. A divisão de um lote não é considerada para a cota de novas tentativas.

Se as medidas de tratamento de erros falharem, o Lambda descartará os registros e continuará processando lotes provenientes da transmissão. Com as configurações padrão, isso significa que um registro ruim pode bloquear o processamento no fragmento afetado por até uma semana. Para evitar isso, configure o mapeamento de fontes de eventos da sua função com um número razoável de tentativas e uma idade máxima de registro que se adapte ao seu caso de uso.

Para reter um registro dos eventos descartados, configure um destino para eventos com falha. O Lambda envia um documento para a fila de destino ou o tópico com detalhes sobre o lote.

Como configurar um destino para registros de eventos com falha

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Em Function overview (Visão geral da função), escolha Add destination (Adicionar destino).
4. Em Source (Origem), escolha Stream invocation (Chamada de fluxo).
5. Para Stream (Fluxo), escolha um fluxo mapeado para a função.
6. Em Destination type (Tipo de destino), escolha o tipo de recurso que recebe o registro da invocação.
7. Em Destination (Destino), escolha um recurso.
8. Escolha Save (Salvar).

O exemplo a seguir mostra um registro de invocação de uma transmissão do Kinesis.

Example Registro de invocação

```
{
  "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"
    }
}
```

Você pode usar essas informações para recuperar os registros afetados da transmissão para solução de problemas. Os registros reais não estão incluídos, portanto, você deve processar esses registros e recuperá-los da transmissão antes que eles expirem e sejam perdidos.

Métricas do Amazon CloudWatch

O Lambda emite a métrica `IteratorAge` quando a sua função termina de processar um lote de registros. A métrica indica a idade do último registro no lote quando o processamento foi concluído. Se a sua função estiver processando novos eventos, você poderá usar a idade do iterador para estimar a latência entre quando um registro é adicionado e quando a função o processa.

Uma tendência crescente na idade do iterador pode indicar problemas com sua função. Para obter mais informações, consulte [Trabalhar com métricas de função do AWS Lambda \(p. 717\)](#).

Janelas de tempo

As funções do Lambda podem executar aplicações de processamento contínuo de transmissões. Um stream representa dados não vinculados que fluem continuamente por meio de sua aplicação. Para analisar as informações dessa entrada de atualização contínua, você pode vincular os registros incluídos usando uma janela definida em termos de tempo.

As janelas de tumbling são janelas de tempo distintas que abrem e fecham em intervalos regulares. Por padrão, as invocações do Lambda são sem estado. Não é possível usá-las para processar dados ao longo de várias invocações contínuas sem um banco de dados externo. No entanto, com as janelas de tumbling, você pode manter seu estado em todas as invocações. Esse estado contém o resultado agregado das mensagens previamente processadas para a janela atual. Seu estado pode ter no máximo 1 MB por fragmento. Se exceder esse tamanho, o Lambda encerra a janela antes.

Cada registro de um stream pertence a uma janela específica. Um registro é processado apenas uma vez, quando o Lambda processa a janela à qual o registro pertence. Em cada janela, você pode executar cálculos, como uma soma ou média, no nível da [chave de partição](#) dentro de um fragmento.

Agregação e processamento

Sua função gerenciada pelo usuário é chamada tanto para agregação quanto para processamento dos resultados finais dessa agregação. O Lambda agrupa todos os registros recebidos na janela. Você pode receber esses registros em vários lotes, cada um como uma invocação separada. Cada invocação recebe

um estado. Assim, ao usar janelas de tumbling, sua resposta de função do Lambda deve conter uma propriedade de state. Se a resposta não contiver uma propriedade de state, o Lambda considerará esta uma invocação com falha. Para satisfazer essa condição, a função pode retornar um objeto do TimeWindowEventResponse, que tem a seguinte forma JSON:

Example TimeWindowEventResponse Valores de

```
{  
    "state": {  
        "1": 282,  
        "2": 715  
    },  
    "batchItemFailures": []  
}
```

Note

Para funções Java, recomendamos o uso de um Map<String, String> para representar o estado.

No final da janela, a sinalização isFinalInvokeForWindow é definida como true para indicar que esse é o estado final e que está pronto para processamento. Após o processamento, a janela é concluída e sua invocação final é concluída e, em seguida, o estado é descartado.

No final da janela, o Lambda usa o processamento final para ações sobre os resultados da agregação. Seu processamento final é invocado de forma síncrona. Após a invocação bem-sucedida, sua função define os pontos de verificação no número da sequência e o processamento de streams continua. Se a invocação não for bem-sucedida, sua função do Lambda suspenderá o processamento adicional até uma chamada bem-sucedida.

Example KinesisTimeWindowEvent

```
{  
    "Records": [  
        {  
            "kinesis": {  
                "kinesisSchemaVersion": "1.0",  
                "partitionKey": "1",  
                "sequenceNumber":  
"49590338271490256608559692538361571095921575989136588898",  
                "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",  
                "approximateArrivalTimestamp": 1607497475.000  
            },  
            "eventSource": "aws:kinesis",  
            "eventVersion": "1.0",  
            "eventID":  
"shardId-000000000006:49590338271490256608559692538361571095921575989136588898",  
            "eventName": "aws:kinesis:record",  
            "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-kinesis-role",  
            "awsRegion": "us-east-1",  
            "eventSourceARN": "arn:aws:kinesis:us-east-1:123456789012:stream/lambda-stream"  
        }  
    ],  
    "window": {  
        "start": "2020-12-09T07:04:00Z",  
        "end": "2020-12-09T07:06:00Z"  
    },  
    "state": {
```

```
        "1": 282,
        "2": 715
    },
    "shardId": "shardId-000000000006",
    "eventSourceARN": "arn:aws:kinesis:us-east-1:123456789012:stream/lambda-stream",
    "isFinalInvokeForWindow": false,
    "isWindowTerminatedEarly": false
}
```

Configuração

Você pode configurar janelas em cascata ao criar ou atualizar um [mapeamento de fonte de eventos \(p. 170\)](#). Para configurar uma janela em cascata, especifique a janela em segundos. O comando de exemplo da AWS Command Line Interface (AWS CLI) a seguir cria um mapeamento de fonte de eventos em streaming com uma janela em cascata de 120 segundos. A função do Lambda definida para agregação e processamento é chamada de `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 "arn:aws:lambda:us-east-1:123456789018:function:tumbling-window-example-function" --region us-east-1 --starting-position TRIM_HORIZON --tumbling-window-in-seconds 120
```

O Lambda determina os limites da janela em cascata com base no horário em que os registros foram inseridos no stream. Todos os registros têm um carimbo de data/hora aproximado disponível que o Lambda usa para determinar os limites.

As agregações de janelas em cascata não são compatíveis com refragmentação. Quando o fragmento termina, o Lambda considera a janela como fechada e os fragmentos filhos iniciam suas próprias janelas em um novo estado.

As janelas em cascata são totalmente compatíveis com as políticas `maxRetryAttempts` e `maxRecordAge`.

Example Handler.py: agregação e processamento

A função do Python a seguir demonstra como agregar e, em seguida, processar seu estado final:

```
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}
```

Gerar relatórios de falhas de itens de lote

Ao consumir e processar dados de transmissão de uma fonte de eventos, o Lambda definirá checkpoints por padrão no número mais elevado na sequência de um lote somente quando o lote for um sucesso total. O Lambda trata todos os outros resultados como uma falha completa e tenta processar novamente o lote até o limite de novas tentativas. Para permitir sucessos parciais durante o processamento de lotes de um stream, ative `ReportBatchItemFailures`. Permitir sucessos parciais pode ajudar a reduzir o número de novas tentativas em um registro, embora não impeça totalmente a possibilidade de novas tentativas em um registro bem-sucedido.

Para ativar `ReportBatchItemFailures`, inclua o valor de enum `ReportBatchItemFailures` na lista `FunctionResponseTypes`. Essa lista indica quais tipos de resposta estão habilitados para sua função. Você pode configurar essa lista ao criar ou atualizar um [mapeamento de fonte de eventos \(p. 170\)](#).

Sintaxe do relatório

Ao configurar relatórios sobre falhas de itens de lote, a classe `StreamsEventResponse` é retornada com uma lista de falhas de itens de lote. É possível usar um objeto `StreamsEventResponse` para retornar o número sequencial do primeiro registro com falha no lote. Você também pode criar sua própria classe personalizada usando a sintaxe de resposta correta. A seguinte estrutura JSON mostra a sintaxe de resposta necessária:

```
{  
  "batchItemFailures": [  
    {  
      "itemIdentifier": "<id>"  
    }  
  ]  
}
```

Condições de sucesso e falha

O Lambda trata um lote como um sucesso completo se você retornar qualquer um destes:

- Uma lista de `batchItemFailure` vazia
- Uma lista de `batchItemFailure` nula
- Uma vazi `EventResponse`
- Uma nul `EventResponse`

O Lambda trata um lote como uma falha absoluta se você retornar qualquer um dos seguintes:

- Uma string vazi `itemIdentifier`
- Uma nul `itemIdentifier`
- Um `itemIdentifier` com um nome de chave inválido

O Lambda faz novas tentativas após falhas com base na sua estratégia de repetição.

Dividir um lote

Se a invocação falhar e `BisectBatchOnFunctionError` estiver ativado, o lote será dividido independentemente da configuração de `ReportBatchItemFailures`.

Quando uma resposta de sucesso parcial do lote é recebida e tanto `BisectBatchOnFunctionError` quanto `ReportBatchItemFailures` estão ativados, o lote é dividido no número de sequência retornado e o Lambda tenta novamente apenas os registros restantes.

Java

Example Handler.java: retornar um novo StreamsEventResponse()

```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

public class ProcessKinesisRecords implements RequestHandler<KinesisEvent,
    Serializable> {

    @Override
    public Serializable 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) {
                //Return failed record's sequence number
                batchItemFailures.add(new
                StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

        return new StreamsEventResponse(batchItemFailures);
    }
}
```

Python

Example Handler.py: retornar batchItemFailures[]

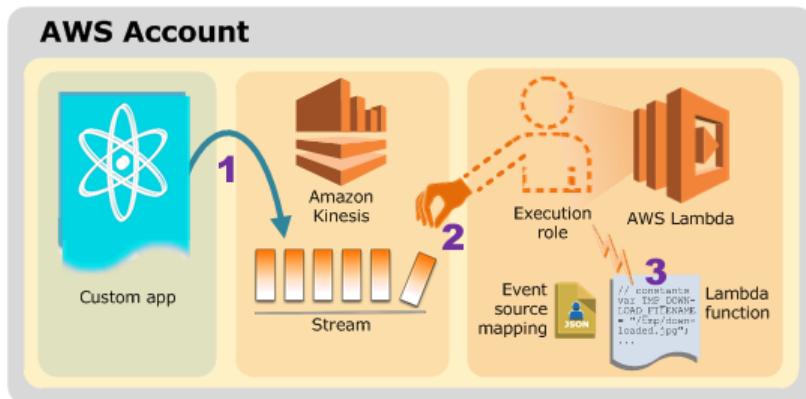
```
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    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": []}
```

Tutorial: Usar o AWS Lambda com o Amazon Kinesis

Neste tutorial, você cria uma função do Lambda para consumir eventos de uma transmissão do Kinesis. O diagrama a seguir ilustra o fluxo do aplicativo:



1. O aplicativo personalizado grava registros no fluxo.
2. O AWS Lambda sonda a transmissão e, quando detecta novos registros, invoca sua função do Lambda.
3. O AWS Lambda executa a função do Lambda assumindo a função de execução especificada no momento da criação da função do Lambda.

Prerequisites

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em [Conceitos básicos do Lambda \(p. 9\)](#) para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, você precisa de um terminal de linha de comando ou de um shell para executar os comandos. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

Você deve ver a saída a seguir:

```
aws-cli/2.0.57 Python/3.7.4 Darwin/19.6.0 exe/x86_64
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

Criar a função de execução

Crie a [função de execução \(p. 57\)](#) que dá à sua função permissão para acessar recursos do AWS.

Para criar uma função de execução

1. Abra a [página Roles \(Funções\)](#) no console do IAM.
2. Selecione Create role.
3. Crie uma função com as seguintes propriedades.

- Trusted entity (Entidade confiável – AWS Lambda).
- Permissões: AWSLambdaKinesisExecutionRole.
- Nome da função – **lambda-kinesis-role**.

A política AWSLambdaKinesisExecutionRole tem as permissões necessárias para a função ler itens do Kinesis e gravar logs no CloudWatch Logs.

Criar a função

O código de amostra a seguir recebe uma entrada de evento do Kinesis e processa as mensagens que ela contém. Na ilustração, o código grava alguns dos dados de entrada no CloudWatch Logs.

Note

Para o código de amostra em outras linguagens, consulte [Código de exemplo da função do \(p. 403\)](#).

Example index.js

```
console.log('Loading function');

exports.handler = function(event, context) {
    //console.log(JSON.Stringify(event, null, 2));
    event.Records.forEach(function(record) {
        // Kinesis data is base64 encoded so decode here
        var payload = Buffer.from(record.kinesis.data, 'base64').toString('ascii');
        console.log('Decoded payload:', payload);
    });
};
```

Para criar a função

1. Copie o código de amostra em um arquivo chamado `index.js`.
2. Crie um pacote de implantação.

```
zip function.zip index.js
```

3. Crie uma função do Lambda com o comando `create-function`.

```
aws lambda create-function --function-name ProcessKinesisRecords \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs12.x \
--role arn:aws:iam::123456789012:role/lambda-kinesis-role
```

Testar a função do Lambda

Invoque sua função do Lambda manualmente usando o comando `invoke` da CLI do AWS Lambda e um evento do Kinesis de amostra.

Para testar a função do Lambda

1. Copie o JSON a seguir em um arquivo e salve-o como `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::123456789012:role/lambda-kinesis-role",  
    "awsRegion": "us-east-2",  
    "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-  
stream"  
},  
}  
}
```

2. Use o comando `invoke` para enviar o evento para a função.

```
aws lambda invoke --function-name ProcessKinesisRecords --payload file://input.txt  
out.txt
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

A resposta é salva no `out.txt`.

Criar uma transmissão do Kinesis

Use o comando `create-stream` para criar um fluxo.

```
aws kinesis create-stream --stream-name lambda-stream --shard-count 1
```

Execute o comando `describe-stream` a seguir para obter o ARN do stream.

```
aws kinesis describe-stream --stream-name lambda-stream
```

Você deve ver a saída a seguir:

```
{  
    "StreamDescription": {  
        "Shards": [  
            {  
                "ShardId": "shardId-000000000000",  
                "HashKeyRange": {  
                    "StartingHashKey": "0",  
                    "EndingHashKey": "340282366920746074317682119384634633455"  
                },  
                "SequenceNumberRange": {  
                    "StartingSequenceNumber":  
                    "49591073947768692513481539594623130411957558361251844610"  
                }  
            }  
        ],  
    }  
}
```

```
"StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/lambda-stream",
"StreamName": "lambda-stream",
"StreamStatus": "ACTIVE",
"RetentionPeriodHours": 24,
"EnhancedMonitoring": [
    {
        "ShardLevelMetrics": []
    }
],
"EncryptionType": "NONE",
"KeyId": null,
"StreamCreationTimestamp": 1544828156.0
}
```

Você usa o ARN do stream na próxima etapa para associar o stream à sua função do Lambda.

Adicionar uma fonte de eventos no AWS Lambda

Execute o seguinte comando AWS CLI da add-event-source.

```
aws lambda create-event-source-mapping --function-name ProcessKinesisRecords \
--event-source arn:aws:kinesis:us-west-2:123456789012:stream/lambda-stream \
--batch-size 100 --starting-position LATEST
```

Observe o ID do mapeamento para uso posterior. Você pode obter uma lista de mapeamentos de origem de evento executando o comando list-event-source-mappings a seguir.

```
aws lambda list-event-source-mappings --function-name ProcessKinesisRecords \
--event-source arn:aws:kinesis:us-west-2:123456789012:stream/lambda-stream
```

Na resposta, pode verificar que o valor do status é enabled. Os mapeamentos de origem de eventos podem ser desabilitados para pausar a pesquisa temporariamente, sem perder nenhum registro.

Testar a configuração

Para testar o mapeamento da origem do evento, adicione registros de eventos ao seu stream do Kinesis. O valor --data é uma string que a CLI codifica em base64 antes de enviá-la ao Kinesis. Você pode executar o mesmo comando mais de uma vez para adicionar vários registros ao fluxo.

```
aws kinesis put-record --stream-name lambda-stream --partition-key 1 \
--data "Hello, this is a test."
```

O Lambda usa a função de execução para ler registros da transmissão. Em seguida, ele invoca sua função do Lambda, transmitindo lotes de registros. A função decodifica os dados de cada registro e os registra, enviando a saída para CloudWatch Logs. Visualize os logs no [console do CloudWatch](#).

Limpar recursos da

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua conta da AWS.

Para excluir a função de execução

1. Abra a página [Roles](#) (Funções) no console do IAM.

2. Selecione a função de execução que você criou.
3. Selecione Delete role (Excluir função).
4. Escolha Sim, excluir.

Como excluir a função do Lambda

1. Abra a página [Functions](#) (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Actions, Delete.
4. Escolha Delete.

Para excluir a transmissão do Kinesis

1. Faça login no AWS Management Console e abra o console do Kinesis em <https://console.aws.amazon.com/kinesis>.
2. Selecione a transmissão criada.
3. Escolha Actions, Delete.
4. Digite **delete** na caixa de texto.
5. Escolha Delete.

Código de exemplo da função do

Para processar eventos do Amazon Kinesis, percorra os registros incluídos no objeto de evento e decodifique os dados codificados em base64 incluídos em cada um.

Note

O código nesta página não oferece suporte a [registros agregados](#). Você pode desativar a agregação na [configuração](#) da Biblioteca do Produtor ou usar a [biblioteca de agregação de registros do Kinesis](#) para desagregar registros.

O código de amostra está disponível para as seguintes linguagens.

Tópicos

- [Node.js 12.x \(p. 403\)](#)
- [Java 11 \(p. 404\)](#)
- [C# \(p. 404\)](#)
- [Python 3 \(p. 405\)](#)
- [Go \(p. 406\)](#)

Node.js 12.x

O código de amostra a seguir recebe uma entrada de evento do Kinesis e processa as mensagens que ela contém. Na ilustração, o código grava alguns dos dados de entrada no CloudWatch Logs.

Example index.js

```
console.log('Loading function');

exports.handler = function(event, context) {
    //console.log(JSON.stringify(event, null, 2));
```

```
event.Records.forEach(function(record) {
    // Kinesis data is base64 encoded so decode here
    var payload = Buffer.from(record.kinesis.data, 'base64').toString('ascii');
    console.log('Decoded payload:', payload);
});
```

Compaca o código do exemplo para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Node.js com arquivos .zip \(p. 517\)](#).

Java 11

Veja a seguir um exemplo de código Java que recebe dados de registro de evento do Kinesis como uma entrada e a processa. Na ilustração, o código grava alguns dos dados de eventos de entrada no CloudWatch Logs.

No código, `recordHandler` é o handler. O handler usa a classe predefinida `KinesisEvent` definida na biblioteca `aws-lambda-java-events`.

Example ProcessKinesisEvents.java

```
package example;

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.KinesisEventRecord;

public class ProcessKinesisRecords implements RequestHandler<KinesisEvent, Void>{
    @Override
    public Void handleRequest(KinesisEvent event, Context context)
    {
        for(KinesisEventRecord rec : event.getRecords()) {
            System.out.println(new String(rec.getKinesis().getData().array()));
        }
        return null;
    }
}
```

Se o manipulador obtém um retorno normal sem exceções, o Lambda considera que os lotes de entrada de registros foram processados com êxito e começa a ler novos registros no fluxo. Se o manipulador gera uma exceção, o Lambda considera que os lotes de entrada de registros não foram processados e invoca novamente a função com o mesmo lote de registros.

Dependencies

- `aws-lambda-java-core`
- `aws-lambda-java-events`
- `aws-java-sdk`

Crie o código com as dependências da biblioteca do Lambda para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Java com arquivos .zip ou JAR \(p. 599\)](#).

C#

Veja a seguir um exemplo de código C# que recebe dados de registro de evento do Kinesis como uma entrada e a processa. Na ilustração, o código grava alguns dos dados de eventos de entrada no CloudWatch Logs.

No código, `HandleKinesisRecord` é o handler. O handler usa a classe predefinida `KinesisEvent` definida na biblioteca `Amazon.Lambda.KinesisEvents`.

Example ProcessingKinesisEvents.cs

```
using System;
using System.IO;
using System.Text;

using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;

namespace KinesisStreams
{
    public class KinesisSample
    {
        [LambdaSerializer(typeof(JsonSerializer))]
        public void HandleKinesisRecord(KinesisEvent kinesisEvent)
        {
            Console.WriteLine($"Beginning to process {kinesisEvent.Records.Count} records...");

            foreach (var record in kinesisEvent.Records)
            {
                Console.WriteLine($"Event ID: {record.EventId}");
                Console.WriteLine($"Event Name: {record.EventName}");

                string recordData = GetRecordContents(record.Kinesis);
                Console.WriteLine($"Record Data:");
                Console.WriteLine(recordData);
            }
            Console.WriteLine("Stream processing complete.");
        }

        private string GetRecordContents(KinesisEvent.Record streamRecord)
        {
            using (var reader = new StreamReader(streamRecord.Data, Encoding.ASCII))
            {
                return reader.ReadToEnd();
            }
        }
    }
}
```

Substitua o `Program.cs` em um projeto do .NET Core pelo exemplo acima. Para obter instruções, consulte [Implantar funções do Lambda em C# com arquivos .zip \(p. 668\)](#).

Python 3

Veja a seguir um exemplo de código Python que recebe dados de registro de eventos do Kinesis como uma entrada e a processa. Na ilustração, o código grava alguns dos dados de eventos de entrada no CloudWatch Logs.

Example ProcessKinesisRecords.py

```
from __future__ import print_function
import json
import base64
def lambda_handler(event, context):
    for record in event['Records']:
        #Kinesis data is base64 encoded so decode here
        payload=base64.b64decode(record["kinesis"]["data"])
```

```
    print("Decoded payload: " + str(payload))
```

Compreenda o código do exemplo para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Python com arquivos .zip \(p. 543\)](#).

Go

O seguinte é um exemplo de código Go que recebe dados de registros de eventos do Kinesis como uma entrada e os processa. Na ilustração, o código grava alguns dos dados de entrada no CloudWatch Logs.

Example ProcessKinesisRecords.go

```
import (
    "strings"
    "github.com/aws/aws-lambda-go/events"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) {
    for _, record := range kinesisEvent.Records {
        kinesisRecord := record.Kinesis
        dataBytes := kinesisRecord.Data
        dataText := string(dataBytes)

        fmt.Printf("%s Data = %s \n", record.EventName, dataText)
    }
}
```

Crie o executável com o `go build` e crie um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Go com arquivos .zip \(p. 640\)](#).

Modelo do AWS SAM para uma aplicação do Kinesis

Você pode criar esse aplicativo usando [AWS SAM](#). Para saber mais sobre como criar modelos do AWS SAM, consulte [Noções básicas de modelos do AWS SAM](#) no Guia do desenvolvedor do AWS Serverless Application Model.

Veja abaixo um modelo de exemplo do AWS SAM para a aplicação do Lambda do [tutorial \(p. 399\)](#). A função e o manipulador no modelo são para o código Node.js. Se você usar um exemplo de código diferente, atualize os valores adequadamente.

Example template.yaml – stream do Kinesis

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  LambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs12.x
      Timeout: 10
      Tracing: Active
    Events:
      Stream:
        Type: Kinesis
        Properties:
          Stream: !GetAtt stream.Arn
          BatchSize: 100
```

```
    StartingPosition: LATEST
stream:
  Type: AWS::Kinesis::Stream
  Properties:
    ShardCount: 1
Outputs:
  FunctionName:
    Description: "Function name"
    Value: !Ref LambdaFunction
  StreamARN:
    Description: "Stream ARN"
    Value: !GetAtt stream.Arn
```

O modelo cria uma função do Lambda, uma transmissão do Kinesis e um mapeamento de fontes de eventos. O mapeamento da fonte do evento lê o fluxo e invoca a função.

Para usar um [consumidor de fluxo HTTP/2 \(p. 389\)](#), crie o consumidor no modelo e configurar o mapeamento da fonte do evento para ler a partir do consumidor, e não do fluxo.

Example template.yaml – consumidor do stream do Kinesis

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: A function that processes data from a Kinesis stream.
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs12.x
      Timeout: 10
      Tracing: Active
      Events:
        Stream:
          Type: Kinesis
          Properties:
            Stream: !GetAtt streamConsumer.ConsumerARN
            StartingPosition: LATEST
            BatchSize: 100
  stream:
    Type: "AWS::Kinesis::Stream"
    Properties:
      ShardCount: 1
  streamConsumer:
    Type: "AWS::Kinesis::StreamConsumer"
    Properties:
      StreamARN: !GetAtt stream.Arn
      ConsumerName: "TestConsumer"
Outputs:
  FunctionName:
    Description: "Function name"
    Value: !Ref function
  StreamARN:
    Description: "Stream ARN"
    Value: !GetAtt stream.Arn
  ConsumerARN:
    Description: "Stream consumer ARN"
    Value: !GetAtt streamConsumer.ConsumerARN
```

Para obter informações sobre como empacotar e implantar o aplicativo sem servidor usando os comandos de empacotamento e implantação, consulte [Implantar aplicativos sem servidor](#) no Guia do desenvolvedor do AWS Serverless Application Model.

O uso do AWS Lambda Com o Amazon Lex

É possível usar o Amazon Lex para integrar um bot de conversação à sua aplicação. O bot do Amazon Lex fornece uma interface de conversação com seus usuários. O Amazon Lex fornece integração pré-criada com o Lambda, que permite usar uma função do Lambda com o bot do Amazon Lex.

Ao configurar um bot do Amazon Lex, é possível especificar uma função do Lambda para executar validação, atendimento ou ambos. Para validação, o Amazon Lex invoca a função do Lambda após cada resposta do usuário. A função do Lambda pode validar a resposta e fornecer feedback corretivo para o usuário, se necessário. Para atendimento, o Amazon Lex invoca a função do Lambda para atender à solicitação do usuário depois que o bot coleta com êxito todas as informações necessárias e recebe a confirmação do usuário.

É possível [gerenciar a simultaneidade \(p. 113\)](#) da função do Lambda para controlar o número máximo de conversas simultâneas do bot atendidas. A API do Amazon Lex retornará um código de status HTTP 429 — Too Many Requests (Excesso de solicitações) se a função estiver na simultaneidade máxima.

A API retornará um código de status HTTP 424 — Dependency Failed Exception (Exceção com falha de dependência) se a função do Lambda lançar uma exceção.

O bot do Amazon Lex invoca a função do Lambda [de forma síncrona \(p. 158\)](#). O parâmetro de evento contém informações sobre o bot e o valor de cada slot na caixa de diálogo. O parâmetro `invocationSource` indica se a função do Lambda deve validar as entradas (`DialogCodeHook`) ou atender à intenção (`FulfillmentCodeHook`).

Example Evento do Amazon Lex

```
{  
  "messageVersion": "1.0",  
  "invocationSource": "FulfillmentCodeHook",  
  "userId": "ABCD1234",  
  "sessionAttributes": {  
    "key1": "value1",  
    "key2": "value2",  
  },  
  "bot": {  
    "name": "OrderFlowers",  
    "alias": "prod",  
    "version": "1"  
  },  
  "outputDialogMode": "Text",  
  "currentIntent": {  
    "name": "OrderFlowers",  
    "slots": {  
      "FlowerType": "lilies",  
      "PickupDate": "2030-11-08",  
      "PickupTime": "10:00"  
    },  
    "confirmationStatus": "Confirmed"  
  }  
}
```

O Amazon Lex espera uma resposta de uma função do Lambda no formato a seguir. O campo `dialogAction` é obrigatório. Os campos `sessionAttributes` e `recentIntentSummaryView` são opcionais.

Example Evento do Amazon Lex

```
{
```

```
"sessionAttributes": {  
    "key1": "value1",  
    "key2": "value2"  
    ...  
},  
"recentIntentSummaryView": [  
{  
    "intentName": "Name",  
    "checkpointLabel": "Label",  
    "slots": {  
        "slot name": "value",  
        "slot name": "value"  
    },  
    "confirmationStatus": "None, Confirmed, or Denied (intent confirmation, if  
configured)",  
    "dialogActionType": "ElicitIntent, ElicitSlot, ConfirmIntent, Delegate, or Close",  
    "fulfillmentState": "Fulfilled or Failed",  
    "slotToElicit": "Next slot to elicit"  
}  
],  
"dialogAction": {  
    "type": "Close",  
    "fulfillmentState": "Fulfilled",  
    "message": {  
        "contentType": "PlainText",  
        "content": "Thanks, your pizza has been ordered."  
    },  
    "responseCard": {  
        "version": integer-value,  
        "contentType": "application/vnd.amazonaws.card.generic",  
        "genericAttachments": [  
            {  
                "title": "card-title",  
                "subTitle": "card-sub-title",  
                "imageUrl": "URL of the image to be shown",  
                "attachmentLinkUrl": "URL of the attachment to be associated with the card",  
                "buttons": [  
                    {  
                        "text": "button-text",  
                        "value": "Value sent to server on button click"  
                    }  
                ]  
            }  
        ]  
    }  
}  
}
```

Os campos adicionais necessários para `dialogAction` variam com base no valor do campo `type`. Para obter mais informações sobre os campos de evento e resposta, consulte [Formato de evento e resposta do Lambda](#) no Guia do desenvolvedor do Amazon Lex. Para obter um exemplo de tutorial que mostra como usar o Lambda com o Amazon Lex, consulte [Exercício 1: criar um bot do Amazon Lex usando um esquema](#) no Guia do desenvolvedor do Amazon Lex.

Funções e permissões

É necessário configurar uma função vinculada ao serviço como [Função de execução do \(p. 57\)](#). O Amazon Lex define a função vinculada ao serviço com permissões predefinidas. Ao criar um bot do Amazon Lex usando o console, a função vinculada ao serviço é criada automaticamente. Para criar uma função vinculada ao serviço com a AWS CLI da AWS, use o comando `create-service-linked-role`.

```
aws iam create-service-linked-role --aws-service-name lex.amazonaws.com
```

Esse comando cria a função a seguir.

```
{  
    "Role": {  
        "AssumeRolePolicyDocument": {  
            "Version": "2012-10-17",  
            "Statement": [  
                {  
                    "Action": "sts:AssumeRole",  
                    "Effect": "Allow",  
                    "Principal": {  
                        "Service": "lex.amazonaws.com"  
                    }  
                }  
            ]  
        },  
        "RoleName": "AWSServiceRoleForLexBots",  
        "Path": "/aws-service-role/lex.amazonaws.com/",  
        "Arn": "arn:aws:iam::account-id:role/aws-service-role/lex.amazonaws.com/  
AWSServiceRoleForLexBots"  
    }  
}
```

Se a função do Lambda usar outros serviços da AWS, será necessário adicionar as permissões correspondentes à função vinculada ao serviço.

Use uma política de permissões baseada em recursos para permitir que a intenção do Amazon Lex invoque a função do Lambda. Se você usar o console do Amazon Lex, a política de permissões será criada automaticamente. Na AWS CLI, use o comando `add-permission` do Lambda para definir a permissão. O exemplo a seguir define permissão para a intenção `OrderFlowers`.

```
aws lambda add-permission \  
--function-name OrderFlowersCodeHook \  
--statement-id LexGettingStarted-OrderFlowersBot \  
--action lambda:InvokeFunction \  
--principal lex.amazonaws.com \  
--source-arn "arn:aws:lex:us-east-1:123456789012 ID:intent:OrderFlowers:*
```

Usar o Lambda com o Amazon MQ

O Amazon MQ é um serviço gerenciado de agente de mensagem para o [Apache ActiveMQ](#) e o [RabbitMQ](#). Um agente de mensagens habilita aplicações de software e componentes para se comunicarem usando várias linguagens de programação, sistemas operacionais e protocolos de mensagens formais por meio de destinos de eventos de tópico ou fila.

O Amazon MQ também pode gerenciar instâncias do Amazon Elastic Compute Cloud (Amazon EC2) em seu nome instalando agentes do ActiveMQ ou RabbitMQ e fornecendo diferentes topologias de rede e outras necessidades de infraestrutura.

Você pode usar uma função do Lambda para processar registros do seu agente de mensagens do Amazon MQ. O Lambda invoca sua função por meio de um [mapeamento de fontes de eventos \(p. 170\)](#), um recurso do Lambda que lê mensagens de seu agente e invoca a função [de modo síncrono \(p. 158\)](#).

O mapeamento de fontes de eventos do Amazon MQ tem as seguintes restrições de configuração:

- Contas cruzadas: o Lambda não é compatível com o processamento de contas cruzadas. Não é possível usar o Lambda para processar registros de um agente de mensagens do Amazon MQ que esteja em uma conta da AWS diferente.
- Autenticação: para o ActiveMQ, somente o [SimpleAuthenticationPlugin](#) do ActiveMQ é compatível. Para RabbitMQ, somente o [PLAIN](#)Mecanismo de autenticação é compatível. Os usuários devem usarAWS Secrets Managerpara gerenciar suas credenciais. Para obter mais informações sobre a autenticação do ActiveMQ, consulte[Integração de corretores ActiveMQ com LDAP](#)noGuia do desenvolvedor do Amazon MQ.
- Cota de conexão: os agentes têm um número máximo de conexões permitidas por protocolo de nível de transmissão de dados. Essa cota é baseada no tipo de instância do agente. Para obter mais informações, consulte o [Operadores](#)seção doCotas no Amazon MQnoGuia do desenvolvedor do Amazon MQ.
- Conectividade: você pode criar agentes em uma Virtual Private Cloud (VPC) pública ou privada. Para VPCs privadas, sua função do Lambda precisa acessar a VPC para receber mensagens. Para obter mais informações, consulte [the section called “API do mapeamento da fonte de eventos” \(p. 415\)](#) mais adiante neste tópico.
- Destinos de eventos: somente destinos de fila são compatíveis. No entanto, você pode usar um tópico virtual, que se comporta como um tópico internamente enquanto interage com o Lambda como uma fila. Para obter mais informações, consulte [Virtual Destinations](#) no site do Apache ActiveMQ e [Virtual Hosts](#) no site do RabbitMQ.
- Topologia de rede: para o ActiveMQ, somente um agente de instância única ou em espera é aceito por mapeamento de fontes de eventos. Para o RabbitMQ, apenas um agente de instância única ou implantação de cluster é aceito por mapeamento de fonte de eventos. Os agentes de instância única requerem um endpoint de failover. Para obter mais informações sobre esses modos de implantação do agente, consulte[Arquitetura de operador do MQeArquitetura de MQ do Rabbit](#)noGuia do desenvolvedor do Amazon MQ.
- Protocolos — Os protocolos suportados dependem do tipo de integração do Amazon MQ.
 - Para integrações do ActiveMQ, o Lambda consome mensagens usando o protocolo OpenWire/Java Message Service (JMS). Nenhum outro protocolo é compatível para o consumo de mensagens. Dentro do protocolo JMS, somente [TextMessage](#) e [BytesMessage](#) são compatíveis. Para obter mais informações sobre o protocolo OpenWire, consulte [OpenWire](#) no site do Apache ActiveMQ.
 - Para integrações RabbitMQ, o Lambda consome mensagens usando o protocolo AMQP 0-9-1. Nenhum outro protocolo é compatível para o consumo de mensagens. Para obter mais informações sobre a implementação do protocolo AMQP 0-9-1 pelo RabbitMQ, consulte[AMQP 0-9-1 Guia de referência completa](#)no site do RabbitMQ.

O Lambda oferece suporte automaticamente às versões mais recentes do ActiveMQ e RabbitMQ que não têm suporte no Amazon MQ. Para obter as versões compatíveis mais recentes, consulte[Notas de versão do Amazon MQ](#)noGuia do desenvolvedor do Amazon MQ.

Note

Por padrão, o Amazon MQ tem uma janela de manutenção semanal para agentes. Durante essa janela de tempo, os agentes não estão disponíveis. Para agentes sem espera, o Lambda não é possível processar nenhuma mensagem durante essa janela.

Seções

- [Grupo de consumidores do Lambda \(p. 412\)](#)
- [Permissões da função de execução \(p. 414\)](#)
- [Configurar um agente como uma fonte de eventos \(p. 414\)](#)
- [API do mapeamento da fonte de eventos \(p. 415\)](#)
- [Erros de mapeamento da fonte de eventos \(p. 417\)](#)

Grupo de consumidores do Lambda

Para interagir com o Amazon MQ, o Lambda cria um grupo de consumidores que pode ler a partir de seus agentes do Amazon MQ. O grupo de consumidores é criado com o mesmo ID que um UUID de mapeamento da fonte de eventos.

O Lambda extrairá as mensagens até que ele tenha processado um máximo de 6 MB, até o tempo limite ou até que o tamanho do lote seja atendido. Quando configurado, o tamanho do lote determina o número máximo de itens a serem recuperados em um único lote. Seu lote é convertido em uma carga útil do Lambda e sua função de destino é invocada. As mensagens não são persistentes nem desserializadas. Em vez disso, eles são recuperados pelo grupo de consumidores como um BLOB de bytes e são codificados em base64 para uma carga útil JSON.

Note

O tempo máximo de invocação da função é de 14 minutos.

Você pode monitorar o uso da simultaneidade de uma determinada função usando a métrica `ConcurrentExecutions` no Amazon CloudWatch. Para obter mais informações sobre a simultaneidade, consulte [the section called “Simultaneidade” \(p. 113\)](#).

Example Eventos de registro do Amazon MQ

ActiveMQ

```
{
  "eventSource": "aws:amq",
  "eventSourceArn": "arn:aws:mq:us-west-2:112556298976:broker:test:b-9bcfa592-423a-4942-879d-eb284b418fc8",
  "messages": [
    [
      {
        "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-west-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
        "messageType": "jms/text-message",
        "data": "QUJDOkFBQUE=",
        "connectionId": "myJMSSoID",
        "redelivered": false,
        "destination": {
          "queueName": "myQueue"
        }
      }
    ]
  ]
}
```

```
        "physicalname": "testQueue"
    },
    "timestamp": 1598827811958,
    "brokerInTime": 1598827811958,
    "brokerOutTime": 1598827811959
},
{
    "messageID": "ID:b-9bcfa592-423a-4942-879d-eb284b418fc8-1.mq.us-
west-2.amazonaws.com-37557-1234520418293-4:1:1:1:1",
    "messageType": "jms/bytes-message",
    "data": "3DTOOW7crj51prgVLQaGQ82S48k=",
    "connectionId": "myJMSCoID1",
    "persistent": false,
    "destination": {
        "physicalname": "testQueue"
    },
    "timestamp": 1598827811958,
    "brokerInTime": 1598827811958,
    "brokerOutTime": 1598827811959
}
]
```

RabbitMQ

```
{
    "eventSource": "aws:rmq",
    "eventSourceArn": "arn:aws:mq:us-
west-2:112556298976: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": null
        ]
    ]
}
```

```
        "expiration": "60000",
        "messageId": null,
        "timestamp": "Jan 1, 1970, 12:33:41 AM",
        "type": null,
        "userId": "AIDACKCEVSO6C2EXAMPLE",
        "appId": null,
        "clusterId": null,
        "bodySize": 80
    },
    "redelivered": false,
    "data": "eyJ0aW1lb3V0IjowLCJkYXRhIjoiQ1pybWYwR3c4T3Y0YnFMUXhENEUiifQ=="
}
]
}
```

Note

No exemplo RabbitMQ.pizzaQueue é o nome da fila do RabbitMQ e é o nome do host virtual. Ao receber mensagens, a origem do evento lista as mensagens e pizzaQueue::/.

Permissões da função de execução

Para ler registros de um agente do Amazon MQ, sua função do Lambda precisa das seguintes permissões adicionadas à respectiva [função de execução \(p. 57\)](#):

- mq:DescribeBroker
- secretsmanager:GetSecretValue
- ec2>CreateNetworkInterface
- ec2>DeleteNetworkInterface
- ec2:DescribeNetworkInterfaces
- ec2:DescribeSecurityGroups
- ec2:DescribeSubnets
- ec2:DescribeVpcs
- logs>CreateLogGroup
- logs>CreateLogStream
- logs:PutLogEvents

Note

Ao usar uma chave criptografada gerenciada pelo cliente, adicione a permissão `kms:Decrypt` também.

Configurar um agente como uma fonte de eventos

Crie um [mapeamento de fontes de eventos \(p. 170\)](#) para instruir o Lambda a enviar registros de um agente do Amazon MQ para uma função do Lambda. É possível criar vários mapeamentos de origem de evento para processar os mesmos dados com várias funções ou processar itens de várias fontes com uma única função.

Para configurar sua função para ler do Amazon MQ, crie um acionador do MQ no console do Lambda.

Para criar um trigger

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Em Function overview (Visão geral da função), escolha Add trigger (Adicionar gatilho).
4. Escolha um tipo de acionador.
5. Configure as opções necessárias e escolha Add (Adicionar).

O Lambda oferece suporte às seguintes opções de fontes de eventos do Amazon MQ:

- Agente do MQ— Selecione um corretor do Amazon MQ.
- Batch size (Tamanho do lote): defina o número máximo de mensagens a serem recuperadas em um único lote.
- Queue name (Nome da fila): digite a fila do Amazon MQ a ser consumida.
- Configuração do acesso à fonte— Insira as informações do host virtual e o segredo do Secrets Manager que armazena as credenciais do agente.
- Enable trigger (Habilitar acionador): desabilite o acionador para interromper o processamento de registros.

Para habilitar ou desabilitar o trigger (ou excluí-lo), selecione o trigger do MQ no designer. Para reconfigurar o trigger, use as operações da API de mapeamento de fontes de eventos.

API do mapeamento da fonte de eventos

Para gerenciar uma origem de evento com a [AWS CLI](#) ou o [AWS SDK](#), você pode usar as seguintes operações da API:

- [CreateEventSourceMapping \(p. 786\)](#)
- [ListEventSourceMappings \(p. 888\)](#)
- [GetEventSourceMapping \(p. 837\)](#)
- [UpdateEventSourceMapping \(p. 956\)](#)
- [DeleteEventSourceMapping \(p. 812\)](#)

Para criar o mapeamento da fonte de eventos com a AWS Command Line Interface (AWS CLI), use o comando `create-event-source-mapping`.

Por padrão, os agentes do Amazon MQ são criados com a sinalização `PubliclyAccessible` definida como falsa. O agente recebe um endereço IP público somente quando `PubliclyAccessible` é definido como verdadeiro.

Para ter acesso total ao mapeamento de fontes de eventos, seu agente deve usar um endpoint público ou fornecer acesso à VPC. Para atender aos requisitos de acesso do Amazon Virtual Private Cloud (Amazon VPC), é possível executar um dos seguintes procedimentos:

- Configure um gateway NAT por sub-rede pública. Para obter mais informações, consulte [Acesso aos serviços e à Internet para funções conectadas à VPC \(p. 129\)](#).
- Crie uma conexão entre a Amazon VPC e o Lambda. Sua Amazon VPC também deve se conectar ao AWS Security Token Service(AWS STS) e endpoints do Secrets Manager. Para obter mais informações, consulte [Configurar VPC endpoints de interface para o Lambda \(p. 131\)](#).

As regras de grupos de segurança da Amazon VPC que você configurar deverão ter as seguintes configurações, no mínimo:

- Regras de entrada: para um agente sem acessibilidade pública, permita todo o tráfego em todas as portas para o grupo de segurança especificado como sua fonte. Para um corretor com acessibilidade pública, permita todo o tráfego em todas as portas para todos os destinos.
- Regras de saída: permitir todo o tráfego em todas as portas para todos os destinos.

A configuração da Amazon VPC é detectável pela [API do Amazon MQ](#) e não precisa ser configurada nas definições de `create-event-source-mapping`.

O exemplo de comando da AWS CLI a seguir cria uma fonte de eventos que mapeia uma função do Lambda chamada `MQ-Example-Function` para um agente baseado em Amazon MQ RabbitMQ chamado `ExampleMQBroker`. O comando também fornece o nome do host virtual e o ARN de um segredo do Secrets Manager que armazena as credenciais do agente.

```
aws lambda create-event-source-mapping \
--event-source-arn arn:aws:mq:us-east-1:123456789012:broker:ExampleMQBroker:b-24cacbb4-
b295-49b7-8543-7ce7ce9dfb98 \
--function-name arn:aws:lambda:us-east-1:123456789012:function:MQ-Example-Function \
--queues ExampleQueue \
--source-access-configuration Type=VIRTUAL_HOST,URI="/" \
Type=BASIC_AUTH,URI=arn:aws:secretsmanager:us-
east-1:123456789012:secret:ExampleMQBrokerUserPassword-xPBMTt \
```

Você deve ver a saída a seguir:

```
{
    "UUID": "91eaeb7e-c976-1234-9451-8709db01f137",
    "BatchSize": 100,
    "EventSourceArn": "arn:aws:mq:us-east-1:123456789012:broker:ExampleMQBroker:b-b4d492ef-
bdc3-45e3-a781-cd1a3102ecc",
    "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:MQ-Example-Function",
    "LastModified": 1601927898.741,
    "LastProcessingResult": "No records processed",
    "State": "Creating",
    "StateTransitionReason": "USER_INITIATED",
    "Queues": [
        "ExampleQueue"
    ],
    "SourceAccessConfigurations": [
        {
            "Type": "BASIC_AUTH",
            "URI": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:ExampleMQBrokerUserPassword-xPBMTt"
        }
    ]
}
```

Usando o comando `update-event-source-mapping`, você pode configurar opções adicionais, por exemplo, como o Lambda processa os lotes, e especificar quando descartar registros que não podem ser processados. O exemplo de comando a seguir atualiza um mapeamento de fonte de eventos para ter um tamanho de lote igual a 2.

```
aws lambda update-event-source-mapping \
--uuid 91eaeb7e-c976-1234-9451-8709db01f137 \
--batch-size 2
```

Você deve ver a saída a seguir:

```
{
    "UUID": "91eaeb7e-c976-1234-9451-8709db01f137",
```

```
"BatchSize": 2,  
"EventSourceArn": "arn:aws:mq:us-east-1:123456789012:broker:ExampleMQBroker:b-b4d492ef-  
bdc3-45e3-a781-cd1a3102ecc",  
"FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:MQ-Example-Function",  
"LastModified": 1601928393.531,  
"LastProcessingResult": "No records processed",  
"State": "Updating",  
"StateTransitionReason": "USER_INITIATED"  
}
```

O Lambda atualiza essas configurações de forma assíncrona. A saída não refletirá as alterações até que esse processo seja concluído. Use o comando `get-event-source-mapping` para visualizar o status atual do seu recurso.

```
aws lambda get-event-source-mapping \  
--uuid 91eaeb7e-c976-4939-9451-8709db01f137
```

Você deve ver a saída a seguir:

```
{  
    "UUID": "91eaeb7e-c976-4939-9451-8709db01f137",  
    "BatchSize": 2,  
    "EventSourceArn": "arn:aws:mq:us-east-1:123456789012:broker:ExampleMQBroker:b-b4d492ef-  
bdc3-45e3-a781-cd1a3102ecc",  
    "FunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:MQ-Example-Function",  
    "LastModified": 1601928393.531,  
    "LastProcessingResult": "No records processed",  
    "State": "Enabled",  
    "StateTransitionReason": "USER_INITIATED"  
}
```

Erros de mapeamento da fonte de eventos

Quando uma função do Lambda encontra um erro irrecuperável, o consumidor do Amazon MQ interrompe o processamento de registros. Qualquer outro consumidor pode continuar o processamento, contanto que não encontre o mesmo erro. Para determinar a possível causa de um consumidor interrompido, verifique o campo `StateTransitionReason` nos detalhes de devolução do `EventSourceMapping` para obter um dos seguintes códigos:

ESM_CONFIG_NOT_VALID

A configuração do mapeamento da fonte de eventos não é válida.

EVENT_SOURCE_AUTHN_ERROR

O Lambda falhou ao autenticar a fonte de eventos.

EVENT_SOURCE_AUTHZ_ERROR

O Lambda não tem as permissões necessárias para acessar a fonte de eventos.

FUNCTION_CONFIG_NOT_VALID

A configuração da função não é válida.

Os registros também não serão processados se o Lambda os descartar devido ao seu tamanho. O limite de tamanho para registros do Lambda é de 6 MB. Para entregar mensagens novamente após um erro da função, use uma Dead Letter Queue (DLQ – Fila de mensagens mortas). Para obter mais informações, consulte [Message Redelivery and DLQ Handling](#) no site do Apache ActiveMQ e [Reliability Guide](#) no site do RabbitMQ.

Note

O Lambda não oferece suporte às políticas de reentrega personalizadas. Em vez disso, o Lambda usa uma política com os valores padrão da propriedade[Política de Reentrega](#)no site do Apache ActiveMQ, com `maximumRedeliveries` definido como 5.

Usar o Lambda com o Amazon MSK

O [Amazon Managed Streaming for Apache Kafka \(Amazon MSK\)](#) é um serviço totalmente gerenciado que você pode usar para criar e executar aplicações que usam o Apache Kafka para processar dados em streaming. O Amazon MSK simplifica a configuração, o dimensionamento e o gerenciamento de clusters que executam o Kafka. O Amazon MSK também facilita a configuração de seu aplicativo para várias zonas de disponibilidade e para segurança com AWS Identity and Access Management(IAM). Além disso, o Amazon MSK é compatível com várias versões de código aberto do Kafka.

O Amazon MSK como uma origem de evento funciona de forma semelhante ao uso do Amazon Simple Queue Service (Amazon SQS) ou do Amazon Kinesis. O Lambda pesquisa internamente por novas mensagens da origem do evento e, em seguida, chama de forma síncrona a função do Lambda de destino. O Lambda lê as mensagens em lotes e fornece estas para a sua função como uma carga de eventos. O tamanho máximo do lote é configurável. (O valor padrão é de 100 mensagens.)

Para obter um exemplo de como configurar o Amazon MSK como uma origem de evento, consulte [Usar o Amazon MSK como uma origem de evento para o AWS Lambda](#) no AWS Blogue de computação. Além disso, consulte [Integração do Lambda no Amazon MSK](#), Amazon MSK Labs, para um tutorial completo.

O Lambda lê as mensagens sequencialmente para cada partição. Depois que o Lambda processa cada lote, ele confirma os deslocamentos das mensagens nesse lote. Se sua função retorna um erro para qualquer uma das mensagens em um lote, o Lambda tenta novamente todo o lote de mensagens até que o processamento seja bem-sucedido ou as mensagens expiram.

Lambda permite que uma função seja executada por até 14 minutos antes de interrompê-la.

O Lambda envia o lote de mensagens no parâmetro de evento quando ele chama sua função. O payload do evento contém uma matriz de mensagens. Cada item de array contém detalhes do tópico do Amazon MSK e do identificador de partição, juntamente com um carimbo de data/hora e uma mensagem codificada em base64.

```
{
  "eventSource": "aws:kafka",
  "eventSourceArn": "arn:aws:kafka:sa-east-1:123456789012:cluster/vpc-2priv-2pub/751d2973-a626-431c-9d4e-d7975eb44dd7-2",
  "records": [
    "mytopic-0": [
      {
        "topic": "mytopic",
        "partition": "0",
        "offset": 15,
        "timestamp": 1545084650987,
        "timestampType": "CREATE_TIME",
        "value": "SGVsbG8sIHRoaXMgYSB0ZXN0Lg==",
        "headers": [
          {
            "headerKey": [
              104,
              101,
              97,
              100,
              101,
              114,
              86,
              97,
              108,
              117,
              101
            ]
          }
        ]
      }
    ]
}
```

```
        ]  
    }]  
}
```

Tópicos

- [Gerenciar acesso e permissões \(p. 420\)](#)
- [Configuração de rede \(p. 380\)](#)
- [Adicionar o Amazon MSK como uma origem de evento \(p. 421\)](#)
- [Auto scaling da origem do evento do Amazon MSK \(p. 423\)](#)
- [Parâmetros de configuração do Amazon MSK \(p. 423\)](#)

Gerenciar acesso e permissões

Para que o Lambda enquete seu tópico do Kafka e atualize outros recursos de cluster, sua função do Lambda, bem como seus usuários e funções do IAM, devem ter as seguintes permissões.

Permissões de função do Lambda necessárias

A [função de execução \(p. 57\)](#) da execução do Lambda deve ter permissão para ler registros de seu cluster do Amazon MSK em seu nome. Você pode adicionar a política gerenciada da AWS `AWSLambdaMSKExecutionRole` à sua função de execução ou criar uma política personalizada com permissão para realizar as seguintes ações:

- `kafka:DescribeCluster`
- `kafka:GetBootstrapBrokers`
- `ec2>CreateNetworkInterface`
- `ec2:DescribeNetworkInterfaces`
- `ec2:DescribeVpcs`
- `ec2>DeleteNetworkInterface`
- `ec2:DescribeSubnets`
- `ec2:DescribeSecurityGroups`
- `logs>CreateLogGroup`
- `logs>CreateLogStream`
- `logs:PutLogEvents`

Adicionar uma política à sua função de execução

Siga estas etapas para adicionar a política gerenciada pela AWS `AWSLambdaMSKExecutionRole` à sua função de execução usando o console do IAM.

Para adicionar uma política gerenciada da AWS

1. Abra a [página Policies \(Políticas\)](#) do console do IAM.
2. Na caixa de pesquisa, insira o nome da política (`AWSLambdaMSKExecutionRole`).
3. Selecione a política na lista e, em seguida, escolha Policy actions (Ações de política), Attach (Associar).
4. No Anexar política, selecione sua função de execução na lista e escolha Anexar política.

Conceder acesso aos usuários com uma política do IAM

Por padrão, usuários e funções do IAM não têm permissão para executar operações de API do Amazon MSK. Para conceder acesso a usuários em sua organização ou conta, você pode precisar de uma política baseada em identidade. Para obter mais informações, consulte [Exemplos de políticas baseadas em identidade do Amazon MSK](#) no Guia do desenvolvedor do Amazon Managed Streaming for Apache Kafka.

Usar a autenticação SASL/SCRAM

O Amazon MSK é compatível com autenticação SASL/SCRAM (Simple Authentication e Security Layer/Salted Challenge Response Authentication Mechanism) com criptografia TLS. Você pode controlar o acesso aos seus clusters do Amazon MSK configurando a autenticação de nome de usuário e senha usando um segredo do AWS Secrets Manager. Para obter mais informações, consulte [autenticação de nome de usuário e senha com AWS Secrets Manager](#) no Guia do desenvolvedor do Amazon Managed Streaming for Apache Kafka.

Observe que o Amazon MSK não oferece suporte à autenticação SASL/PLAIN.

Configuração de rede

O Lambda deve ter acesso aos recursos do Amazon Virtual Private Cloud (Amazon VPC) associados ao cluster do Amazon MSK. Recomendamos que você implante AWS PrivateLink [VPC endpoints](#) para o Lambda e o AWS Security Token Service (AWS STS). Se a autenticação for necessária, implante também um endpoint da VPC para o Secrets Manager.

Como alternativa, verifique se a VPC associada ao cluster do Amazon MSK inclui um gateway NAT por sub-rede pública. Para obter mais informações, consulte [Acesso aos serviços e à Internet para funções conectadas à VPC \(p. 129\)](#).

Seus grupos de segurança da Amazon VPC devem ser configurados com as seguintes regras (no mínimo):

- Regras de entrada: permitir todo o tráfego em todas as portas do grupo de segurança especificado como a fonte de eventos.
- Regras de saída: permitir todo o tráfego em todas as portas para todos os destinos.

Note

Sua configuração da Amazon VPC pode ser detectada por meio do [API do Amazon MSK](#), e não precisa ser configurado durante a configuração usando o `create-event-source-mapping` comando.

Para obter mais informações sobre como configurar a rede, consulte [Configurar o AWS Lambda com um cluster Apache Kafka em uma VPC](#) no AWS Blogue de computação.

Adicionar o Amazon MSK como uma origem de evento

Para criar um [Mapeamento de origens](#) (p. 170), adicione o Amazon MSK como uma função do Lambda [Trigger](#) (p. 18) usando o console do Lambda, o [AWSSDK](#), ou o [AWS Command Line Interface \(AWS CLI\)](#).

Esta seção descreve como criar um mapeamento de origens de eventos usando o console do Lambda e o AWS CLI.

Prerequisites

- Um cluster do Amazon MSK e um tópico do Kafka. Para obter mais informações, consulte [Conceitos básicos do uso do Amazon MSK](#) no Guia do desenvolvedor do Amazon Managed Streaming for Apache Kafka.

- Uma função de execução do Lambda (p. 57) com permissão para acessar os recursos da AWS que seu cluster do Amazon MSK usa.

Adicionar um gatilho do Amazon MSK (console)

Siga estas etapas para adicionar seu cluster do Amazon MSK e um tópico do Kafka como um acionador para sua função do Lambda.

Para adicionar um acionador do Amazon MSK à sua função do Lambda (console)

1. Abra a página [Functions](#) (Funções) no console do Lambda.
2. Escolha o nome da função do Lambda.
3. Em Function overview (Visão geral da função), escolha Add trigger (Adicionar gatilho).
4. Em Trigger configuration (Configuração do acionador), faça o seguinte:
 - a. Selecione oMSKTipo de gatilho.
 - b. para oCluster do MSKSelecione seu cluster do.
 - c. Em Batch size (Tamanho do lote), insira o número máximo de mensagens a serem recebidas em um único lote.
 - d. Em Topic name (Nome do tópico), insira um nome para o tópico do Kafka.
 - e. (Opcional) ParaPosição inicial, escolhaMais recentepara começar a ler o fluxo do registro mais recente. Ou escolhaHorizonte de cortepara iniciar no registro mais antigo disponível.
 - f. (Opcional) ParaChave secreta, escolha a chave secreta para autenticação SASL/SCRAM dos corretores em seu cluster do Amazon MSK.
 - g. Para criar o gatilho em um estado desativado para teste (recomendado), desmarqueAtivar gatilho. Ou, para habilitar o gatilho imediatamente, selecioneAtivar gatilho.
5. Para criar o acionador, selecione Add (Adicionar).

Adicionando um gatilho do Amazon MSK (AWS CLI)

Use os comandos de exemplo a seguir da AWS CLI para criar e visualizar um acionador do Amazon MSK para sua função do Lambda.

Criar um trigger usando a AWS CLI

O exemplo a seguir usa `aws lambda create-event-source-mapping` AWS CLI para mapear uma função do Lambda chamada `my-kafka-function` para um tópico do Kafka chamado `AWSKafkaTopic`. A posição inicial do tópico está definida como `LATEST`.

```
aws lambda create-event-source-mapping \
--event-source-arn arn:aws:kafka:us-west-2:arn:aws:kafka:us-west-2:111111111111:cluster/
my-cluster/fc2f5bdf-fd1b-45ad-85dd-15b4a5a6247e-2 \
--topics AWSKafkaTopic \
--starting-position LATEST \
--function-name my-kafka-function
```

Para obter mais informações, consulte a documentação de referência da API.

Visualizar o status usando a AWS CLI

O exemplo a seguir usa o comando `aws lambda get-event-source-mapping` da AWS CLI para descrever o status do mapeamento de fontes de eventos que você criou.

```
aws lambda get-event-source-mapping \
```

--uuid **6d9bce8e-836b-442c-8070-74e77903c815**

Auto scaling da origem do evento do Amazon MSK

Quando você cria inicialmente uma origem de evento do Amazon MSK, o Lambda aloca um consumidor para processar todas as partições no tópico do Kafka. O Lambda aumenta ou diminui automaticamente o número de consumidores, com base na workload. Para preservar a ordenação de mensagens em cada partição, o número máximo de consumidores é um consumidor por partição no tópico.

A cada 15 minutos, o Lambda avalia o atraso de compensação do consumidor de todas as partições no tópico. Se o atraso for muito alto, a partição está recebendo mensagens mais rápido do que o Lambda pode processá-las. Se necessário, o Lambda adiciona ou remove os consumidores do tópico.

Se sua função alvo do Lambda estiver sobrecarregada, o Lambda reduz o número de consumidores. Essa ação reduz a workload na função, reduzindo o número de mensagens que os consumidores podem recuperar e enviar para a função.

Para monitorar a taxa de transferência do seu tópico do Kafka, você pode visualizar a[Métricas de consumo do Amazon MSK](#). Para ajudar você a encontrar as métricas para essa função do Lambda, o valor do consumer group logs é definido como o UUID de origem de evento.

Para verificar quantas invocações de função ocorrem em paralelo, você também pode monitorar o[métricas de simultaneidade \(p. 719\)](#)Para sua função.

Parâmetros de configuração do Amazon MSK

Todos os tipos de origem de evento Lambda compartilham o mesmo[CreateEventSourceMapping \(p. 786\)](#)e[UpdateEventSourceMapping \(p. 956\)](#)Operações de API do. No entanto, apenas alguns dos parâmetros se aplicam ao Amazon MSK.

Parâmetros de origem de evento que se aplicam ao Amazon MSK

Parâmetro	Obrigatório	Padrão	Observações
BatchSize	N	100	Máximo: 10.000.
Enabled	N	Enabled	
EventSourceArn	Y		Pode definir apenas em Criar
FunctionName	Y		
SourceAccessConfiguration	N	Sem credenciais do	Informações da VPC ou credenciais de autenticação SASL/SCRAM para sua origem de evento
StartingPosition	Y		TRIM_HORIZON Pode definir apenas em Criar
Tópicos	Y		Nome do tópico do Kafka Pode definir apenas em Criar

O uso do AWS Lambda Com o Amazon RDS

Você pode usar o AWS Lambda Para processar notificações de eventos de um banco de dados do Amazon Relational Database Service (Amazon RDS). O Amazon RDS envia notificações para um tópico do Amazon Simple Notification Service (Amazon SNS), que você pode configurar para invocar uma função do Lambda. O Amazon SNS envolve a mensagem do Amazon RDS em seu próprio documento de evento e a envia para sua função.

Example Mensagem do Amazon RDS em um evento do 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": "2019-01-02T12:45:07.000Z",  
                "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\":\"2019-01-02  
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.amazonaws.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&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"  
            }  
        }  
    ]  
}
```

Tópicos

- [Tutorial: configurar uma função do Lambda para acessar o Amazon RDS em uma Amazon VPC \(p. 425\)](#)
- [Configurar a função \(p. 429\)](#)

Tutorial: configurar uma função do Lambda para acessar o Amazon RDS em uma Amazon VPC

Neste tutorial, você faz o seguinte:

- Inicie uma instância do mecanismo de banco de dados MySQL do Amazon RDS na Amazon VPC padrão. Na instância MySQL, crie um banco de dados (ExampleDB) com uma tabela de amostra (Funcionários). Para obter mais informações sobre eventos do Amazon RDS, consulte [Amazon RDS](#).
- Crie uma função do Lambda para acessar o banco de dados ExampleDB, crie uma tabela (Funcionários), adicione alguns registros e recupere-os da tabela.

- Invoque a função do Lambda e verifique os resultados da consulta. É assim que você verifica se a função do Lambda foi capaz de acessar a instância MySQL do RDS na VPC.

Para obter detalhes sobre o uso do Lambda com a Amazon VPC, consulte [Configurar uma função do Lambda para acessar recursos em uma VPC \(p. 124\)](#).

Prerequisites

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em [Conceitos básicos do Lambda \(p. 9\)](#) para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, você precisa de um terminal de linha de comando ou de um shell para executar os comandos. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

Você deve ver a saída a seguir:

```
aws-cli/2.0.57 Python/3.7.4 Darwin/19.6.0 exe/x86_64
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

Criar a função de execução

Crie a [função de execução \(p. 57\)](#) que dá à sua função permissão para acessar recursos do AWS.

Para criar uma função de execução

1. Abra a [página Roles](#) (Funções) no console do IAM.
2. Selecione Create role.
3. Crie uma função com as seguintes propriedades.
 - Entidade confiável: Lambda.
 - Permissions (Permissões): AWSLambdaVPCAccessExecutionRole.
 - Nome da função – **lambda-vpc-role**.

A AWSLambdaVPCAccessExecutionRole tem as permissões que a função precisa para gerenciar conexões de rede com uma VPC.

Crie uma instância de banco de dados do Amazon RDS

Neste tutorial, a função do Lambda de exemplo cria uma tabela (Funcionários), insere alguns registros e, em seguida, os recupera. A tabela que a função do Lambda cria tem o seguinte esquema:

```
Employee(EmpID, Name)
```

O EmpID é a chave primária. Agora, você precisa adicionar alguns registros a essa tabela.

Primeiro, inicie uma instância MySQL do RDS na VPC padrão com o banco de dados ExampleDB. Se você já tem uma instância MySQL do RDS em execução na sua VPC padrão, ignore essa etapa.

Você pode iniciar uma instância MySQL do RDS usando um dos seguintes métodos:

- Siga as instruções em [Criar uma instância de banco de dados MySQL e conectar a um banco de dados em uma instância de banco de dados MySQL](#) no Manual do usuário do Amazon RDS.
- Use o seguinte comando AWS CLI:

```
aws rds create-db-instance --db-name ExampleDB --engine MySQL \
--db-instance-identifier MySQLForLambdaTest --backup-retention-period 3 \
--db-instance-class db.t2.micro --allocated-storage 5 --no-publicly-accessible \
--master-username username --master-user-password password
```

Anote o nome do banco de dados, o nome do usuário e a senha. Você também precisa do endereço do host (endpoint) da instância de Banco de Dados, que pode obter do console do RDS. Pode ser necessário aguardar até que o status da instância esteja disponível e o valor do endpoint seja exibido no console.

Criar um pacote de implantação

O código em Python de exemplo a seguir executa uma consulta SELECT para a tabela Funcionários na instância MySQL do RDS que você criou na VPC. O código cria uma tabela no banco de dados ExampleDB, adiciona registros de amostra e recupera esse registros.

O método a seguir para lidar com credenciais de banco de dados é apenas para fins ilustrativos. Em um ambiente de produção, recomendamos usar o AWS Secrets Manager em vez de variáveis de ambiente para armazenar credenciais de banco de dados. Para obter mais informações, consulte [Configurar o acesso ao banco de dados para uma função do Lambda](#).

Example app.py

```
import sys
import logging
import rds_config
import pymysql
#rds settings
rds_host = "rds-instance-endpoint"
name = rds_config.db_username
password = rds_config.db_password
db_name = rds_config.db_name

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

try:
    conn = pymysql.connect(host=rds_host, user=name, passwd=password, db=db_name,
                           connect_timeout=5)
except pymysql.MySQLError as e:
    logger.error("ERROR: Unexpected error: Could not connect to MySQL instance.")
    logger.error(e)
    sys.exit()

logger.info("SUCCESS: Connection to RDS MySQL instance succeeded")
def handler(event, context):
    """
    This function fetches content from MySQL RDS instance
    """

    item_count = 0
```

```
with conn.cursor() as cur:
    cur.execute("create table Employee ( EmpID  int NOT NULL, Name varchar(255) NOT
NULL, PRIMARY KEY (EmpID))")
    cur.execute('insert into Employee (EmpID, Name) values(1, "Joe")')
    cur.execute('insert into Employee (EmpID, Name) values(2, "Bob")')
    cur.execute('insert into Employee (EmpID, Name) values(3, "Mary")')
    conn.commit()
    cur.execute("select * from Employee")
    for row in cur:
        item_count += 1
        logger.info(row)
        #print(row)
    conn.commit()

return "Added %d items from RDS MySQL table" %(item_count)
```

A execução de `pymysql.connect()` fora do manipulador permite que sua função reutilize a conexão com o banco de dados para proporcionar uma melhor performance.

Um segundo arquivo contém informações de conexão para a função.

Example `rds_config.py`

```
#config file containing credentials for RDS MySQL instance
db_username = "username"
db_password = "password"
db_name = "ExampleDB"
```

O pacote de implantação é um arquivo .zip que contém o código de sua função do Lambda e as dependências. O código de exemplo do tem as seguintes dependências:

Dependencies

- `pymysql`: o código da função do Lambda usa essa biblioteca para acessar a instância MySQL (consulte [PyMySQL](#)).

Como criar um pacote de implantação

- Instale dependências com o Pip e crie um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Python com arquivos .zip \(p. 543\)](#).

Criar a função do Lambda

Crie a função do Lambda com o comando `create-function`. Você pode encontrar os IDs de sub-rede e o ID do grupo de segurança da VPC padrão no [console do Amazon VPC](#).

```
aws lambda create-function --function-name CreateTableAddRecordsAndRead --runtime
python3.8 \
--zip-file fileb://app.zip --handler app.handler \
--role arn:aws:iam::123456789012:role/lambda-vpc-role \
--vpc-config SubnetIds=subnet-0532bb6758ce7c71f,subnet-
d6b7fda068036e11f,SecurityGroupIds=sg-0897d5f549934c2fb
```

Testar a função do Lambda

Nesta etapa, você invoca a função do Lambda manualmente usando o comando `invoke`. Quando a função do Lambda é executada, ela executa a consulta SELECT para a tabela Funcionários na instância MySQL do RDS e imprime os resultados, que também vão para o CloudWatch Logs.

1. invoque a função do Lambda com o comando `invoke`.

```
aws lambda invoke --function-name CreateTableAddRecordsAndRead output.txt
```

2. Verifique se a função do Lambda foi executada com êxito, da seguinte forma:

- Analise o arquivo `output.txt`.
- Analisar os resultados no console do AWS Lambda.
- Verifique os resultados no CloudWatch Logs.

Agora que você criou uma função do Lambda que acessa um banco de dados na sua VPC, a função já poderá ser invocada em resposta a eventos. Para obter informações sobre como configurar fontes de eventos e exemplos, consulte [Usar o AWS Lambda com outros serviços \(p. 277\)](#).

Limpar recursos da

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua conta da AWS.

Como excluir a função do Lambda

1. Abra a página [Functions](#) (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Actions, Delete.
4. Escolha Delete.

Para excluir a função de execução

1. Abra a página [Roles](#) (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Selecione Delete role (Excluir função).
4. Escolha Sim, excluir.

Para excluir a instância de banco de dados MySQL do

1. Abrir a [Página Bancos de dados](#) do console do Amazon RDS.
2. Selecione o banco de dados que você criou.
3. Escolha Actions, Delete.
4. Desmarque a caixa de seleção Create final snapshot (Criar snapshot final).
5. Digite **delete me** na caixa de texto.
6. Escolha Delete.

Configurar a função

A seção a seguir mostra configurações e tópicos adicionais que recomendamos como parte deste tutorial.

- Se muitas instâncias de função forem executadas simultaneamente, uma ou mais instâncias poderão falhar ao obter uma conexão de banco de dados. É possível usar a simultaneidade reservada para limitar a simultaneidade máxima da função. Defina a simultaneidade reservada como menor que o número de conexões de banco de dados. A simultaneidade reservada também reserva essas instâncias

para essa função, o que pode não ser ideal. Se estiver invocando as funções do Lambda pelo aplicativo, recomendamos que você escreva um código que limita o número de instâncias simultâneas. Para obter mais informações, consulte [Gerenciar a simultaneidade de uma função do Lambda](#).

- Para obter mais informações sobre como configurar um banco de dados do Amazon RDS para enviar notificações, consulte [Usar notificações de eventos do Amazon RDS](#).
- Para obter mais informações sobre o uso do Amazon SNS como acionador, consulte [Usar o Lambda com o Amazon SNS \(p. 457\)](#).

Usar o AWS Lambda com o Amazon S3

Você pode usar o Lambda para processar [notificações de eventos](#) do Amazon Simple Storage Service. O Amazon S3 pode enviar um evento para uma função do Lambda quando um objeto é criado ou excluído. Você define as configurações de notificação em um bucket e concede permissão ao Amazon S3 para invocar uma função na política de permissões baseada em recursos da função.

Warning

Se a função do Lambda usar o mesmo bucket que a ação, isso poderá fazer a função ser executada em um loop. Por exemplo, se o bucket disparar uma função sempre que um objeto for carregado e a função fizer upload de um objeto no bucket, a função vai se disparar indiretamente. Para evitar isso, use dois buckets ou configure o trigger para só se aplicar a um prefixo usado em objetos recebidos.

O Amazon S3 invoca a função [de forma assíncrona \(p. 161\)](#) com um evento que contém detalhes sobre o objeto. O exemplo a seguir mostra um evento que o Amazon S3 enviou quando um pacote de implantação foi carregado no Amazon S3.

Example Evento de notificação do 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":
        "v1R7PnpV2Ce8110PRw6jlUpck7Jo5ZsQjryTjKlc5aLWGVHPZLj5NeC6qMa0emYBDXOo6QBU0Wo="
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "828aa6fc-f7b5-4305-8584-487c791949c1",
        "bucket": {
          "name": "lambda-artifacts-deafc19498e3f2df",
          "ownerIdentity": {
            "principalId": "A3I5XTEXAMA13E"
          },
          "arn": "arn:aws:s3:::lambda-artifacts-deafc19498e3f2df"
        },
        "object": {
          "key": "b21b84d653bb07b05b1e6b33684dc11b",
          "size": 1305107,
          "eTag": "b21b84d653bb07b05b1e6b33684dc11b",
          "sequencer": "0C0F6F405D6ED209E1"
        }
      }
    }
  ]
}
```

Para invocar a função, o Amazon S3 precisa de permissão da [política baseada em recursos \(p. 62\)](#) da função. Quando você configura um acionador do Amazon S3 no console do Lambda, o console modifica a política baseada em recursos para permitir que o Amazon S3 invoque a função se o nome do bucket e o ID da conta corresponderem. Se você configurar a notificação no Amazon S3, use a API do Lambda para atualizar a política. Também é possível usar a API do Lambda para conceder permissão a outra conta ou restringir a permissão a um alias designado.

Se a sua função usa o AWS SDK para gerenciar recursos do Amazon S3, ela também precisa de permissões do Amazon S3 em sua [função de execução \(p. 57\)](#).

Tópicos

- [Tutorial: Usar um acionador do Amazon S3 para invocar uma função do Lambda \(p. 432\)](#)
- [Tutorial: Usar um acionador do Amazon S3 para criar imagens em miniatura \(p. 437\)](#)
- [Modelo do AWS SAM para uma aplicação do Amazon S3 \(p. 450\)](#)

Tutorial: Usar um acionador do Amazon S3 para invocar uma função do Lambda

Neste tutorial, você usará o console para criar uma função do Lambda e configurar um acionador para o Amazon Simple Storage Service (Amazon S3). O acionador invoca a sua função toda vez que você adiciona um objeto ao bucket do Amazon S3.

Recomendamos que você conclua este tutorial baseado em console antes de tentar o [tutorial para criar imagens em miniatura \(p. 437\)](#).

Prerequisites

Para usar o Lambda e outros serviços da AWS, você precisa de uma conta da AWS. Se você não tiver uma conta, visite aws.amazon.com e escolha Create an AWS account (Criar uma conta da AWS). Para saber como, consulte [Como faço para criar e ativar uma nova conta da AWS?](#)

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em [Conceitos básicos do Lambda \(p. 9\)](#) para criar sua primeira função do Lambda.

Crie um bucket e faça upload de um objeto de exemplo

Crie um bucket do Amazon S3 e faça upload de um arquivo de teste para o novo bucket. A sua função do Lambda recupera informações sobre esse arquivo quando você testa a função do console.

Para criar um bucket do Amazon S3 usando o console

1. Abra o [console do Amazon S3](#).
2. Selecione Create bucket (Criar bucket).
3. Em General configuration (Configuração geral), faça o seguinte:
 - a. Em Bucket name (Nome do bucket), insira um nome exclusivo.
 - b. Em AWS Region (Região da AWS), escolha uma região. Observe que você deve criar sua função do Lambda na mesma região.
4. Selecione Create bucket (Criar bucket).

Depois de criar o bucket, o Amazon S3 abre a página Buckets, que exibe uma lista de todos os buckets da sua conta na região atual.

Para fazer upload de um objeto de teste usando o console do Amazon S3

1. Na [página Buckets](#) do console do Amazon S3, escolha o nome do bucket que você criou.
2. Na guia Objects (Objetos), escolha Upload (Fazer upload).
3. Arraste um arquivo de teste da sua máquina local para a página Upload (Fazer upload).
4. Escolha Upload (Fazer upload).

Criar a função do Lambda

Use um [esquema de função \(p. 28\)](#) para criar a função do Lambda. Um esquema fornece uma função de exemplo que demonstra como usar o Lambda com outros produtos da AWS. Além disso, um esquema inclui código de exemplo e predefinições de configuração de função para um determinado tempo de execução. Para este tutorial, você pode escolher o esquema para o tempo de execução de Node.js ou Python.

Para criar uma função do Lambda de um esquema no console

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha Create function.
3. Na página Create function (Criar função), selecione Usar um blueprint (Usar um esquema).
4. Em Blueprints (Esquemas), insira **s3** na caixa de pesquisa.
5. Nos resultados da pesquisa, siga um destes procedimentos:
 - Para uma função do Node.js, escolha s3-get-object.
 - Para uma função do Python, escolha s3-get-object-python.
6. Selecione Configurar.
7. Em Basic information (Informações básicas), faça o seguinte:
 - a. Em Function name (Nome da função), insira **my-s3-function**.
 - b. Em Execution Role (Função de execução), selecione Create a new role from AWS policy templates (Criar uma nova função de modelos de política da AWS).
 - c. Em Role name (Nome da função), insira **my-s3-function-role**.
8. Em S3 trigger (Trigger do S3), escolha o bucket do S3 que você criou anteriormente.

Quando você configura um acionador do S3 usando o console do Lambda, o console modifica a [política baseada em recursos \(p. 62\)](#) da sua função para permitir que o Amazon S3 invoque a função.

9. Escolha Create function.

Revise o código de função

A função do Lambda recupera o nome do bucket do S3 de origem e o nome da chave do objeto carregada do parâmetro de evento que recebe. A função usa a API `getObject` do Amazon S3 para recuperar o tipo de conteúdo do objeto.

Ao visualizar a sua função no [console do Lambda](#), você pode revisar o código de função na guia Code (Código), em Code source (Código-fonte). A URL tem a seguinte aparência:

Node.js

Example index.js

```
console.log('Loading function');
```

```
const aws = require('aws-sdk');

const s3 = new aws.S3({ apiVersion: '2006-03-01' });

exports.handler = async (event, context) => {
    //console.log('Received event:', JSON.stringify(event, null, 2));

    // 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.getObject(params).promise();
        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);
    }
};
```

Python

Example lambda-function.py

```
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
```

Teste no console

Invoque a função do Lambda manualmente usando dados de exemplo de eventos do Amazon S3.

Para testar a função do Lambda usando o console.

1. Na guia Code (Código), em Code source (Código-fonte), escolha a seta ao lado de Test (Testar) e escolha Configure test events (Configurar eventos de teste) na lista suspensa.
2. Na janela Configure test event (Configurar evento de teste), faça o seguinte:
 - a. Selecione Create new test event (Criar evento de teste).
 - b. Em Event model (Modelo de evento), escolha Amazon S3 Put (s3-put).
 - c. Em Event name (Nome do evento), insira um nome para o evento de teste. Por exemplo, **mys3testevent**.
 - d. No evento de teste JSON, substitua o nome do bucket do S3 (`example-bucket`) e a chave de objeto (`test/key`) pelo nome do bucket e o nome do arquivo de teste. O seu evento de teste deve ter a seguinte aparência:

```
{  
    "Records": [  
        {  
            "eventVersion": "2.0",  
            "eventSource": "aws:s3",  
            "awsRegion": "us-west-2",  
            "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": "my-s3-bucket",  
                    "ownerIdentity": {  
                        "principalId": "EXAMPLE"  
                    },  
                    "arn": "arn:aws:s3:::example-bucket"  
                },  
                "object": {  
                    "key": "HappyFace.jpg",  
                    "size": 1024,  
                    "eTag": "0123456789abcdef0123456789abcdef",  
                    "sequencer": "0A1B2C3D4E5F678901"  
                }  
            }  
        }  
    ]  
}
```

- e. Escolha Create (Criar).
3. Para invocar a função com seu evento de teste, em Code source (Código-fonte), escolha Test (Testar).

A guia Execution results (Resultados da execução) exibe a resposta, os logs de função e o ID da solicitação, semelhante ao seguinte:

```
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: 'my-s3-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

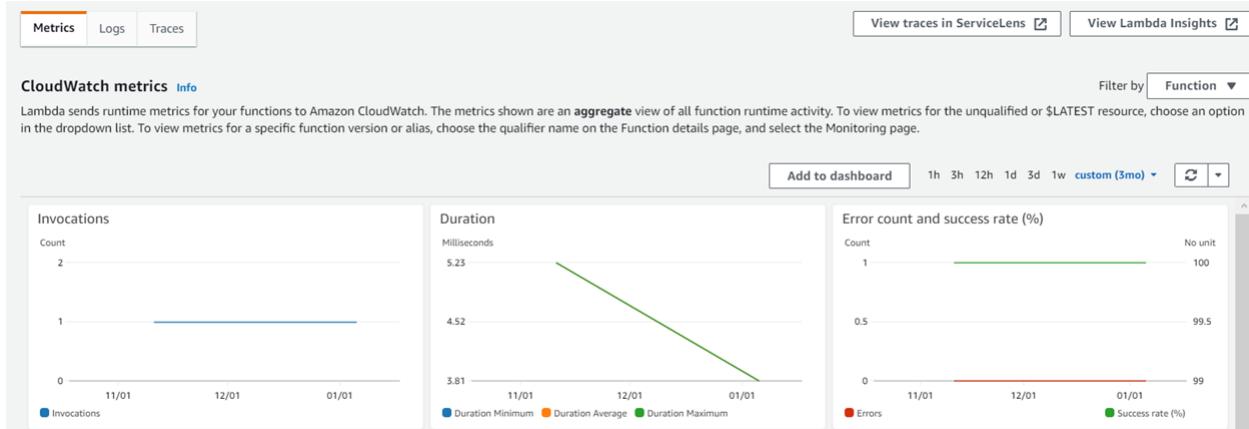
Request ID
12b3cae7-5f4e-415e-93e6-416b8f8b66e6
```

Teste com o trigger do S3

Invoque a sua função ao fazer upload de um arquivo para o bucket de origem do Amazon S3.

Para testar a função do Lambda usando o acionador do S3

1. Na [página Buckets](#) do console do Amazon S3, escolha o nome do bucket de origem que você criou anteriormente.
2. Na página Upload (Fazer upload), faça upload de alguns arquivos de imagem em .jpg ou .png para o bucket.
3. Abra a [página Functions \(Funções\)](#) no console do Lambda.
4. Escolha o nome da sua função (my-s3-function).
5. Para verificar se a função foi executada uma vez para cada arquivo que você enviou por upload, escolha a guia Monitor (Monitorar). Esta página mostra gráficos relacionados às métricas que o Lambda envia ao CloudWatch. A contagem no gráfico Invocações deve corresponder ao número de arquivos que você enviou por upload ao bucket do Amazon S3.



Para obter mais informações sobre esses gráficos, consulte [Funções de monitoramento no console do AWS Lambda \(p. 708\)](#).

6. (Opcional) Para visualizar os logs no console do CloudWatch, escolha [View logs in CloudWatch](#) (Visualizar logs no CloudWatch). Escolha um fluxo de log para ver a saída de logs para uma das invocações de função.

Limpar recursos da

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua conta da AWS.

Como excluir a função do Lambda

1. Abra a página [Functions](#) (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Actions, Delete.
4. Escolha Delete.

Para excluir a política do IAM

1. Abra a página [Policies](#) (Políticas) do console do AWS Identity and Access Management (IAM).
2. Selecione a política que o Lambda criou para você. O nome da política começa com AWSLAMBDAExecutionRole-.
3. Escolha Policy actions (Ações de política) e Delete (Excluir).
4. Escolha Delete.

Para excluir a função de execução

1. Abra a página [Roles](#) (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Selecione Delete role (Excluir função).
4. Escolha Sim, excluir.

Para excluir o bucket do S3

1. Abra o [console do Amazon S3](#).
2. Selecione o bucket que você criou.
3. Escolha Delete.
4. Insira o nome do bucket na caixa de texto.
5. Selecione a opção Confirmar.

Próximas etapas

Experimente o tutorial mais avançado. Neste tutorial, o trigger do S3 invocará uma função para [criar uma imagem em miniatura](#) (p. 437) para cada arquivo de imagem que é carregado no seu bucket do S3. Este tutorial requer um nível moderado de conhecimento de domínios da AWS e do Lambda. Use a AWS Command Line Interface (AWS CLI) para criar recursos e criar um pacote de implantação de arquivamento de arquivo.zip para a sua função e suas dependências.

Tutorial: Usar um acionador do Amazon S3 para criar imagens em miniatura

Neste tutorial, você criará uma função do Lambda e configurará um acionador para o Amazon Simple Storage Service (Amazon S3). O Amazon S3 invoca `CreateThumbnail` para cada arquivo de imagem

que é enviado por upload a um bucket do S3. A função lê o objeto de imagem do bucket do S3 de origem e cria uma imagem em miniatura para salvar em um bucket do S3 de destino.

Note

Este tutorial requer um nível moderado de conhecimento de domínios da AWS e do Lambda.

Recomendamos que você tente primeiro o [Tutorial: Usar um acionador do Amazon S3 para invocar uma função do Lambda \(p. 432\)](#).

Neste tutorial, você usará a AWS Command Line Interface (AWS CLI) para criar os recursos da AWS a seguir:

Recursos do Lambda

- Uma função Lambda. Você pode escolher Node.js, Python ou Java para o código de função.
- Um pacote de implantação de arquivamento de arquivo .zip para a função.
- Uma política de acesso que concede permissões do Amazon S3 para invocar a função.

Recursos do AWS Identity and Access Management (IAM)

- Uma função de execução com uma política de permissões associada para conceder permissões necessárias para a sua função.

Recursos do Amazon S3

- Um bucket de origem do S3 com uma configuração de notificação que invoca a função.
- Um bucket de destino do S3 em que a função salva imagens redimensionadas.

Tópicos

- [Prerequisites \(p. 438\)](#)
- [Etapa 1. Crie buckets do S3 e faça upload de um objeto de exemplo \(p. 439\)](#)
- [Etapa 2. Criar a política do IAM \(p. 439\)](#)
- [Etapa 3. Criar a função de execução \(p. 440\)](#)
- [Etapa 4. Criar o código da função \(p. 440\)](#)
- [Etapa 5. Criar o pacote de implantação \(p. 445\)](#)
- [Etapa 6. Criar a função do Lambda \(p. 446\)](#)
- [Etapa 7. Testar a função do Lambda \(p. 447\)](#)
- [Etapa 8 Configurar o Amazon S3 para publicar eventos \(p. 448\)](#)
- [Etapa 9. Teste usando o trigger do S3 \(p. 449\)](#)
- [Etapa 10. Limpar recursos da \(p. 449\)](#)

Prerequisites

- AWS account

Para usar o Lambda e outros serviços da AWS, você precisa de uma conta da AWS. Se você não tiver uma conta, visite aws.amazon.com e escolha Create an AWS account (Criar uma conta da AWS). Para saber como, consulte [Como faço para criar e ativar uma nova conta da AWS?](#)

- Linha de comando

Para concluir as etapas a seguir, você precisa de um terminal de linha de comando ou de um shell para executar os comandos. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

Você deve ver a saída a seguir:

```
aws-cli/2.0.57 Python/3.7.4 Darwin/19.6.0 exe/x86_64
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

- AWS CLI

Neste tutorial, você usa comandos da AWS CLI para criar e invocar a função do Lambda. [Instale a AWS CLI e configure-a com suas credenciais da AWS](#).

- Ferramentas de idiomas

Instale as ferramentas de suporte a idiomas e um gerenciador de pacotes para o idioma que você deseja usar: Node.js, Python ou Java. Para ver as ferramentas sugeridas, consulte [Ferramentas de criação de código \(p. 6\)](#).

Etapa 1. Crie buckets do S3 e faça upload de um objeto de exemplo

Siga as etapas para criar buckets do S3 e fazer upload de um objeto.

1. Abra o [console do Amazon S3](#).
2. [Crie dois buckets do S3](#). O bucket de destino deve ser nomeado para **source-resized**, no qual **origem** é o nome do bucket de origem. Por exemplo, um bucket de origem com o nome `mybucket` e um bucket de destino com o nome `mybucket-resized`.
3. No bucket de origem, [faça upload](#) de um objeto em .jpg, por exemplo, `HappyFace.jpg`.

Você deve criar esse objeto de exemplo antes de testar a sua função do Lambda. Ao testar a função manualmente usando o comando invoke do Lambda, você transfere dados do objeto de exemplo para a função que especifica o nome do bucket de origem e `HappyFace.jpg` como o objeto recém-criado.

Etapa 2. Criar a política do IAM

Crie uma política do IAM que defina as permissões para a função do Lambda. A função deve ter permissões para:

- Obter o objeto do bucket do S3 de origem.
- Colocar o objeto redimensionado no bucket do S3 de destino.
- Gravar logs de acesso no Amazon CloudWatch Logs

Para criar uma política do IAM

1. Abra a [página Policies \(Políticas\)](#) no console do IAM.
2. Escolha Create policy (Criar política).
3. Selecione a guia JSON e, em seguida, cole a política a seguir. Certifique-se de substituir `mybucket` pelo nome do bucket de origem que você criou anteriormente.

```
{  
    "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:::mybucket/*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutObject"  
            ],  
            "Resource": "arn:aws:s3:::mybucket-resized/*"  
        }  
    ]  
}
```

4. Escolha Next: Tags (Próximo: tags).
5. Selecione Next: Review.
6. No campo Review policy (Política de revisão), em Name (Nome), insira **AWSLambdaS3Policy**.
7. Escolha Create policy (Criar política).

Etapa 3. Criar a função de execução

Crie a [função de execução \(p. 57\)](#) que dá à sua função do Lambda permissão para acessar recursos da AWS.

Para criar uma função de execução

1. Abra a página [Roles \(Funções\)](#) no console do IAM.
2. Selecione Create role.
3. Crie uma função com as seguintes propriedades:
 - Entidade confiável – Lambda
 - Permissions policy (Política de permissões): **AWSLambdaS3Policy**
 - Nome da função – **lambda-s3-role**

Etapa 4. Criar o código da função

Nos exemplos de código a seguir, o evento do Amazon S3 contém o nome do bucket do S3 de origem e o nome da chave de objeto. Se o objeto for um arquivo de imagem de .jpg ou .png, ele lê a imagem do bucket de origem, gera uma imagem em miniatura e salva a miniatura no bucket do S3 de destino.

Observe o seguinte:

- O código assume que o bucket de destino existe e que seu nome é uma concatenação do nome do bucket de origem e `-resized`.
- Para cada arquivo de miniatura criado, o código da função do Lambda deriva o nome da chave de objeto como uma concatenação de `resized-` e o nome da chave do objeto de origem. Por exemplo, se a chave do objeto de origem `sample.jpg`, o código cria um objeto em miniatura com a chave `resized-sample.jpg`.

Node.js

Copie o exemplo de código a seguir em um arquivo com o nome `index.js`.

Example index.js

```
// dependencies
const AWS = require('aws-sdk');
const util = require('util');
const sharp = require('sharp');

// get reference to S3 client
const s3 = new AWS.S3();

exports.handler = async (event, context, callback) => {

    // Read options from the event parameter.
    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;
    }

    // Download the image from the S3 source bucket.

    try {
        const params = {
            Bucket: srcBucket,
            Key: srcKey
        };
        var origimage = await s3.getObject(params).promise();

    } 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 buffer = await sharp(origimage.Body).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: buffer,
        ContentType: "image"
    };

    const putResult = await s3.putObject(destparams).promise();

} catch (error) {
    console.log(error);
    return;
}

console.log('Successfully resized ' + srcBucket + '/' + srcKey +
    ' and uploaded to ' + dstBucket + '/' + dstKey);
};
```

Python

Copie o exemplo de código a seguir em um arquivo com o nome `lambda_function.py`.

Example `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), key)
```

Java

O código Java implementa a interface `RequestHandler` fornecida na biblioteca do `aws-lambda-java-core`. Ao criar uma função do Lambda, você especifica a classe como o manipulador (neste exemplo de código, `example.handler`). Para obter mais informações sobre o uso de interfaces para fornecer um manipulador, consulte [Interfaces do manipulador \(p. 597\)](#).

O manipulador usa `S3Event` como o tipo de entrada, que fornece métodos convenientes para o seu código de função ler informações do evento de entrada do Amazon S3. O Amazon S3 invoca sua função do Lambda de forma assíncrona. Como você está implementando uma interface que exige a especificação de um tipo de retorno, o manipulador usa `String` como o tipo de retorno.

Copie o exemplo de código a seguir em um arquivo com o nome `Handler.java`.

Example Handler.java

```
package example;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.imageio.ImageIO;

import com.amazonaws.AmazonServiceException;
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.s3.AmazonS3;
import com.amazonaws.services.s3.event.S3EventNotification.S3EventNotificationRecord;
import com.amazonaws.services.s3.model.GetObjectRequest;
import com.amazonaws.services.s3.model.ObjectMetadata;
import com.amazonaws.services.s3.model.S3Object;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;

public class Handler implements
    RequestHandler<S3Event, String> {
    private static final float MAX_WIDTH = 100;
    private static final float MAX_HEIGHT = 100;
    private final String JPG_TYPE = (String) "jpg";
    private final String JPG_MIME = (String) "image/jpeg";
    private final String PNG_TYPE = (String) "png";
    private final String PNG_MIME = (String) "image/png";

    public String handleRequest(S3Event s3event, Context context) {
        try {
            S3EventNotificationRecord record = s3event.getRecords().get(0);

            String srcBucket = record.getS3().getBucket().getName();

            // Object key may have spaces or unicode non-ASCII characters.
            String srcKey = record.getS3().getObject().getUrlDecodedKey();
```

```
String dstBucket = srcBucket + "-resized";
String dstKey = "resized-" + srcKey;

// Sanity check: validate that source and destination are different
// buckets.
if (srcBucket.equals(dstBucket)) {
    System.out
        .println("Destination bucket must not match source bucket.");
    return "";
}

// Infer the image type.
Matcher matcher = Pattern.compile(".*\\\\.([^\\\\.]*)").matcher(srcKey);
if (!matcher.matches()) {
    System.out.println("Unable to infer image type for key "
        + srcKey);
    return "";
}
String imageType = matcher.group(1);
if (!(JPG_TYPE.equals(imageType)) && !(PNG_TYPE.equals(imageType))) {
    System.out.println("Skipping non-image " + srcKey);
    return "";
}

// Download the image from S3 into a stream
AmazonS3 s3Client = AmazonS3ClientBuilder.defaultClient();
S3Object s3Object = s3Client.getObject(new GetObjectRequest(
    srcBucket, srcKey));
InputStream objectData = s3Object.getObjectContent();

// Read the source image
BufferedImage srcImage = ImageIO.read(objectData);
int srcHeight = srcImage.getHeight();
int srcWidth = srcImage.getWidth();
// Infer the scaling factor to avoid stretching the image
// unnaturally
float scalingFactor = Math.min(MAX_WIDTH / srcWidth, MAX_HEIGHT
    / srcHeight);
int width = (int) (scalingFactor * srcWidth);
int height = (int) (scalingFactor * srcHeight);

BufferedImage resizedImage = new BufferedImage(width, height,
    BufferedImage.TYPE_INT_RGB);
Graphics2D g = resizedImage.createGraphics();
// Fill with white before applying semi-transparent (alpha) images
g.setPaint(Color.white);
g.fillRect(0, 0, width, height);
// Simple bilinear resize
g.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
    RenderingHints.VALUE_INTERPOLATION_BILINEAR);
g.drawImage(srcImage, 0, 0, width, height, null);
g.dispose();

// Re-encode image to target format
ByteArrayOutputStream os = new ByteArrayOutputStream();
ImageIO.write(resizedImage, imageType, os);
InputStream is = new ByteArrayInputStream(os.toByteArray());
// Set Content-Length and Content-Type
ObjectMetadata meta = new ObjectMetadata();
meta.setContentLength(os.size());
if (JPG_TYPE.equals(imageType)) {
    meta.setContentType(JPG_MIME);
}
if (PNG_TYPE.equals(imageType)) {
    meta.setContentType(PNG_MIME);
}
```

```
// Uploading to S3 destination bucket
System.out.println("Writing to: " + dstBucket + "/" + dstKey);
try {
    s3Client.putObject(dstBucket, dstKey, is, meta);
}
catch(AmazonServiceException e)
{
    System.err.println(e.getErrorMessage());
    System.exit(1);
}
System.out.println("Successfully resized " + srcBucket + "/"
+ srcKey + " and uploaded to " + dstBucket + "/" + dstKey);
return "Ok";
} catch (IOException e) {
    throw new RuntimeException(e);
}
}
```

Etapa 5. Criar o pacote de implantação

O pacote de implantação é um [arquivo .zip \(p. 37\)](#) que contém o seu código da função do Lambda e suas dependências.

Node.js

A função de exemplo deve incluir o módulo do Sharp no pacote de implantação.

Como criar um pacote de implantação

1. Abra um shell ou um terminal da linha de comando em um ambiente Linux. Verifique se a versão do Node.js no ambiente local corresponde à versão do Node.js da função.
2. Salve o código da função como `index.js` em um diretório chamado `lambda-s3`.
3. Instale a biblioteca do Sharp com `npm`. Para o Linux, use o comando a seguir:

```
npm install sharp
```

Após esta etapa, você terá a seguinte estrutura de diretórios:

```
lambda-s3
|- index.js
|- /node_modules/sharp
# /node_modules/...
```

4. Crie um pacote de implantação com o código da função e suas dependências. Defina a opção `-r` (recursiva) para o comando `zip` para comprimir as subpastas.

```
zip -r function.zip .
```

Python

Dependencies

- [Pillow](#)

Como criar um pacote de implantação

- Recomendamos usar o comando [sam build](#) da CLI do AWS SAM com a opção `--use-container` para criar pacotes de implantação que contenham bibliotecas escritas em C ou C++, como a biblioteca [Pillow \(PIL\)](#).

Java

Dependencies

- `aws-lambda-java-core`
- `aws-lambda-java-events`
- `aws-java-sdk`

Como criar um pacote de implantação

- Crie o código com as dependências da biblioteca do Lambda para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Java com arquivos .zip ou JAR \(p. 599\)](#).

Etapa 6. Criar a função do Lambda

Para criar a função

- Crie uma função do Lambda com o comando `create-function`.

Node.js

```
aws lambda create-function --function-name CreateThumbnail \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs12.x \
--timeout 10 --memory-size 1024 \
--role arn:aws:iam::123456789012:role/lambda-s3-role
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

O comando `create-function` especifica o manipulador de funções como `index.handler`. Esse nome reflete o nome da função como `handler` e o nome do arquivo em que o código do manipulador é armazenado como `index.js`. Para obter mais informações, consulte [AWS LambdaManipulador da função do em Node.js \(p. 514\)](#). O comando especifica um tempo de execução de `nodejs12.x`. Para obter mais informações, consulte [Tempos de execução do Lambda \(p. 214\)](#).

Python

```
aws lambda create-function --function-name CreateThumbnail \
--zip-file fileb://function.zip --handler lambda_function.lambda_handler --runtime
python3.8 \
--timeout 10 --memory-size 1024 \
--role arn:aws:iam::123456789012:role/lambda-s3-role
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

O comando `create-function` especifica o manipulador de funções como `lambda_function.lambda_handler`. Esse nome reflete o nome da função como

`lambda_handler` e o nome do arquivo em que o código do manipulador é armazenado como `lambda_function.py`. Para obter mais informações, consulte [Manipulador de função do Lambda em Python \(p. 540\)](#). O comando especifica um tempo de execução de `python3.8`. Para obter mais informações, consulte [Tempos de execução do Lambda \(p. 214\)](#).

Java

```
aws lambda create-function --function-name CreateThumbnail \
--zip-file fileb://function.zip --handler example.handler --runtime java11 \
--timeout 10 --memory-size 1024 \
--role arn:aws:iam::123456789012:role/lambda-s3-role
```

O comando cli-binary-format é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

O comando `create-function` especifica o manipulador de funções como `example.handler`. A função pode usar o formato de manipulador abreviado `package.Class` porque a mesma implementa uma interface de manipulador. Para obter mais informações, consulte [AWS LambdaManipulador de função do em Java \(p. 595\)](#). O comando especifica um tempo de execução de `java11`. Para obter mais informações, consulte [Tempos de execução do Lambda \(p. 214\)](#).

Para o parâmetro de função, substitua `123456789012` pelo seu [ID da conta da AWS](#). O exemplo de comando anterior especifica um valor de tempo limite de 10 segundos como a configuração da função. Dependendo do tamanho dos objetos que você fizer upload, pode ser necessário aumentar o valor do tempo limite usando o seguinte comando da AWS CLI:

```
aws lambda update-function-configuration --function-name CreateThumbnail --timeout 30
```

Etapa 7. Testar a função do Lambda

Invoque a função do Lambda manualmente usando dados de exemplo de eventos do Amazon S3.

Para testar a função do Lambda

1. Salve os dados de eventos de exemplo a seguir do Amazon S3 em um arquivo com o nome `inputFile.txt`. Certifique-se de substituir `sourcebucket` e `HappyFace.jpg` pelo nome do seu bucket de origem do S3 e uma chave de objeto em .jpg, respectivamente.

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-west-2",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "AIDAJDPLRKLG7UEXAMPLE"
      },
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "responseElements": {
        "x-amz-request-id": "C3D13FE58DE4C810",
        "x-amz-id-2": "FMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/JRWeUWerMUE5JgHvANOjpD"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "objectKey": "HappyFace.jpg",
        "objectSize": 1024,
        "bucketName": "sourcebucket",
        "bucketRegion": "us-west-2",
        "versionId": null
      }
    }
  ]
}
```

```
"configurationId": "testConfigRule",
"bucket": {
    "name": "sourcebucket",
    "ownerIdentity": {
        "principalId": "A3NL1KOZZKExample"
    },
    "arn": "arn:aws:s3:::sourcebucket"
},
"object": {
    "key": "HappyFace.jpg",
    "size": 1024,
    "eTag": "d41d8cd98f00b204e9800998ecf8427e",
    "versionId": "096fKKXTRTl3on89fV0.nfljtsv6qko"
}
}
```

2. Invoque a função com o comando `invoke` a seguir. Observe que o comando solicita a execução assíncrona (`--invocation-type Event`). Opcionalmente, você pode invocar a função de forma síncrona especificando `RequestResponse` como o valor de parâmetro `invocation-type`.

```
aws lambda invoke
--function-name CreateThumbnail \
--invocation-type Event \
--payload file://inputFile.txt outfile.txt
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

3. Verifique se a miniatura é criada no bucket de destino do S3.

Etapa 8 Configurar o Amazon S3 para publicar eventos

Conclua a configuração para que o Amazon S3 possa publicar eventos criados por objeto no Lambda e invocar a sua função do Lambda. Nesta etapa, faça o seguinte:

- Adicione permissões à política de acesso da função do Amazon S3 para invocar a função.
- Adicione uma configuração de notificação ao bucket de origem do S3. Na configuração de notificação, você fornece o seguinte:
 - O tipo de evento em que você deseja que o Amazon S3 publique eventos. Para este tutorial, especifique o tipo de evento `s3:ObjectCreated:*`, para que o Amazon S3 publique eventos quando objetos forem criados.
 - A função para invocar.

Como adicionar permissões à política de função

1. Execute o comando `add-permission` a seguir para conceder ao principal do serviço do Amazon S3 (`s3.amazonaws.com`) permissões para executar a ação `lambda:InvokeFunction`. Observe que a permissão será concedida ao Amazon S3 para invocar a função somente se as seguintes condições forem atendidas:
 - Um evento criado por objeto deve ser detectado em um bucket específico do S3.
 - O bucket do S3 deve ser de propriedade da sua conta da AWS. Se você excluir um bucket, é possível que outra conta da AWS crie um bucket com o mesmo nome de recurso da Amazon (ARN).

```
aws lambda add-permission --function-name CreateThumbnail --principal s3.amazonaws.com \
\--statement-id s3invoke --action "lambda:InvokeFunction" \
\--source-arn arn:aws:s3:::sourcebucket \
\--source-account account-id
```

2. Verifique a política de acesso da função executando o comando get-policy.

```
aws lambda get-policy --function-name CreateThumbnail
```

Para permitir que o Amazon S3 publique eventos criados por objeto no Lambda, adicione uma configuração de notificação no bucket de origem do S3.

Important

Este procedimento configura o bucket do S3 para invocar a função sempre que um objeto é criado no bucket. Certifique-se de configurar esta opção somente no bucket de origem. Não permita que a sua função crie objetos no bucket de origem. Isso pode fazer com que sua função seja [invocada continuamente em um ciclo \(p. 431\)](#).

Como configurar notificações

1. Abra o [console do Amazon S3](#).
2. Escolha o nome do bucket de origem do S3.
3. Escolha a guia Properties (Propriedades).
4. Em Event notifications (Notificações de eventos), escolha Create event notification (Criar notificação de evento) para configurar uma notificação com as seguintes configurações:
 - Event name (Nome do evento): **lambda-trigger**
 - Event types (Tipos de evento): **All object create events**
 - Destino – **Lambda function**
 - Lambda function (Função do Lambda): **CreateThumbnail**

Para obter mais informações sobre configuração de eventos, consulte [Habilitar e configurar notificações de eventos usando o console do Amazon S3](#) no Guia de usuário do Amazon Simple Storage Service Console.

Etapa 9. Teste usando o trigger do S3

Teste a configuração da seguinte maneira:

1. Faça upload dos objetos em .jpg ou .png no bucket de origem do S3 usando o [console do Amazon S3](#).
2. Verifique para cada objeto de imagem se uma miniatura é criada no bucket de destino do S3 usando a função **CreateThumbnail** do Lambda.
3. Visualize os logs no [console do CloudWatch](#).

Etapa 10. Limpar recursos da

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua conta da AWS.

Como excluir a função do Lambda

1. Abra a página [Functions](#) (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Actions, Delete.
4. Escolha Delete.

Para excluir a política que você criou

1. Abra a página [Policies](#) (Políticas) do console do IAM.
2. Selecione a política que você criou (AWSLambdaS3Policy).
3. Escolha Policy actions (Ações de política) e Delete (Excluir).
4. Escolha Delete.

Para excluir a função de execução

1. Abra a página [Roles](#) (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Selecione Delete role (Excluir função).
4. Escolha Sim, excluir.

Para excluir o bucket do S3

1. Abra o [console do Amazon S3](#).
2. Selecione o bucket que você criou.
3. Escolha Delete.
4. Insira o nome do bucket na caixa de texto.
5. Selecione a opção Confirmar.

Modelo do AWS SAM para uma aplicação do Amazon S3

Você pode criar esse aplicativo usando [AWS SAM](#). Para saber mais sobre como criar modelos do AWS SAM, consulte [Noções básicas de modelos do AWS SAM](#) no Guia do desenvolvedor do AWS Serverless Application Model.

Veja abaixo um modelo de exemplo do AWS SAM para a aplicação do Lambda do [tutorial \(p. 432\)](#). Copie o texto abaixo para um arquivo .yaml e salve-o ao lado do pacote ZIP criado previamente. Observe que os valores dos parâmetros Handler e Runtime devem corresponder àqueles usados quando você criou a função na seção anterior.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  CreateThumbnail:
    Type: AWS::Serverless::Function
    Properties:
      Handler: handler
      Runtime: runtime
```

```
Timeout: 60
Policies: AWSLambdaExecute
Events:
  CreateThumbnailEvent:
    Type: S3
    Properties:
      Bucket: !Ref SrcBucket
      Events: s3:ObjectCreated:*
SrcBucket:
  Type: AWS::S3::Bucket
```

Para obter informações sobre como empacotar e implantar o aplicativo sem servidor usando os comandos de empacotamento e implantação, consulte [Implantar aplicativos sem servidor](#) no Guia do desenvolvedor do AWS Serverless Application Model.

Usar o AWS Lambda com operações em lote do Amazon S3

É possível usar operações em lote do Amazon S3 para invocar uma função do Lambda em um grande conjunto de objetos do Amazon S3. O Amazon S3 rastreia o andamento das operações em lote, envia notificações e armazena um relatório de conclusão que mostra o status de cada ação.

Para executar uma operação em lote, crie um Amazon S3[trabalho de operações em lote](#). Ao criar o trabalho, forneça um manifesto (a lista de objetos) e configure a ação a ser executada nesses objetos.

Quando o trabalho em lote é iniciado, o Amazon S3 invoca a função do Lambda [de forma síncrona \(p. 158\)](#) para cada objeto no manifesto. O parâmetro do evento inclui os nomes do bucket e do objeto.

O exemplo a seguir mostra o evento que o Amazon S3 envia à função do Lambda para um objeto que é chamado `customerImage1.jpg` no bucket `examplebucket`.

Example Eventos de solicitação em lote do Amazon S3

```
{  
  "invocationSchemaVersion": "1.0",  
  "invocationId": "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",  
  "job": {  
    "id": "f3cc4f60-61f6-4a2b-8a21-d07600c373ce"  
  },  
  "tasks": [  
    {  
      "taskId": "dGFza2lkZ29lc2hlcUmUK",  
      "s3Key": "customerImage1.jpg",  
      "s3VersionId": "1",  
      "s3BucketArn": "arn:aws:s3:us-east-1:0123456788:examplebucket"  
    }  
  ]  
}
```

A função do Lambda deve retornar um objeto JSON com os campos, conforme mostrado no exemplo a seguir. É possível copiar o `invocationId` e `taskId` do parâmetro do evento. Você pode retornar uma string `noresultString`. O Amazon S3 salva `oresultString` no relatório de conclusão.

Example Resposta em lote do Amazon S3

```
{  
  "invocationSchemaVersion": "1.0",  
  "treatMissingKeysAs" : "PermanentFailure",  
  "invocationId" : "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",  
  "results": [  
    {  
      "taskId": "dGFza2lkZ29lc2hlcUmUK",  
      "resultCode": "Succeeded",  
      "resultString": "[\"Alice\", \"Bob\"]"  
    }  
  ]  
}
```

Invoker funções do Lambda de operações em lote do Amazon S3

É possível invocar a função do Lambda com um ARN de função qualificado ou não qualificado. Se você quiser usar a mesma versão de função para todo o trabalho em lote, configure uma versão de função específica no parâmetro `FunctionARN` ao criar o trabalho. Se você configurar um alias ou o qualificador `$LATEST`, o trabalho em lote começará imediatamente a chamar a nova versão da função se o alias ou `$LATEST` for atualizado durante a execução do trabalho.

Observe que não é possível reutilizar uma função baseada em evento do Amazon S3 para operações em lote. Isso ocorre, pois a operação em lote do Amazon S3 transmite um parâmetro de evento diferente para a função do Lambda e espera uma mensagem de retorno com uma estrutura JSON específica.

Na [política baseada em recursos \(p. 62\)](#) criada para o trabalho em lote do Amazon S3, verifique se você definiu a permissão para o trabalho invocar a função do Lambda.

Na [função de execução \(p. 57\)](#) da função, defina uma política de confiança para que o Amazon S3 assuma a função quando ele executar a função.

Se a função usar o AWS SDK para gerenciar recursos do Amazon S3, será necessário adicionar permissões do Amazon S3 na função de execução.

Quando o trabalho é executado, o Amazon S3 inicia várias instâncias de função para processar os objetos do Amazon S3 em paralelo, até o [Limite de simultaneidade \(p. 32\)](#) da função do. O Amazon S3 limita a aceleração de instâncias para evitar custo em excesso para trabalhos menores.

Se a função do Lambda retornar um código de resposta `TemporaryFailure`, o Amazon S3 tentará novamente realizar a operação.

Para obter mais informações sobre como gerenciar operações em lote do Amazon S3, consulte [Executar operações em lote](#) no Guia do desenvolvedor do Amazon S3.

Para obter um exemplo de como usar uma função do Lambda nas operações em lote do Amazon S3, consulte [Invoker uma função do Lambda de operações em lote do Amazon S3](#) no Guia do desenvolvedor do Amazon S3.

Usar o AWS Lambda com o Secrets Manager

O Secrets Manager usa uma função do Lambda para [alternar o segredo](#) de um serviço seguro ou para o banco de dados. É possível personalizar a função do Lambda para implementar os detalhes específicos do serviço de como executar a rotação de um segredo.

O Secrets Manager invoca a função de alternância do Lambda como uma invocação síncrona. O parâmetro event contém os seguintes campos:

```
{  
  "Step" : "request.type",  
  "SecretId" : "string",  
  "ClientRequestToken" : "string"  
}
```

Para obter mais informações sobre como usar o Lambda com Secrets Manager, consulte [Understanding Your Lambda rotation function](#).

Usar o AWS Lambda com o Amazon SES

Ao usar o Amazon SES para receber mensagens, é possível configurá-lo para chamar sua função do Lambda quando as mensagens chegarem. O serviço pode invocar sua função do Lambda passando o evento de e-mail recebido que, na realidade, é uma mensagem do Amazon SES em um evento do Amazon SNS, como um parâmetro.

Example Evento de mensagem do Amazon SES

```
{
  "Records": [
    {
      "eventVersion": "1.0",
      "ses": {
        "mail": {
          "commonHeaders": {
            "from": [
              "Jane Doe <janedoe@example.com>"
            ],
            "to": [
              "johndoe@example.com"
            ],
            "returnPath": "janedoe@example.com",
            "messageId": "<0123456789example.com>",
            "date": "Wed, 7 Oct 2015 12:34:56 -0700",
            "subject": "Test Subject"
          },
          "source": "janedoe@example.com",
          "timestamp": "1970-01-01T00:00:00.000Z",
          "destination": [
            "johndoe@example.com"
          ],
          "headers": [
            {
              "name": "Return-Path",
              "value": "<janedoe@example.com>"
            },
            {
              "name": "Received",
              "value": "from mailer.example.com (mailer.example.com [203.0.113.1]) by inbound-smtp.us-west-2.amazonaws.com with SMTP id o3vrnil0e2ic for johndoe@example.com; Wed, 07 Oct 2015 12:34:56 +0000 (UTC)"
            },
            {
              "name": "DKIM-Signature",
              "value": "v=1; a=rsa-sha256; c=relaxed/relaxed; d=example.com; s=example; h=mime-version:from:date:message-id:subject:to:content-type; bh=jX3F0bCAI7sIbkHyy3mLYO28ieDQz2R0P8HwQkk1Fj4=; b=sQwJ+LMe9RjkesGu+vqU56asvMhrLRRYrWCbV"
            },
            {
              "name": "MIME-Version",
              "value": "1.0"
            },
            {
              "name": "From",
              "value": "Jane Doe <janedoe@example.com>"
            },
            {
              "name": "Date",
              "value": "Wed, 7 Oct 2015 12:34:56 -0700"
            },
            {
              "name": "Message-ID",
              "value": "<0123456789example.com>@"
            }
          ]
        }
      }
    }
  ]
}
```

```
        "value": "<0123456789example.com>"  
    },  
    {  
        "name": "Subject",  
        "value": "Test Subject"  
    },  
    {  
        "name": "To",  
        "value": "johndoe@example.com"  
    },  
    {  
        "name": "Content-Type",  
        "value": "text/plain; charset=UTF-8"  
    }  
],  
"headersTruncated": false,  
"messageId": "o3vrnil0e2ic28tr"  
},  
"receipt": {  
    "recipients": [  
        "johndoe@example.com"  
    ],  
    "timestamp": "1970-01-01T00:00:00.000Z",  
    "spamVerdict": {  
        "status": "PASS"  
    },  
    "dkimVerdict": {  
        "status": "PASS"  
    },  
    "processingTimeMillis": 574,  
    "action": {  
        "type": "Lambda",  
        "invocationType": "Event",  
        "functionArn": "arn:aws:lambda:us-west-2:012345678912:function:Example"  
    },  
    "spfVerdict": {  
        "status": "PASS"  
    },  
    "virusVerdict": {  
        "status": "PASS"  
    }  
},  
"eventSource": "aws:ses"  
}  
]  
}
```

Para obter informações, consulte [Ação do Lambda](#) no Guia do desenvolvedor do Amazon SES.

Usar o Lambda com o Amazon SNS

Você pode usar uma função do Lambda para processar notificações do Amazon Simple Notification Service (Amazon SNS). O Amazon SNS oferece suporte às funções do Lambda como destino para mensagens enviadas para um tópico. É possível inscrever a sua função em tópicos na mesma conta ou em outras contas da AWS.

O Amazon SNS invoca a função [de forma assíncrona \(p. 161\)](#) com um evento que contém uma mensagem e metadados.

Example Evento da mensagem do Amazon SNS

```
{  
  "Records": [  
    {  
      "EventVersion": "1.0",  
      "EventSubscriptionArn": "arn:aws:sns:us-east-2:123456789012: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-2.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-2.amazonaws.com/?Action=Unsubscribe&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": "TestInvoke"  
      }  
    }  
  ]  
}
```

Para invocação assíncrona, o Lambda enfileira a mensagem e processa novas tentativas. Se o Amazon SNS não puder acessar o Lambda ou se a mensagem for rejeitada, o Amazon SNS tentará novamente em intervalos crescentes ao longo de várias horas. Para obter detalhes, consulte [Confiabilidade](#) nas Perguntas frequentes do Amazon SNS.

Para executar entregas do Amazon SNS para o Lambda entre contas para o, você deve autorizar o Amazon SNS para invocar a sua função do Lambda. Por sua vez, o Amazon SNS deve permitir que o AWSCom a função do Lambda para assinar o tópico do Amazon SNS. Por exemplo, se o tópico do Amazon SNS estiver em uma conta A e a função do Lambda estiver em uma conta B, as duas contas deverão conceder permissões uma para a outra para acesso a seus respectivos recursos. Como nem todas as opções para configurar permissões entre contas estão disponíveis no AWS Management Console, você deve usar a AWS Command Line Interface (AWS CLI) para a configuração.

Para obter mais informações, consulte [Invocar funções do Lambda](#) no Guia do desenvolvedor do Amazon Simple Notification Service.

Tipos de entrada para eventos do Amazon SNS

Para exemplos de tipo de entrada para eventos do Amazon SNS em Java, .NET e Go, consulte o seguinte no repositório do GitHub da AWS:

- [SNSEvent.java](#)
- [SNSEvent.cs](#)
- [sns.go](#)

Tópicos

- [Como usar oAWS LambdaCom o Amazon Simple Notification Service \(p. 458\)](#)
- [Código de exemplo da função do \(p. 462\)](#)

Como usar oAWS LambdaCom o Amazon Simple Notification Service

Você pode usar uma função do Lambda em uma conta da AWS para assinar um tópico do Amazon SNS em uma conta separada da AWS. Neste tutorial, você usa o AWS Command Line Interface para executar operações do AWS Lambda, como criar uma função do Lambda, criar um tópico do Amazon SNS e conceder permissões para permitir que esses dois recursos acessem um ao outro.

Prerequisites

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em [Conceitos básicos do Lambda \(p. 9\)](#) para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, você precisa de um terminal de linha de comando ou de um shell para executar os comandos. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

Você deve ver a saída a seguir:

```
aws-cli/2.0.57 Python/3.7.4 Darwin/19.6.0 exe/x86_64
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

No tutorial, você usa duas contas. Os comandos da AWS CLI ilustram isso usando dois [perfis nomeados](#), cada um configurado para uso com uma conta diferente. Se você usar perfis com nomes diferentes, ou o perfil padrão e um perfil nomeado, modifique os comandos conforme necessário.

Criar um tópico do Amazon SNS

Na conta A (01234567891A), crie o tópico do Amazon SNS de origem.

```
aws sns create-topic --name sns-topic-for-lambda --profile accountA
```

Anote o ARN do tópico retornado pelo comando. Você precisará dele ao adicionar permissões à função do Lambda para assinar o tópico.

Criar a função de execução

Na conta B (01234567891B), crie a [função de execução \(p. 57\)](#) que concede à sua função permissão para acessar recursos da AWS.

Para criar uma função de execução

1. Abra a [página Roles](#) (Funções) no console do IAM.
2. Selecione Create role.
3. Crie uma função com as seguintes propriedades.
 - Trusted entity (Entidade confiável – AWS Lambda).
 - Permissions (Permissões): AWSLambdaBasicExecutionRole.
 - Nome da função – **lambda-sns-role**.

A política AWSLambdaBasicExecutionRole tem as permissões necessárias para a função gravar logs no CloudWatch Logs.

Criar uma função do Lambda

Na conta B (01234567891B), crie a função que processa os eventos do Amazon SNS. O código de amostra a seguir recebe uma entrada de evento do Amazon SNS e processa as mensagens que ela contém. Na ilustração, o código grava alguns dos dados de entrada no CloudWatch Logs.

Note

Para o código de amostra em outras linguagens, consulte [Código de exemplo da função do \(p. 462\)](#).

Example index.js

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
// console.log('Received event:', JSON.stringify(event, null, 4));

    var message = event.Records[0].Sns.Message;
    console.log('Message received from SNS:', message);
    callback(null, "Success");
};
```

Para criar a função

1. Copie o código de amostra em um arquivo chamado `index.js`.
2. Crie um pacote de implantação.

```
zip function.zip index.js
```

3. Crie uma função do Lambda com o comando `create-function`.

```
aws lambda create-function --function-name Function-With-SNS \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs12.x \
--role arn:aws:iam::01234567891B:role/service-role/lambda-sns-execution-role \
--timeout 60 --profile accountB
```

Anote o ARN da função retornado pelo comando. Você precisará dele ao adicionar permissões para permitir que o Amazon SNS invoque sua função.

Configurar permissões entre contas

Na conta A (01234567891A), conceda permissão para a conta B (01234567891B) assinar o tópico:

```
aws sns add-permission --label lambda-access --aws-account-id 12345678901B \
--topic-arn arn:aws:sns:us-east-2:12345678901A:sns-topic-for-lambda \
--action-name Subscribe ListSubscriptionsByTopic --profile accountA
```

Na conta B (01234567891B) adicione a permissão do Lambda para permitir a invocação no Amazon SNS.

```
aws lambda add-permission --function-name Function-With-SNS \
--source-arn arn:aws:sns:us-east-2:12345678901A:sns-topic-for-lambda \
--statement-id function-with-sns --action "lambda:InvokeFunction" \
--principal sns.amazonaws.com --profile accountB
```

Você deve ver a saída a seguir:

```
{
  "Statement": "{\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":
    \"arn:aws:sns:us-east-2:12345678901A:sns-topic-for-lambda\"}},
  \"Action\":[\"lambda:InvokeFunction\"],
  \"Resource\":\"arn:aws:lambda:us-east-2:01234567891B:function:Function-With-SNS\",
  \"Effect\":\"Allow\",
  \"Principal\":{\"Service\":\"sns.amazonaws.com\"},
  \"Sid\":\"function-with-sns\"}"
}
```

Não use o parâmetro `--source-account` para adicionar uma conta de origem à política do Lambda ao adicionar a política. A conta de origem não é compatível com fontes de eventos do Amazon SNS e resultará em acesso negado.

Note

Se a conta com o tópico do SNS estiver hospedada em uma região opcional, será necessário especificar a região no principal. Para obter um exemplo, consulte [Invocar funções do Lambda usando notificações do Amazon SNS](#) no Guia do desenvolvedor do Amazon Simple Notification Service.

Criar uma assinatura

Na conta B, assine a função do Lambda no tópico. Quando uma mensagem é enviada para o tópico `sns-topic-for-lambda` na conta A (01234567891A), o Amazon SNS invoca a função `Function-With-SNS` na conta B (01234567891B).

```
aws sns subscribe --protocol lambda \
--topic-arn arn:aws:sns:us-east-2:12345678901A:sns-topic-for-lambda \
--notification-endpoint arn:aws:lambda:us-east-2:12345678901B:function:Function-With-SNS \
--profile accountB
```

Você deve ver a saída a seguir:

```
{  
    "SubscriptionArn": "arn:aws:sns:us-east-2:12345678901A:sns-topic-for-lambda:5d906xxxx-7c8x-45dx-a9dx-0484e31c98xx"  
}
```

A saída contém o ARN da assinatura do tópico.

Testar a assinatura

Na conta A (01234567891A), teste a assinatura. Digite Hello World em um arquivo de texto e salve-o como message.txt. Em seguida, execute o seguinte comando:

```
aws sns publish --message file://message.txt --subject Test \  
--topic-arn arn:aws:sns:us-east-2:12345678901A:sns-topic-for-lambda \  
--profile accountA
```

Isso retornará um id de mensagem com um identificador exclusivo indicando que a mensagem foi aceita pelo serviço do Amazon SNS. O Amazon SNS tentará enviá-lo aos assinantes do tópico. Como alternativa, você pode fornecer uma string JSON diretamente para o parâmetro message, mas o uso de um arquivo de texto permite quebras de linha na mensagem.

Para saber mais sobre o Amazon SNS, consulte [O que é o Amazon Simple Notification Service](#).

Limpar recursos da

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua conta da AWS.

Para excluir o tópico do Amazon SNS

1. Abra a [página Topics](#) (Tópicos) no console do Amazon SNS.
2. Selecione o tópico que você criou.
3. Escolha Delete.
4. Digite **delete me** na caixa de texto.
5. Escolha Delete.

Para excluir a função de execução

1. Abra a página [Roles](#) (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Selecione Delete role (Excluir função).
4. Escolha Sim, excluir.

Como excluir a função do Lambda

1. Abra a página [Functions](#) (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Actions, Delete.
4. Escolha Delete.

Para excluir a inscrição do Amazon SNS

1. Abrir o [Página Assinaturas](#) no console do Amazon SNS.
2. Selecione a assinatura que você criou.
3. Escolha Delete (Excluir), Yes, Delete (Sim, excluir).

Código de exemplo da função do

O código de amostra está disponível para as seguintes linguagens.

Tópicos

- [Node.js 12.x \(p. 462\)](#)
- [Java 11 \(p. 462\)](#)
- [Go \(p. 463\)](#)
- [Python 3 \(p. 463\)](#)

Node.js 12.x

O exemplo a seguir processa mensagens do Amazon SNS registra seu conteúdo.

Example index.js

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
// console.log('Received event:', JSON.stringify(event, null, 4));

    var message = event.Records[0].Sns.Message;
    console.log('Message received from SNS:', message);
    callback(null, "Success");
};
```

Compreenda o código do exemplo para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Node.js com arquivos .zip \(p. 517\)](#).

Java 11

O exemplo a seguir processa mensagens do Amazon SNS registra seu conteúdo.

Example LogEvent.java

```
package example;

import java.text.SimpleDateFormat;
import java.util.Calendar;

import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.events.SNSEvent;

public class LogEvent implements RequestHandler<SNSEvent, Object> {
    public Object handleRequest(SNSEvent request, Context context){
        String timeStamp = new SimpleDateFormat("yyyy-MM-
dd_HH:mm:ss").format(Calendar.getInstance().getTime());
        context.getLogger().log("Invocation started: " + timeStamp);
```

```
    context.getLogger().log(request.getRecords().get(0).getSNS().getMessage());  
  
    timeStamp = new SimpleDateFormat("yyyy-MM-  
dd_HH:mm:ss").format(Calendar.getInstance().getTime());  
    context.getLogger().log("Invocation completed: " + timeStamp);  
    return null;  
}  
}
```

Dependencies

- aws-lambda-java-core
- aws-lambda-java-events

Crie o código com as dependências da biblioteca do Lambda para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Java com arquivos .zip ou JAR \(p. 599\)](#).

Go

O exemplo a seguir processa mensagens do Amazon SNS registra seu conteúdo.

Example lambda_handler.go

```
package main  
  
import (  
    "context"  
    "fmt"  
    "github.com/aws/aws-lambda-go/lambda"  
    "github.com/aws/aws-lambda-go/events"  
)  
  
func handler(ctx context.Context, snsEvent events.SNSEvent) {  
    for _, record := range snsEvent.Records {  
        snsRecord := record.SNS  
        fmt.Printf("[%s %s] Message = %s \n", record.EventSource, snsRecord.Timestamp,  
        snsRecord.Message)  
    }  
}  
  
func main() {  
    lambda.Start(handler)  
}
```

Crie o executável com o `go build` e crie um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Go com arquivos .zip \(p. 640\)](#).

Python 3

O exemplo a seguir processa mensagens do Amazon SNS registra seu conteúdo.

Example lambda_handler.py

```
from __future__ import print_function  
import json  
print('Loading function')  
  
def lambda_handler(event, context):  
    #print("Received event: " + json.dumps(event, indent=2))
```

```
message = event['Records'][0]['Sns']['Message']
print("From SNS: " + message)
return message
```

Compaca o código do exemplo para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Python com arquivos .zip \(p. 543\)](#).

O uso do AWS Lambda Com o Amazon SQS

Você pode usar um AWS Lambda para processar mensagens em uma fila do Amazon Simple Queue Service (Amazon SQS). Lambda [mapeamentos da fonte de eventos do \(p. 170\)](#) suporta [filas padrão e filas FIFO \(primeiro a entrar, primeiro a sair\)](#). Com o Amazon SQS, você pode descarregar tarefas de um componente do aplicativo enviando-as a uma fila e processando-as de forma assíncrona.

O Lambda sonda a fila e invoca a função do Lambda [síncrona \(p. 158\)](#) com um evento que contém mensagens de fila. O Lambda lê mensagens em lotes e invoca sua função uma vez para cada lote. Quando sua função processa um lote com êxito, o Lambda exclui suas mensagens da fila. O exemplo a seguir mostra um evento para um lote de duas mensagens.

Example Evento de mensagem do Amazon SQS (fila padrão)

```
{  
    "Records": [  
        {  
            "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",  
            "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",  
            "body": "Test message.",  
            "attributes": {  
                "ApproximateReceiveCount": "1",  
                "SentTimestamp": "1545082649183",  
                "SenderId": "AIDAENQZJLO23YYJ4VO",  
                "ApproximateFirstReceiveTimestamp": "1545082649185"  
            },  
            "messageAttributes": {},  
            "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": "AIDAENQZJLO23YYJ4VO",  
                "ApproximateFirstReceiveTimestamp": "1545082650649"  
            },  
            "messageAttributes": {},  
            "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",  
            "eventSource": "aws:sqs",  
            "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",  
            "awsRegion": "us-east-2"  
        }  
    ]  
}
```

Por padrão, o Lambda invoca sua função assim que os registros são disponibilizados na fila do SQS. O Lambda pesquisará até 10 mensagens em sua fila de uma só vez e enviará esse lote para a função. Para evitar invocar a função com um número pequeno de registros, você pode instruir à origem dos eventos para fazer o buffer dos registros por até cinco minutos, configurando uma janela de lote. Antes de invocar a função, o Lambda continua a pesquisar mensagens da fila padrão do SQS até a janela do lote expirar, e o [limite da carga útil \(p. 53\)](#) ou o tamanho máximo do lote ser alcançado.

Para as filas FIFO, os registros contêm atributos adicionais relacionados a desduplicação e sequenciamento.

Example Evento de mensagem do Amazon SQS (fila FIFO)

```
{  
    "Records": [  
        {  
            "messageId": "11d6ee51-4cc7-4302-9e22-7cd8afdaadf5",  
            "receiptHandle": "AQEBBX8nesZEXmkhsmZeyIE8iQAMig7qw...",  
            "body": "Test message.",  
            "attributes": {  
                "ApproximateReceiveCount": "1",  
                "SentTimestamp": "1573251510774",  
                "SequenceNumber": "18849496460467696128",  
                "MessageGroupId": "1",  
                "SenderId": "AIDAIO23YVJENQZJOL4VO",  
                "MessageDuplicationId": "1",  
                "ApproximateFirstReceiveTimestamp": "1573251510774"  
            },  
            "messageAttributes": {},  
            "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",  
            "eventSource": "aws:sqs",  
            "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:fifo fifo",  
            "awsRegion": "us-east-2"  
        }  
    ]  
}
```

Quando o Lambda lê um lote, as mensagens permanecem na fila, mas são ocultadas durante o [tempo limite de visibilidade](#) da fila. Se a função processar com êxito um lote, o Lambda excluirá as mensagens da fila. Se a função for [limitada \(p. 32\)](#), retornar um erro ou não responder, a mensagem ficará visível novamente. Todas as mensagens em um lote com falha retornam à fila e, portanto, seu código de função deve ser capaz de processar a mesma mensagem várias vezes sem efeitos colaterais.

Seções

- [Escalabilidade e processamento \(p. 466\)](#)
- [Configurar uma fila para usar com o Lambda \(p. 467\)](#)
- [Permissões da função de execução \(p. 467\)](#)
- [Configurar uma fila como uma fonte de eventos \(p. 467\)](#)
- [APIs de mapeamento da fonte de eventos \(p. 336\)](#)
- [Tutorial: como usar Lambda com Amazon SQS \(p. 469\)](#)
- [Exemplo de código de função do Amazon SQS \(p. 473\)](#)
- [Modelo do AWS SAM para uma aplicação do Amazon SQS \(p. 476\)](#)

Escalabilidade e processamento

Para filas padrão, o Lambda usa [sondagem longa](#) para sondar uma fila até que ela se torne ativa. Quando as mensagens estão disponíveis, o Lambda lê até cinco lotes e os envia para sua função. Se ainda houver mensagens disponíveis, o Lambda aumentará o número de processos de leitura de lotes em até 60 instâncias a mais por minuto. O número máximo de lotes que podem ser processados simultaneamente por um mapeamento da origem do evento é 1000.

Para filas FIFO, o Lambda envia mensagens para a função na ordem em que as recebe. Ao enviar uma mensagem para uma fila do FIFO, você especifica um [ID do grupo de mensagens](#). O Amazon SQS garante que as mensagens no mesmo grupo sejam entregues ao Lambda em ordem. O Lambda classifica as mensagens em grupos e envia apenas um lote de cada vez para um grupo. Se a função retornar um erro, serão feitas todas as tentativas nas mensagens afetadas antes que o Lambda receba mensagens adicionais do mesmo grupo.

Sua função pode reduzir a simultaneidade para o número de grupos de mensagens ativos. Para obter mais informações, consulte [FIFO do SQS como uma origem de evento](#) no blog de computação da AWS.

Configurar uma fila para usar com o Lambda

Crie [uma fila do SQS](#) para servir como uma fonte de eventos para sua função do Lambda. Depois, configure a fila para permitir que a sua função do Lambda processe cada lote de eventos e para o Lambda tentar novamente em resposta a erros de limitação à medida que a escala aumenta na vertical.

Para permitir que a função tenha tempo para processar cada lote de registros, defina o tempo limite de visibilidade da fila de origem para pelo menos seis vezes o [tempo limite \(p. 95\)](#) configurado na função. O tempo extra permite que o Lambda repita o processo se a execução da sua função for limitada enquanto sua função está processando um lote anterior.

Se uma mensagem falhar no processamento várias vezes, o Amazon SQS poderá enviá-la para uma [fila de mensagens mortas](#). Quando sua função retorna um erro, o Lambda a deixa na fila. Após o tempo limite de visibilidade, o Lambda recebe a mensagem novamente. Para enviar mensagens para uma segunda fila após vários recebimentos, configure uma fila de mensagens mortas na fila de origem.

Note

Certifique-se de configurar a fila de mensagens mortas na fila de origem, e não na função do Lambda. A fila dead-letter que você configura em uma função é usada para a [fila de invocação assíncrona \(p. 161\)](#) da função, e não para filas de origem de evento.

Se a função retornar um erro ou não puder ser invocada porque está na simultaneidade máxima, o processamento poderá ter êxito com tentativas adicionais. Para que as mensagens tenham maior chance de serem processadas antes de serem enviadas para a fila de mensagens mortas, defina a `maxReceiveCount` na política de redirecionamento da fila de origem como pelo menos 5.

Permissões da função de execução

O Lambda precisa das permissões a seguir para gerenciar mensagens em sua fila do Amazon SQS. Adicione-as à função de execução da sua função.

- `sqs:ReceiveMessage`
- `sqs:DeleteMessage`
- `sqs:GetQueueAttributes`

Para obter mais informações, consulte [AWS LambdaFunção de execução do \(p. 57\)](#).

Configurar uma fila como uma fonte de eventos

Crie um mapeamento de fontes de eventos para orientar o Lambda a enviar itens da sua fila para uma função do Lambda. Você pode criar vários mapeamentos da fonte do evento para processar itens de várias filas com uma única função. Quando o Lambda invoca a função de destino, o evento pode conter vários itens, até um tamanho de lote máximo configurável.

Para configurar sua função para leitura a partir do Amazon SQS no console do Lambda, crie um acionador do SQS.

Para criar um trigger

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.

3. Em Function overview (Visão geral da função), escolha Add trigger (Adicionar gatilho).
4. Escolha um tipo de acionador.
5. Configure as opções necessárias e escolha Add (Adicionar).

O Lambda oferece suporte às opções de fontes de eventos do Amazon SQS a seguir.

Opções de fonte do evento

- SQS queue (Fila do SQS): a fila do Amazon SQS de onde os registros serão lidos.
- Batch size (Tamanho do lote): o número de registros a serem enviados para a função em cada lote. Para uma fila padrão, isso pode ser de até 10.000 registros. Para uma fila FIFO, o máximo é 10. Para um tamanho de lote acima de 10, você também deve definir o `MaximumBatchingWindowInSeconds`parâmetro para pelo menos 1 segundo. O Lambda transmite todos os registros no batch para a função em uma única chamada, enquanto o tamanho total dos eventos não excede o [limite de carga útil \(p. 53\)](#) para invocação síncrona (6 MB).

São gerados metadados pelo Lambda e pelo Amazon SQS para cada registro. Esses metadados adicionais são contabilizados para efeitos do tamanho total da carga útil e podem fazer com que o número total de registros enviados em um lote seja menor do que o tamanho do seu lote configurado. Os campos de metadados enviados pelo Amazon SQS podem variar em comprimento. Para obter mais informações sobre os campos de metadados do Amazon SQS, consulte [oReceiveMessage](#)documentação noReferência da API do Amazon Simple Queue Service.

- Batch window (Janela de lote): especifique o máximo de tempo para reunir registros antes de invocar a função, em segundos. Aplicável apenas a filas padrão.

Se você estiver usando uma janela de lote maior que 0 segundo, considere o maior tempo de processamento no tempo limite de visibilidade de sua fila. Recomendamos definir o tempo limite de visibilidade da fila para seis vezes o tempo limite da função, mais o valor de `MaximumBatchingWindowInSeconds`. Isso permite que sua função do Lambda tenha tempo para processar cada lote de eventos e tentar novamente em caso de erro de controle de utilização.

Note

Se a janela do batch for maior que 0 e `(batch window) + (function timeout) > (queue visibility timeout)`, seu tempo limite efetivo de visibilidade da fila será `(batch window) + (function timeout) + 30s`.

O Lambda processa até cinco lotes de cada vez. Isso significa que há um máximo de cinco trabalhadores disponíveis para agrupar e processar mensagens em paralelo a qualquer momento. Cada operador mostrará uma invocação do Lambda distinta para seu lote atual de mensagens.

- Enabled (Habilitado): defina como verdadeiro para habilitar o mapeamento de fontes de eventos. Defina como falso para interromper o processamento de registros.

Note

O Amazon SQS tem um nível gratuito vitalício para solicitações. Além do nível gratuito, o Amazon SQS cobra por milhão de solicitações. Enquanto o mapeamento de fontes de eventos estiver ativo, o Lambda fará solicitações à fila para obter itens. Para obter detalhes de preço, consulte[Preços do Amazon Simple Queue Service](#).

Para gerenciar a configuração da fonte do evento posteriormente, escolha o gatilho no designer.

Configure o tempo limite de sua função para permitir tempo suficiente para processar um lote inteiro de itens. Se os itens levarem muito tempo para serem processados, escolha um tamanho de lote menor. Um tamanho de lote grande pode melhorar a eficiência de cargas de trabalho muito rápidas ou com muita sobrecarga. No entanto, se a sua função retornar um erro, todos os itens no lote retornarão à

fila. Se você configurar a [simultaneidade reservada \(p. 113\)](#) em sua função, defina, no mínimo, cinco execuções simultâneas para reduzir a chance de erros de controle de utilização quando o Lambda invocar sua função. Para eliminar a chance de erros de controle de utilização, defina o valor da [simultaneidade reservada \(p. 113\)](#) como 1000, que é o número máximo de execuções simultâneas para uma fonte de eventos do Amazon SQS.

APIs de mapeamento da fonte de eventos

Para gerenciar uma origem de evento com a [AWS CLI](#) ou o [AWS SDK](#), você pode usar as seguintes operações da API:

- [CreateEventSourceMapping \(p. 786\)](#)
- [ListEventSourceMappings \(p. 888\)](#)
- [GetEventSourceMapping \(p. 837\)](#)
- [UpdateEventSourceMapping \(p. 956\)](#)
- [DeleteEventSourceMapping \(p. 812\)](#)

O exemplo a seguir usa a AWS CLI para mapear uma função chamada `my-function` para uma fila do Amazon SQS especificada pelo nome do recurso da Amazon (ARN), com um tamanho de lote de 5 e uma janela de lote de 60 segundos.

```
aws lambda create-event-source-mapping --function-name my-function --batch-size 5 \
--maximum-batching-window-in-seconds 60 \
--event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue
```

Você deve ver a saída a seguir:

```
{  
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",  
    "BatchSize": 5,  
    "MaximumBatchingWindowInSeconds": 60,  
    "EventSourceArn": "arn:aws:sqs:us-east-2:123456789012:my-queue",  
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",  
    "LastModified": 1541139209.351,  
    "State": "Creating",  
    "StateTransitionReason": "USER_INITIATED"  
}
```

Tutorial: como usar Lambda com Amazon SQS

Neste tutorial, você criará uma função do Lambda que consuma mensagens de uma fila do [Amazon Simple Queue Service \(Amazon SQS\)](#).

Prerequisites

Este tutorial presume que você tenha algum conhecimento de operações básicas do Lambda e do console do Lambda. Caso ainda não tenha feito isso, siga as instruções em [Conceitos básicos do Lambda \(p. 9\)](#) para criar sua primeira função do Lambda.

Para concluir as etapas a seguir, você precisa de um terminal de linha de comando ou de um shell para executar os comandos. Os comandos e a saída esperada são mostrados em blocos separados:

```
aws --version
```

Você deve ver a saída a seguir:

```
aws-cli/2.0.57 Python/3.7.4 Darwin/19.6.0 exe/x86_64
```

Para comandos longos, um caractere de escape (\) é usado para dividir um comando em várias linhas.

No Linux e no macOS, use seu gerenciador preferido de pacotes e de shell. No Windows 10, você pode [instalar o Subsistema Windows para Linux](#) para obter uma versão do Ubuntu integrada com o Windows e o Bash.

Criar a função de execução

Crie uma [função de execução \(p. 57\)](#) que dê à sua função permissão para acessar recursos da AWS necessários.

Para criar uma função de execução

1. Abra a página [Roles \(Funções\)](#) no console do AWS Identity and Access Management (IAM).
2. Selecione Create role.
3. Crie uma função com as seguintes propriedades.
 - Trusted entity (Entidade confiável – AWS Lambda).
 - Permissões: AWSLambdaSQSQueueExecutionRole.
 - Nome da função – **lambda-sqs-role**.

A política AWSLambdaSQSQueueExecutionRole tem as permissões necessárias para a função ler itens do Amazon SQS e para gravar logs no Amazon CloudWatch Logs.

Criar a função

Crie uma função do Lambda que processa suas mensagens do Amazon SQS. O exemplo de código Node.js 12 a seguir grava cada mensagem em um log do CloudWatch Logs.

Note

Para exemplos de código em outras linguagens, consulte [Exemplo de código de função do Amazon SQS \(p. 473\)](#).

Example index.js

```
exports.handler = async function(event, context) {
  event.Records.forEach(record => {
    const { body } = record;
    console.log(body);
  });
  return {};
}
```

Para criar a função

Note

Seguir estas etapas cria uma função no Node.js 12. Para outras linguagens, as etapas são semelhantes, mas alguns detalhes são diferentes.

1. Salve o exemplo de código como um arquivo chamado `index.js`.
2. Crie um pacote de implantação.

```
zip function.zip index.js
```

3. Crie a função usando o comando do `create-function` AWS Command Line Interface (AWS CLI) .

```
aws lambda create-function --function-name ProcessSQSRecord \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs12.x \
--role arn:aws:iam::123456789012:role/lambda-sqs-role
```

Testar a função

Invoque sua função do Lambda manualmente usando o comando `invoke` AWS CLI e um evento do Amazon SQS de amostra.

Se o manipulador obtém um retorno normal sem exceções, o Lambda considera a mensagem como processada com êxito e começa a ler novas mensagens na fila. Após o processamento com sucesso, uma mensagem, o Lambda a exclui automaticamente da fila. Se o manipulador gera uma exceção, o Lambda considera que o batch de mensagens que não foram processadas com êxito e o Lambda invoca novamente a função com o mesmo batch de mensagens.

1. Salve o JSON a seguir como um arquivo chamado `input.txt`.

```
{
    "Records": [
        {
            "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
            "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
            "body": "test",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1545082649183",
                "SenderId": "AIDAIEQZJOLO23YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082649185"
            },
            "messageAttributes": {},
            "md5OfBody": "098f6bcd4621d373cade4e832627b4f6",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
            "awsRegion": "us-east-2"
        }
    ]
}
```

O JSON anterior simula um evento que o Amazon SQS pode enviar para a função do Lambda, em que "body" contém a mensagem real da fila.

2. Execute o comando `invoke` AWS CLI a seguir.

```
aws lambda invoke --function-name ProcessSQSRecord \
--payload file://input.txt outputfile.txt
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

3. Verifique a saída no arquivo `outputfile.txt`.

Criar uma fila do Amazon SQS

Crie uma fila do Amazon SQS que a função do Lambda possa usar como uma fonte de eventos.

Para criar uma fila

1. Abra o [console do Amazon SQS](#).
2. Selecione Criar fila e, em seguida, configure a fila. Para obter instruções detalhadas, consulte [Criar uma fila do Amazon SQS \(console\)](#) no Guia do desenvolvedor do Amazon Simple Queue Service.
3. Depois de criar a fila, registre o nome do recurso da Amazon (ARN). Você precisará dele na próxima etapa, quando for associar a fila à função do Lambda.

Configurar a origem de evento

Para criar um mapeamento entre a fila do Amazon SQS e a função do Lambda, execute o comando da `create-event-source-mapping` AWS CLI a seguir.

```
aws lambda create-event-source-mapping --function-name ProcessSQSRecord --batch-size 10 \
--event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue
```

Para obter uma lista de mapeamentos de fonte do evento, execute o comando a seguir.

```
aws lambda list-event-source-mappings --function-name ProcessSQSRecord \
--event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue
```

Testar a configuração

Agora você pode testar a configuração da seguinte forma:

1. Abra o [console do Amazon SQS](#).
2. Escolha o nome da fila que você criou anteriormente.
3. Escolha Send and receive messages (Enviar e receber mensagens).
4. No Corpo da mensagem, insira uma mensagem de teste.
5. Escolha Send message (Enviar mensagem).

O Lambda pesquisa a fila para atualizações. Quando há uma nova mensagem, o Lambda invoca a sua função com esses novos dados de evento da fila. A função é executada e cria logs no Amazon CloudWatch. Você pode visualizar os logs no console do [CloudWatch](#).

Limpar recursos da

Agora você pode excluir os recursos criados para este tutorial, a menos que queira mantê-los. Excluindo os recursos da AWS que você não está mais usando, você evita cobranças desnecessárias em sua conta da AWS.

Para excluir a função de execução

1. Abra a página [Roles](#) (Funções) no console do IAM.
2. Selecione a função de execução que você criou.
3. Selecione Delete role (Excluir função).
4. Escolha Sim, excluir.

Como excluir a função do Lambda

1. Abra a página [Functions](#) (Funções) no console do Lambda.
2. Selecione a função que você criou.
3. Escolha Actions, Delete.
4. Escolha Delete.

Para excluir a fila do Amazon SQS

1. Faça login no AWS Management Console e abra o console do Amazon SQS em <https://console.aws.amazon.com/sqs/>.
2. Selecione a fila que você criou.
3. Escolha Delete.
4. Digite **delete** na caixa de texto.
5. Escolha Delete.

Exemplo de código de função do Amazon SQS

O código de amostra está disponível para as seguintes linguagens.

Tópicos

- [Node.js \(p. 473\)](#)
- [Java \(p. 473\)](#)
- [C# \(p. 474\)](#)
- [Go \(p. 475\)](#)
- [Python \(p. 475\)](#)

Node.js

A seguir, um exemplo de código que recebe uma mensagem de evento do Amazon SQS como uma entrada e a processa. Na ilustração, o código grava alguns dos dados de entrada no CloudWatch Logs.

Example index.js (Node.js 12)

```
exports.handler = async function(event, context) {
  event.Records.forEach(record => {
    const { body } = record;
    console.log(body);
  });
  return {};
}
```

Compreenda o código do exemplo para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Node.js com arquivos .zip \(p. 517\)](#).

Java

A seguir, um exemplo de código Java que recebe uma mensagem de evento do Amazon SQS como uma entrada e a processa. Na ilustração, o código grava alguns dos dados de entrada no CloudWatch Logs.

No código, `handleRequest` é o handler. O handler usa a classe predefinida `SQSEvent` definida na biblioteca `aws-lambda-java-events`.

Example Handler.java

```
package example;

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 Handler implements RequestHandler<SQSEvent, Void>{
    @Override
    public Void handleRequest(SQSEvent event, Context context)
    {
        for(SQSMessage msg : event.getRecords()){
            System.out.println(new String(msg.getBody()));
        }
        return null;
    }
}
```

Dependencies

- `aws-lambda-java-core`
- `aws-lambda-java-events`

Crie o código com as dependências da biblioteca do Lambda para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Java com arquivos .zip ou JAR \(p. 599\)](#).

C#

A seguir, um exemplo de código C# que recebe uma mensagem de evento do Amazon SQS como uma entrada e a processa. Na ilustração, o código grava alguns dos dados de eventos de entrada no console.

No código, `handleRequest` é o handler. O handler usa a classe predefinida `SQSEvent` definida na biblioteca `AWS.Lambda.SQSEvents`.

Example ProcessingSQSRecords.cs

```
[assembly: LambdaSerializer(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]

namespace SQSLambdaFunction
{
    public class SQSLambdaFunction
    {
        public string HandleSQSEvent(SQSEvent sqsEvent, ILambdaContext context)
        {
            Console.WriteLine($"Beginning to process {sqsEvent.Records.Count} records...");

            foreach (var record in sqsEvent.Records)
            {
                Console.WriteLine($"Message ID: {record.MessageId}");
                Console.WriteLine($"Event Source: {record.EventSource}");

                Console.WriteLine($"Record Body:");
                Console.WriteLine(record.Body);
            }
        }
    }
}
```

```
        Console.WriteLine("Processing complete.");
    }
}
}
```

Substitua o `Program.cs` em um projeto do .NET Core pelo exemplo acima. Para obter instruções, consulte [Implantar funções do Lambda em C# com arquivos .zip \(p. 668\)](#).

Go

A seguir, um exemplo de código Go que recebe uma mensagem de evento do Amazon SQS como uma entrada e a processa. Na ilustração, o código grava alguns dos dados de eventos de entrada no CloudWatch Logs.

No código, `handler` é o handler. O handler usa a classe predefinida `SQSEvent` definida na biblioteca `aws-lambda-go-events`.

Example ProcessSQSRecords.go

```
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) error {
    for _, message := range sqsEvent.Records {
        fmt.Printf("The message %s for event source %s = %s \n",
            message.MessageId,
            message.EventSource, message.Body)
    }

    return nil
}

func main() {
    lambda.Start(handler)
}
```

Crie o executável com o `go build` e crie um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Go com arquivos .zip \(p. 640\)](#).

Python

A seguir, um exemplo de código Python que aceita um registro do Amazon SQS como entrada e o processa. Na ilustração, o código grava alguns dos dados de entrada no CloudWatch Logs.

Example ProcessSQSRecords.py

```
from __future__ import print_function

def lambda_handler(event, context):
    for record in event['Records']:
        print("test")
        payload = record["body"]
```

```
print(str(payload))
```

Comprenda o código do exemplo para criar um pacote de implantação. Para obter instruções, consulte [Implantar funções do Lambda em Python com arquivos .zip \(p. 543\)](#).

Modelo do AWS SAM para uma aplicação do Amazon SQS

Você pode criar esse aplicativo usando [AWS SAM](#). Para saber mais sobre como criar modelos do AWS SAM, consulte [Noções básicas de modelos do AWS SAM](#) no Guia do desenvolvedor do AWS Serverless Application Model.

Veja abaixo um modelo de exemplo do AWS SAM para a aplicação do Lambda do [tutorial \(p. 469\)](#). Copie o texto abaixo para um arquivo `.yaml` e salve-o ao lado do pacote ZIP criado previamente. Observe que os valores dos parâmetros `Handler` e `Runtime` devem corresponder àqueles usados quando você criou a função na seção anterior.

Example template.yaml

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Example of processing messages on an SQS queue with Lambda
Resources:
  MySQSQueueFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs12.x
    Events:
      MySQSEvent:
        Type: SQS
        Properties:
          Queue: !GetAtt MySqsQueue.Arn
          BatchSize: 10
  MySqsQueue:
    Type: AWS::SQS::Queue
```

Para obter informações sobre como empacotar e implantar o aplicativo sem servidor usando os comandos de empacotamento e implantação, consulte [Implantar aplicativos sem servidor](#) no Guia do desenvolvedor do AWS Serverless Application Model.

Usar o AWS Lambda com o AWS X-Ray

Você pode usar o AWS X-Ray para visualizar os componentes do aplicativo, identificar gargalos de performance e solucionar problemas de solicitações que resultaram em um erro. Suas funções do Lambda enviam dados de rastreamento para o X-Ray, e o X-Ray processa os dados para gerar um mapa de serviço e resumos de rastreamento pesquisáveis.



Se você tiver habilitado o rastreamento do X-Ray em um serviço que invoca sua função, o Lambda enviará rastreamentos para o X-Ray automaticamente. O serviço upstream, como Amazon API Gateway, ou uma aplicação hospedada no Amazon EC2 que é instrumentada com o X-Ray SDK, analisa solicitações de entrada e adiciona um cabeçalho de rastreamento que informa ao Lambda para enviar rastreamentos ou não.

Para rastrear solicitações que não têm um cabeçalho de rastreamento, ative o rastreamento ativo na configuração da sua função.

Para ativar o rastreamento ativo

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Selecione **Configuração**, depois, escolha **Ferramentas de monitoramento**.
4. Selecione **Edit**.
5. Em X-Ray, habilite o **Active tracing (Rastreamento ativo)**.
6. Escolha **Save (Salvar)**.

Pricing

O X-Ray tem um nível gratuito vitalício. Além do limite do nível gratuito, o X-Ray cobra por armazenamento e recuperação de rastreamento. Para obter detalhes, consulte [Definição de preço do AWS X-Ray](#).

Sua função precisa de permissão para carregar dados de rastreamento no X-Ray. Quando você habilita o rastreamento ativo no console do Lambda, o Lambda adiciona as permissões necessárias à [função de execução \(p. 57\)](#) da função. Caso contrário, adicione a política [AWSXRayDaemonWriteAccess](#) à função de execução.

O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento é eficiente, enquanto ainda fornece uma amostra representativa das solicitações que a sua aplicação serviu. A regra de amostragem padrão é uma solicitação por segundo e 5% de solicitações adicionais. Esta taxa de amostragem não pode ser configurada para funções do Lambda.

No X-Ray, um rastreamento registra informações sobre uma solicitação que é processada por um ou mais serviços. Registro de serviçosSegments do que contêm camadas deSubsegmentos. O Lambda registra um segmento para o serviço do Lambda que lida com a solicitação de invocação e um para o trabalho realizado pela função. O segmento de função vem com subsegmentos para [Initialization](#), [Invocation](#) e [Overhead](#). Para obter mais informações, consulte [Lambda execution environment lifecycle \(p. 219\)](#) (Ciclo de vida do ambiente de execução do Lambda).

O subsegmento [Initialization](#) representa a fase inicial do ciclo de vida do ambiente de execução do Lambda. Durante essa fase, Lambda cria ou descongela um ambiente de execução com os recursos configurados, faz download do código da função e de todas as camadas, inicializa extensões, inicializa o tempo de execução e executa o código de inicialização da função.

O subsegmento [Invocation](#) representa a fase de chamada em que o Lambda chama o manipulador de função. Isso começa com o registro do tempo de execução e da extensão e termina quando o tempo de execução está pronto para enviar a resposta.

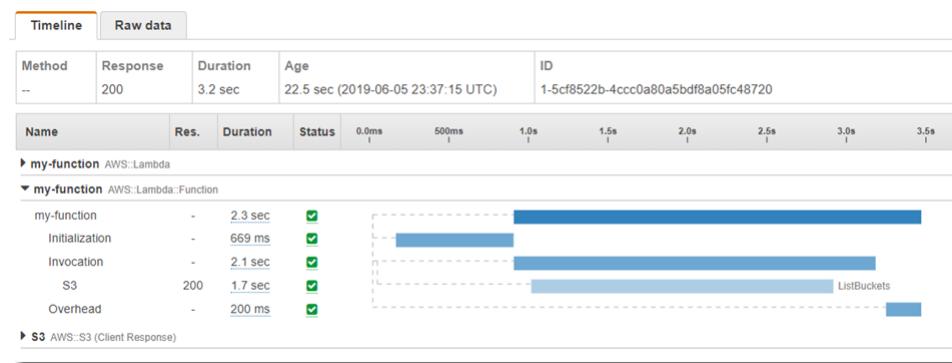
O subsegmento [Overhead](#) representa a fase que ocorre entre o momento em que o tempo de execução envia a resposta e o sinal para a próxima chamada. Durante esse período, o tempo de execução termina todas as tarefas relacionadas a uma chamada e se prepara para congelar o sandbox.

Note

Se sua função do Lambda[Simultaneidade provisionada \(p. 116\)](#), o rastreamento de X-Ray pode exibir uma inicialização de função com uma duração muito longa.

A simultaneidade provisionada inicializa instâncias de função com antecedência, para reduzir o atraso no momento da invocação. Ao longo do tempo, a simultaneidade provisionada atualiza essas instâncias criando novas instâncias para substituir as antigas. Para cargas de trabalho com tráfego estável, as novas instâncias são inicializadas bem antes de sua primeira chamada. O intervalo de tempo é registrado no rastreamento de X-Ray como a duração da inicialização.

O exemplo a seguir mostra um rastreamento com 2 segmentos. Ambos são chamados my-function, mas um é do tipo AWS::Lambda e o outro é AWS::Lambda::Function. O segmento de função é expandido para mostrar seus subsegmentos.



Important

No Lambda, é possível usar o X-Ray SDK para estender o subsegmento [Invocation](#) com subsegmentos adicionais para anotações, metadados e chamadas downstream. Não é possível acessar o segmento de função diretamente nem registrar trabalhos feitos fora do escopo de chamada do manipulador.

Consulte os tópicos a seguir para obter uma introdução específica de uma linguagem ao rastreamento no Lambda:

- [Instrumentação do código Node.js no AWS Lambda \(p. 533\)](#)
- [Instrumentação do código Python no AWS Lambda \(p. 562\)](#)
- [Instrumentar o código Ruby no AWS Lambda \(p. 587\)](#)
- [Instrumentação do código Java no AWS Lambda \(p. 623\)](#)
- [Instrumentação do código Go no AWS Lambda \(p. 656\)](#)
- [Instrumentar o código C # no AWS Lambda \(p. 688\)](#)

Para obter uma lista completa de serviços que oferecem suporte à instrumentação ativa, consulte [Serviços da AWS compatíveis](#) no Guia do desenvolvedor do AWS X-Ray.

Seções

- [Permissões da função de execução \(p. 479\)](#)
- [O daemon do AWS X-Ray \(p. 479\)](#)
- [Habilitar o rastreamento ativo com a API do Lambda \(p. 479\)](#)
- [Habilitar o rastreamento ativo com o AWS CloudFormation \(p. 480\)](#)

Permissões da função de execução

O Lambda precisa das permissões a seguir para enviar dados de rastreamento para o X-Ray. Adicione-as à [função de execução \(p. 57\)](#) da sua função.

- [xray:PutTraceSegments](#)
- [xray:PutTelemetryRecords](#)

Essas permissões estão incluídas na política gerenciada [AWSXRayDaemonWriteAccess](#).

O daemon do AWS X-Ray

Em vez de enviar dados de rastreamento diretamente para a API do X-Ray, o X-Ray SDK usa um processo do daemon. O daemon do AWS X-Ray é uma aplicação que é executada no ambiente do Lambda e escuta o tráfego UDP que contém segmentos e subsegmentos. Ele armazena em buffer os dados recebidos e os grava no X-Ray em lotes, reduzindo a sobrecarga de processamento e memória necessária para rastrear invocações.

O tempo de execução do Lambda permite ao daemon até 3% da memória configurada da sua função ou 16 MB, o que for maior. Se sua função ficar sem memória durante a invocação, o tempo de execução encerrará o processo do daemon primeiro para liberar memória.

O processo do daemon é totalmente gerenciado pelo Lambda e não pode ser configurado pelo usuário. Todos os segmentos gerados por invocações de função são registrados na mesma conta que a função do Lambda. O daemon não pode ser configurado para redirecioná-los para nenhuma outra conta.

Para obter mais informações, consulte [The X-Ray daemon](#) no Guia do desenvolvedor do X-Ray.

Habilitar o rastreamento ativo com a API do Lambda

Para gerenciar a configuração de rastreamento com a AWS CLI ou com o AWS SDK, use as seguintes operações de API:

- [UpdateFunctionConfiguration \(p. 974\)](#)
- [GetFunctionConfiguration \(p. 851\)](#)
- [CreateFunction \(p. 796\)](#)

O exemplo de comando da AWS CLI a seguir habilita o rastreamento ativo em uma função chamada my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

O modo de rastreamento faz parte da configuração específica da versão que é bloqueada quando você publica uma versão da função. Não é possível alterar o modo de rastreamento em uma versão publicada.

Habilitar o rastreamento ativo com o AWS CloudFormation

Para habilitar o rastreamento ativo em um recurso `AWS::Lambda::Function` em um modelo do AWS CloudFormation, use a propriedade `TracingConfig`.

Example [function-inline.yml](#): configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
    ...
```

Para um recurso do AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, use a propriedade `Tracing`.

Example [template.yml](#): configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
    ...
```

Orquestrar funções com o Step Functions

O AWS Step Functions é um serviço de orquestração que permite conectar as funções do Lambda a fluxos de trabalho sem servidor chamados de máquinas de estado. Use o Step Functions para orquestrar fluxos de trabalho de aplicações sem servidor (por exemplo, o processo de finalização de compra de uma loja), criar fluxos de trabalho de longa duração para automação de TI e casos de uso de aprovação humana, ou criar fluxos de trabalho de alto volume e curta duração para processamento e ingestão de dados de transmissão.

Tópicos

- [Padrões de aplicação de máquinas de estado \(p. 481\)](#)
- [Gerenciar máquinas de estado no console do Lambda \(p. 485\)](#)
- [Exemplos de orquestração com o Step Functions \(p. 486\)](#)

Padrões de aplicação de máquinas de estado

No Step Functions, você orquestra os recursos usando máquinas de estado, que são definidas pelo uso de uma linguagem estruturada baseada em JSON, chamada [Amazon States Language](#).

Seções

- [Componentes das máquinas de estado \(p. 481\)](#)
- [Padrões de aplicação de máquinas de estado \(p. 481\)](#)
- [Aplicação de padrões em máquinas de estado \(p. 482\)](#)
- [Exemplo de padrão de aplicação de ramificação \(p. 482\)](#)

Componentes das máquinas de estado

As máquinas de estado contêm elementos chamados [estados](#) que compõem seu fluxo de trabalho. A lógica de cada estado determina qual estado vem a seguir, quais dados devem ser transmitidos e quando encerrar o fluxo de trabalho. Um estado é chamado por seu nome, que, embora possa ser qualquer string, deve ser exclusivo no escopo da máquina de estado como um todo.

Para criar uma máquina de estado que use o Lambda, você precisa dos seguintes componentes:

1. Uma função do AWS Identity and Access Management (IAM) para o Lambda com uma ou mais políticas de permissão (como permissões de serviço [AWSLambdaRole](#)).
2. Uma ou mais funções do Lambda (com a função do IAM anexada) para o tempo de execução específico.
3. Uma máquina de estado criada na Amazon States Language.

Padrões de aplicação de máquinas de estado

Você pode criar orquestrações complexas para máquinas de estado usando padrões de aplicação como:

- Detecção e nova tentativa: lida com erros usando uma funcionalidade sofisticada de detecção e nova tentativa.
- Ramificação: projeta o fluxo de trabalho para escolher diferentes ramificações com base na saída da função do Lambda.
- Encadeamento: conecta as funções em uma série de etapas, com a saída de uma etapa fornecendo a entrada para a próxima etapa.
- Paralelismo: executa funções em paralelo, ou usa paralelismo dinâmico para invocar uma função para cada membro de uma matriz.

Aplicação de padrões em máquinas de estado

Veja a seguir como aplicar esses padrões de aplicação a uma máquina de estado dentro de uma definição da Amazon States Language.

Detecção e nova tentativa

Um campo `Catch` e um campo `Retry` adicionam lógica de detecção e nova tentativa a uma máquina de estado. `Catch` ("Type": "Catch") é uma matriz de objetos que define um estado de fallback.

`Retry` ("Type": "Retry") é uma matriz de objetos que definem uma política de novas tentativas caso o estado encontre erros de tempo de execução.

Ramificação

Um estado `Choice` adiciona a lógica de ramificação a uma máquina de estado. `Choice` ("Type": "Choice") é um conjunto de regras que determina para qual estado a máquina de estado deve mudar em seguida.

Encadeamento

Um padrão de “encadeamento” descreve várias funções do Lambda conectadas a uma máquina de estado. Você pode usar o encadeamento para criar invocações de fluxo de trabalho reutilizáveis a partir do estado de `Tarefa` ("Type": "Task") de uma máquina de estado.

Paralelismo

Um estado `Parallel` adiciona lógica de paralelismo a uma máquina de estado. Você pode usar um estado `Parallel` ("Type": "Parallel") para criar ramificações paralelas de invocação na máquina de estado.

Paralelismo dinâmico

Um estado `Map` adiciona uma lógica “individual” de loop dinâmico a uma máquina de estado. Você pode usar um estado `Map` ("Type": "Map") para executar um conjunto de etapas para cada elemento de uma matriz de entrada em uma máquina de estado. Embora o estado `Parallel` invoque várias ramificações de etapas usando a mesma entrada, um estado `Map` invocará as mesmas etapas para várias entradas da matriz.

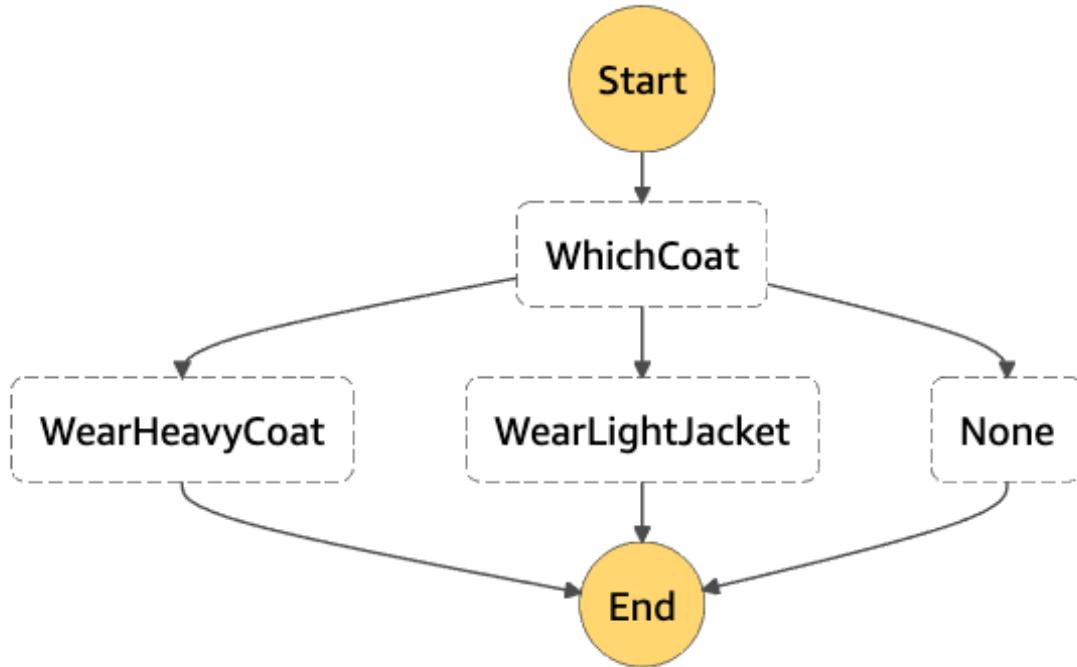
Além dos padrões de aplicação, o Step Functions oferece suporte a vários [padrões de integração de serviços](#), incluindo a capacidade de pausar um fluxo de trabalho para aprovação humana, ou chamar um sistema herdado ou outro terceiro.

Exemplo de padrão de aplicação de ramificação

No exemplo a seguir, a máquina de estado `WhichCoat` estabelecida na definição da Amazon States Language (ASL) mostra um padrão de aplicação de ramificação com um estado `Choice` ("Type": "Choice"). Se a condição de um dos três estados `choice` for atendida, a função do Lambda será invocada como `Tarefa`:

1. O estado `WearHeavyCoat` invocará a função do Lambda `wear_heavy_coat` e retornará uma mensagem.
2. O estado `WearLightJacket` invocará a função do Lambda `wear_light_jacket` e retornará uma mensagem.
3. O estado `None` invocará a função do Lambda `no_jacket` e retornará uma mensagem.

A máquina de estado `WhichCoat` tem a seguinte estrutura:



Example Exemplo de definição da Amazon States Language

A seguinte definição da Amazon States Language da máquina de estado `WhichCoat` usa um [objeto de contexto](#) Variable chamado `Weather`. Se uma das três condições em `StringEquals` for atendida, a função do Lambda definida no [nome do recurso da Amazon \(ARN\) do campo Resource](#) será invocada.

```
{  
    "Comment": "Coat Indicator State Machine",  
    "StartAt": "WhichCoat",  
    "States": {  
        "WhichCoat": {  
            "Type": "Choice",  
            "Choices": [  
                {  
                    "Variable": "$.Weather",  
                    "StringEquals": "FREEZING",  
                    "Next": "WearHeavyCoat"  
                },  
                {  
                    "Variable": "$.Weather",  
                    "StringEquals": "COOL",  
                    "Next": "WearLightJacket"  
                },  
                {  
                    "Variable": "$.Weather",  
                    "StringEquals": "WARM",  
                    "Next": "None"  
                }  
            ]  
        }  
    }  
}
```

```
        "Next": "None"
    }
]
},
"WearHeavyCoat":{
    "Type": "Task",
    "Resource": "arn:aws:lambda:us-west-2:01234567890:function:wear_heavy_coat",
    "End": true
},
"WearLightJacket":{
    "Type": "Task",
    "Resource": "arn:aws:lambda:us-west-2:01234567890:function:wear_light_jacket",
    "End": true
},
"None": {
    "Type": "Task",
    "Resource": "arn:aws:lambda:us-west-2:01234567890:function:no_coat",
    "End": true
}
}
```

Example Exemplo de função Python

A função do Lambda em Python (`wear_heavy_coat`) a seguir pode ser invocada para a máquina de estado definida no exemplo anterior. Se a máquina de estado `whichCoat` for igual a um valor de string `FREEZING`, a função `wear_heavy_coat` será invocada do Lambda, e o usuário receberá a mensagem que corresponde à função: “Use um casaco pesado hoje”.

```
from __future__ import print_function

import datetime

def wear_heavy_coat(message, context):
    print(message)

    response = {}
    response['Weather'] = message['Weather']
    response['Timestamp'] = datetime.datetime.now().strftime("%Y-%m-%d %H-%M-%S")
    response['Message'] = 'You should wear a heavy coat today.'

    return response
```

Example Exemplo de dados de invocação

Os dados de entrada a seguir executam o estado `WearHeavyCoat` que invoca a função do Lambda `wear_heavy_coat` quando a variável `Weather` é igual a um valor string de `FREEZING`.

```
{
    "Weather": "FREEZING"
}
```

Para obter mais informações, consulte [Criar uma máquina de estado do do que usa o Lambda de Step Functions](#) no AWS Step Functions Guia do desenvolvedor.

Gerenciar máquinas de estado no console do Lambda

Você pode usar o console do Lambda para visualizar detalhes sobre as máquinas de estado do Step Functions e as funções do Lambda usadas por elas.

Seções

- [Visualizar detalhes da máquina de estado \(p. 485\)](#)
- [Editar uma máquina de estado \(p. 485\)](#)
- [Executar uma máquina de estado \(p. 486\)](#)

Visualizar detalhes da máquina de estado

O console do Lambda exibe uma lista de máquinas de estado na região da AWS atual que contêm pelo menos uma etapa de fluxo de trabalho que invoca uma função do Lambda.

Escolha uma máquina de estado para visualizar uma representação gráfica do fluxo de trabalho. As etapas destacadas em azul representam funções do Lambda. Use os controles do gráfico para ampliar e diminuir o zoom e para centralizar o gráfico.

Note

Quando uma função do Lambda é [referenciada dinamicamente com JsonPath](#) na definição da máquina de estado, os detalhes da função não podem ser exibidos no console do Lambda. Em vez disso, o nome da função será listado como Dynamic reference (Referência dinâmica), e as etapas correspondentes no gráfico serão desabilitadas.

Como visualizar detalhes da máquina de estado

1. Abra a [página máquinas de estado do Step Functions](#) no console do Lambda.
2. Escolha uma máquina de estado.

`<result>`

O console do Lambda abrirá a página Details (Detalhes).

`</result>`

Para obter mais informações, consulte [Step Functions](#) no Guia do desenvolvedor do AWS Step Functions.

Editar uma máquina de estado

Se desejar editar uma máquina de estado, no Lambda, abra a página Edit definition (Editar definição) do console do Step Functions.

Como editar uma máquina de estado

1. Abra a [página da máquina de estado do Step Functions](#) no console do Lambda.
2. Escolha uma máquina de estado.
3. Selecione Edit.

O console do Step Functions abrirá a página Edit definition (Editar definição).

4. Edite a máquina de estado e escolha Save (Salvar).

Para obter mais informações sobre a edição de máquinas de estado, consulte a [linguagem de máquina de estado do Step Functions](#) no Guia do desenvolvedor do AWS Step Functions.

Executar uma máquina de estado

Se desejar executar uma máquina de estado, no Lambda, abra a página New execution (Nova execução) do console do Step Functions.

Como executar uma máquina de estado

1. Abra a [página máquinas de estado do Step Functions](#) no console do Lambda.
2. Escolha uma máquina de estado.
3. Selecione Execute (Executar).

O console do Step Functions abrirá a página New execution (Nova execução) .

4. (Opcional) Edite a máquina de estado e escolha Start execution (Iniciar execução).

Para obter mais informações sobre a execução de máquinas de estado, consulte os [conceitos de execução da máquina de estado do Step Functions](#) no Guia do desenvolvedor do AWS Step Functions.

Exemplos de orquestração com o Step Functions

Na máquina de estado do Step Functions, todo trabalho é realizado por [Tasks](#). Uma Task executa trabalhos usando uma atividade, uma função do Lambda ou transmitindo parâmetros para as ações de API de outras [Integrações de serviços da AWS para o Step Functions compatíveis](#).

Seções

- [Configuração de uma função do Lambda como tarefa. \(p. 486\)](#)
- [Configurar uma máquina de estado como uma origem de evento \(p. 487\)](#)
- [Manipular erros de função e de serviço \(p. 488\)](#)
- [AWS CloudFormation e AWS SAM \(p. 488\)](#)

Configuração de uma função do Lambda como tarefa.

O Step Functions pode invocar funções do Lambda diretamente de um estado de Task em uma definição da [Amazon States Language](#).

```
...  
  "MyStateName":{  
    "Type": "Task",  
    "Resource": "arn:aws:lambda:us-west-2:01234567890:function:my_lambda_function",  
  
    "End": true  
  ...  
}
```

Você pode criar um estado de Task que invoque a função do Lambda com a entrada para a máquina de estado ou qualquer documento JSON.

Example [event.json](#): entrada para [função de erro aleatório \(p. 500\)](#)

```
{
```

```
        "max-depth": 10,  
        "current-depth": 0,  
        "error-rate": 0.05  
    }
```

Configurar uma máquina de estado como uma origem de evento

Você pode criar uma máquina de estado do Step Functions que invoque uma função do Lambda. O exemplo a seguir mostra um estado de Task que invoca a versão 1 de uma função chamada `my-function` com uma carga de evento de três chaves. Quando a função retorna uma resposta correta, a máquina de estado continua para a próxima tarefa.

Example Exemplo de máquina de estado

```
...  
    "Invoke": {  
        "Type": "Task",  
        "Resource": "arn:aws:states:::lambda:invoke",  
        "Parameters": {  
            "FunctionName": "arn:aws:lambda:us-east-2:123456789012:function:my-function:1",  
            "Payload": {  
                "max-depth": 10,  
                "current-depth": 0,  
                "error-rate": 0.05  
            }  
        },  
        "Next": "NEXT_STATE",  
        "TimeoutSeconds": 25  
    }
```

Permissions

A máquina de estado precisa de permissão para chamar a API do Lambda para invocar uma função. Para conceder permissão a ela, adicione a política gerenciada da AWS, `AWSLambdaRole`, ou uma política em linha com o escopo de função da função dele. Para obter mais informações, consulte [How AWS Step Functions Works with IAM](#) no Guia do desenvolvedor do AWS Step Functions.

Os parâmetros `FunctionName` e `Payload` mapeiam para parâmetros na operação API [Invoke \(p. 875\)](#). Além disso, também é possível especificar os parâmetros `InvocationType` e `ClientContext`. Por exemplo, para invocar a função de forma assíncrona e continuar para o próximo estado sem esperar por um resultado, é possível definir `InvocationType` como `Event`:

```
"InvocationType": "Event"
```

Em vez de codificar a carga do evento na definição da máquina de estado, é possível usar a entrada da execução da máquina de estado. O exemplo a seguir usa a entrada especificada quando você executa a máquina de estado como carga do evento:

```
"Payload.$": "$"
```

Também é possível invocar uma função de forma assíncrona e esperar que ela faça um retorno de chamada com o AWS SDK. Para fazer isso, defina o recurso do estado como `arn:aws:states:::lambda:invoke.waitForTaskToken`.

Para obter mais informações, consulte [Invoke Lambda with Step Functions](#) no Guia do desenvolvedor do AWS Step Functions.

Manipular erros de função e de serviço

Quando a função ou o serviço do Lambda retorna um erro, é possível tentar a invocação novamente ou continuar para um estado diferente com base no tipo de erro.

O exemplo a seguir mostra uma tarefa de invocação que tenta novamente em exceções de API do Lambda de série 5XX (`ServiceException`), controles de utilização (`TooManyRequestsException`), erros de tempo de execução (`Lambda.Unknown`) e um erro definido por função chamado `function.MaxDepthError`. Ela também detecta um erro chamado `function.DoublesRolledError` e continua para um estado chamado `CaughtException` quando o erro ocorre.

Example Exemplo de padrão de detecção e nova tentativa

```
...
"Invoke": {
    "Type": "Task",
    "Resource": "arn:aws:states:::lambda:invoke",
    "Retry": [
        {
            "ErrorEquals": [
                "function.MaxDepthError",
                "Lambda.TooManyRequestsException",
                "Lambda.ServiceException",
                "Lambda.Unknown"
            ],
            "MaxAttempts": 5
        }
    ],
    "Catch": [
        {
            "ErrorEquals": [ "function.DoublesRolledError" ],
            "Next": "CaughtException"
        }
    ],
    "Parameters": {
        "FunctionName": "arn:aws:lambda:us-east-2:123456789012:function:my-function:1",
        ...
    }
}
```

Para detectar erros de função ou tentar novamente, crie um tipo de erro personalizado. O nome do tipo de erro deve corresponder ao `errorType` na resposta de erro formatada que o Lambda retorna quando um erro é lançado.

Para obter mais informações sobre o tratamento de erros no Step Functions, consulte [Handling Error Conditions Using a Step Functions State Machine](#) no Guia do desenvolvedor do AWS Step Functions.

AWS CloudFormation e AWS SAM

Você pode definir máquinas de estado usando um modelo do AWS CloudFormation com o AWS Serverless Application Model (AWS SAM). Ao usar o AWS SAM, é possível definir a máquina de estado em linha no modelo ou em um arquivo separado. O exemplo a seguir mostra uma máquina de estado invocando uma função do Lambda que manipula erros. Ela se refere a um recurso de função definido no mesmo modelo (não mostrado).

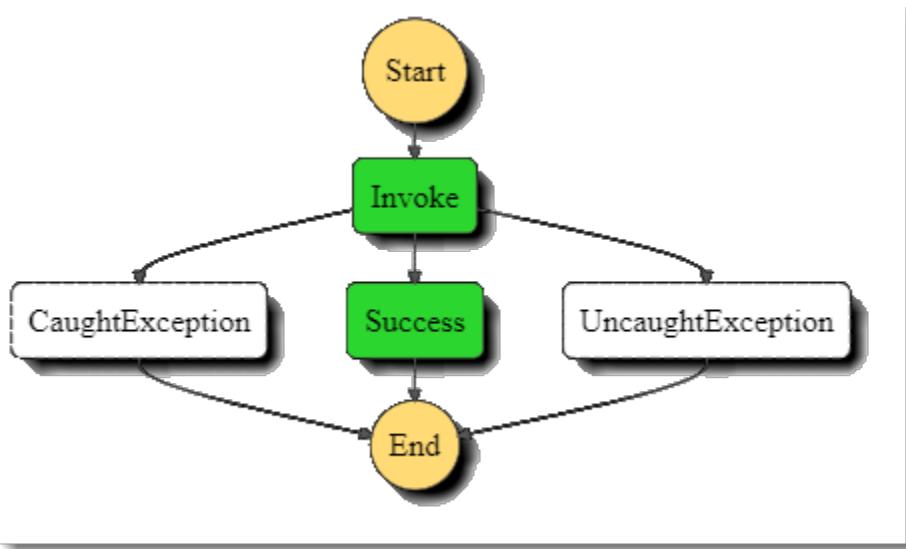
Example Exemplo de padrão de ramificação em template.yml

```
AWSTemplateFormatVersion: '2010-09-09'
```

```
Transform: 'AWS::Serverless-2016-10-31'
Description: An AWS Lambda application that uses AWS Step Functions.

Resources:
  statemachine:
    Type: AWS::Serverless::StateMachine
    Properties:
      DefinitionSubstitutions:
        FunctionArn: !GetAtt function.Arn
      Payload: |
        {
          "max-depth": 5,
          "current-depth": 0,
          "error-rate": 0.2
        }
    Definition:
      StartAt: Invoke
      States:
        Invoke:
          Type: Task
          Resource: arn:aws:states:::lambda:invoke
          Parameters:
            FunctionName: "${FunctionArn}"
            Payload: "${Payload}"
            InvocationType: Event
        Retry:
          - ErrorEquals:
              - function.MaxDepthError
              - function.MaxDepthError
              - Lambda.TooManyRequestsException
              - Lambda.ServiceException
              - Lambda.Unknown
            IntervalSeconds: 1
            MaxAttempts: 5
        Catch:
          - ErrorEquals:
              - function.DoublesRolledError
            Next: CaughtException
          - ErrorEquals:
              - States.ALL
            Next: UncaughtException
            Next: Success
        CaughtException:
          Type: Pass
          Result: The function returned an error.
          End: true
        UncaughtException:
          Type: Pass
          Result: Invocation failed.
          End: true
        Success:
          Type: Pass
          Result: Invocation succeeded!
          End: true
      Events:
        scheduled:
          Type: Schedule
          Properties:
            Description: Run every minute
            Schedule: rate(1 minute)
        Type: STANDARD
        Policies:
          - AWSLambdaRole
        ...
      
```

Isso cria uma máquina de estado com a seguinte estrutura:



Para obter mais informações, consulte [AWS::Serverless::StateMachine](#) no AWS Serverless Application Model Guia do desenvolvedor.

Aplicativos de exemplo do Lambda

O repositório do GitHub para este guia inclui aplicações de exemplo que demonstram o uso de várias linguagens e serviços da AWS. Cada aplicativo de exemplo inclui scripts para fácil implantação e limpeza, um modelo do AWS SAM e recursos de suporte.

Node.js

Aplicações de exemplo do Lambda em Node.js

- [blank-nodejs](#): uma função do Node.js que mostra o uso do registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK.
- [nodejs-apig](#): uma função com endpoint de API pública que processa um evento do API Gateway e retorna uma resposta HTTP.
- [rds-mysql](#): uma função que retransmite consultas para um banco de dados MySQL para RDS. Este exemplo inclui uma VPC privada e uma instância de banco de dados configurada com uma senha no AWS Secrets Manager.
- [efs-nodejs](#): uma função que usa um sistema de arquivos do Amazon EFS em uma Amazon VPC. Esse exemplo inclui uma VPC, um sistema de arquivos, destinos de montagem e ponto de acesso configurado para uso com o Lambda.
- [list-manager](#): uma função processa eventos de um fluxo de dados do Amazon Kinesis e atualiza listas agregadas no Amazon DynamoDB. A função armazena um registro de cada evento em um banco de dados MySQL para RDS em uma VPC privada. Este exemplo inclui uma VPC privada com um endpoint da VPC para o DynamoDB e uma instância de banco de dados.
- [error-processor](#): uma função do Node.js gera erros para uma porcentagem especificada de solicitações. Uma assinatura do CloudWatch Logs invoca uma segunda função quando um erro é registrado. A função do processador usa o AWS SDK para coletar detalhes sobre a solicitação e os armazena em um bucket do Amazon S3.

Python

Aplicativos do Lambda de exemplo do em Python

- [blank-python](#): uma função Python que mostra o uso de registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK.

Ruby

Aplicações de exemplo do Lambda em Ruby

- [blank-ruby](#): uma função do Ruby que mostra o uso de registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK.
- [Ruby Code Samples for AWS Lambda](#): exemplos de código escritos em Ruby que demonstram como interagir com o AWS Lambda.

Java

Aplicações de exemplo do Lambda em Java

- [blank-java](#): uma função Java que mostra o uso das bibliotecas Java do Lambda, registro em log, variáveis de ambiente, camadas, rastreamento do AWS X-Ray, testes de unidade e do AWS SDK.

- [java-basic](#): uma função Java mínima com testes de unidade e configuração de registro em log variável.
- [java-events](#)— Uma função Java mínima que usa a versão mais recente (3.0.0 e mais recente) da [aws-lambda-java-events \(p. 599\)](#) biblioteca. Estes exemplos não exigem o AWSSDK como dependência.
- [s3-java](#): uma função Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.

Go

O Lambda fornece as seguintes aplicações de exemplo para o tempo de execução do Go:

Aplicativos do Lambda de exemplo do em Go

- [blank-go](#): uma função do Go que mostra o uso das bibliotecas do Go do Lambda, o registro em log, as variáveis de ambiente e o AWS SDK.

C#

Aplicativos do Lambda de exemplo do em C#

- [blank-csharp](#): uma função do C# que mostra o uso das bibliotecas .NET do Lambda, do registro em log, das variáveis de ambiente, do rastreamento do AWS X-Ray, dos testes de unidade e do AWS SDK.
- [ec2-spot](#): uma função que gerencia solicitações de instâncias spot no Amazon EC2.

PowerShell

O Lambda fornece os seguintes aplicações de exemplo para o tempo de execução do PowerShell:

- [blank-powershell](#): uma função do PowerShell que mostra o uso do registro em log, as variáveis de ambiente e o AWS SDK.

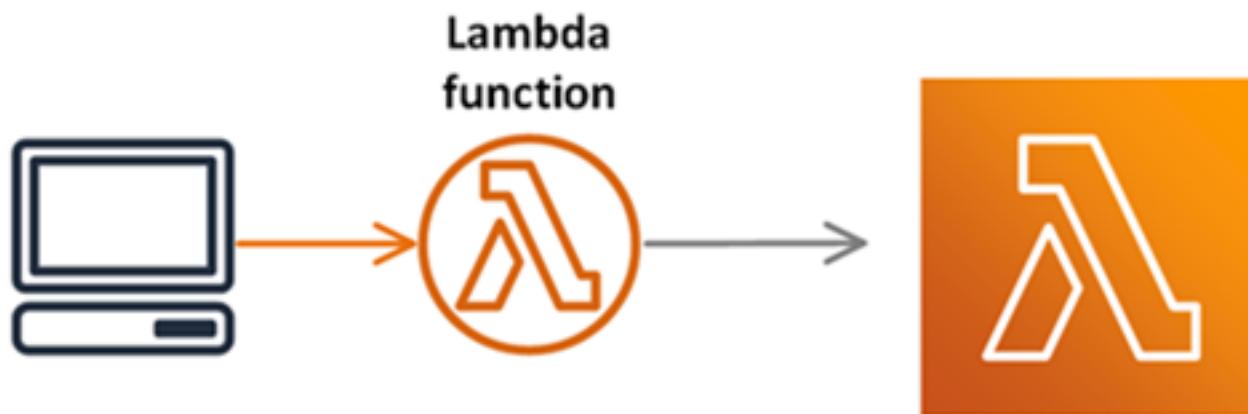
Para implantar um aplicativo de exemplo, siga as instruções no arquivo README. Para saber mais sobre a arquitetura e casos de uso de um aplicativo, leia os tópicos neste capítulo.

Tópicos

- [Aplicativo de exemplo de função em branco para o AWS Lambda \(p. 493\)](#)
- [Aplicativo de exemplo de processador de erros para o AWS Lambda \(p. 500\)](#)
- [Aplicativo de exemplo do gerenciador de listas para o AWS Lambda \(p. 505\)](#)

Aplicativo de exemplo de função em branco para o AWS Lambda

A aplicação de exemplo de função em branco é uma aplicação inicial que demonstra operações comuns no Lambda com uma função que chama a API do Lambda. Ele mostra o uso de registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK. Explore essa aplicação para saber como criar funções do Lambda em sua linguagem de programação, ou usá-lo como ponto de partida para seus próprios projetos.



As variantes deste aplicativo de exemplo estão disponíveis nas seguintes linguagens:

Variants

- Node.js: [blank-nodejs](#).
- Python: [blank-python](#).
- Ruby: [blank-ruby](#).
- Java: [blank-java](#).
- Go: [blank-go](#).
- C#: [blank-csharp](#).
- PowerShell – [blank-powershell](#).

Os exemplos neste tópico destacam o código da versão Node.js, mas os detalhes são geralmente aplicáveis a todas as variantes.

Você pode implantar a amostra em alguns minutos com a AWS CLI e o AWS CloudFormation . Siga as instruções no [README](#) para fazer download, configurar e implantá-la na conta.

Seções

- [Arquitetura e código do manipulador \(p. 494\)](#)
- [Automação de implantação com o AWS CloudFormation e a AWS CLI \(p. 495\)](#)
- [Instrumentação com o AWS X-Ray \(p. 497\)](#)
- [Gerenciamento de dependências com camadas \(p. 498\)](#)

Arquitetura e código do manipulador

O aplicativo de exemplo consiste em um código de função, um modelo do AWS CloudFormation e recursos de suporte. Ao implantar o exemplo, você usa os seguintes serviços da AWS:

- **AWS Lambda**: executa o código da função, envia logs para o CloudWatch Logs e envia dados de rastreamento para o X-Ray. A função também chama a API do Lambda para obter detalhes sobre as cotas e o uso da conta na região atual.
- **AWS X-Ray**: coleta de dados de rastreamento, indexa rastreamentos para pesquisa e gera um mapa de serviço.
- **Amazon CloudWatch**— armazena logs e métricas.
- **AWS Identity and Access Management(IAM)**— concede permissão.
- **Amazon Simple Storage Service (Amazon S3)**— armazena o pacote de implantação da função durante a implantação.
- **AWS CloudFormation**: cria recursos da aplicação e implanta o código da função.

Aplicam-se taxas padrão para cada serviço. Para obter mais informações, consulte [Definição de preço do AWS](#).

O código da função mostra um fluxo de trabalho básico para processar um evento. O manipulador pega um evento do Amazon Simple Queue Service (Amazon SQS) como entrada e percorre pelos registros que ele contém, registrando em log o conteúdo de cada mensagem. Ele registra o conteúdo do evento, o objeto de contexto e as variáveis de ambiente. Depois, ele faz uma chamada com o AWS SDK e passa a resposta de volta para o tempo de execução do Lambda.

Example [blank-nodejs/function/index.js](#): código do handler

```
// Handler
exports.handler = async function(event, context) {
  event.Records.forEach(record => {
    console.log(record.body)
  })
  console.log('## ENVIRONMENT VARIABLES: ' + serialize(process.env))
  console.log('## CONTEXT: ' + serialize(context))
  console.log('## EVENT: ' + serialize(event))

  return getAccountSettings()
}

// Use SDK client
var getAccountSettings = function(){
  return lambda.getAccountSettings().promise()
}

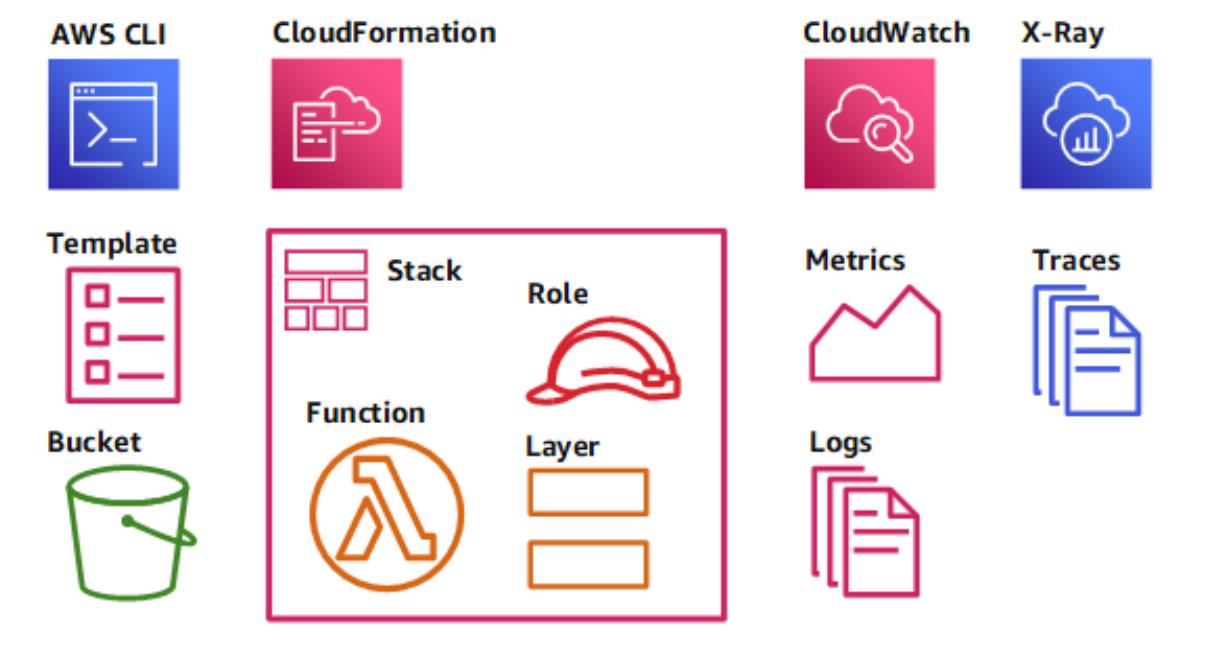
var serialize = function(object) {
  return JSON.stringify(object, null, 2)
}
```

Os tipos de entrada/saída para o manipulador e o suporte para programação assíncrona variam de acordo com o tempo de execução. Neste exemplo, o método do manipulador é `async`, portanto, em Node.js isso significa que ele deve retornar uma promessa ao tempo de execução. O tempo de execução do Lambda espera que a promessa seja resolvida e retorna a resposta ao invocador. Se o código de função ou o cliente AWS SDK retornar um erro, o tempo de execução formatará o erro em um documento JSON e retornará isso.

A amostra de aplicação não inclui uma fila do Amazon SQS para enviar eventos, mas usa um evento do Amazon SQS ([event.json](#)) para ilustrar como os eventos são processados. Para adicionar uma fila do Amazon SQS à aplicação, consulte [O uso do AWS LambdaCom o Amazon SQS \(p. 465\)](#).

Automação de implantação com o AWS CloudFormation e a AWS CLI

Os recursos do aplicativo de exemplo são definidos em um modelo do AWS CloudFormation e implantados com a AWS CLI. O projeto inclui scripts de shell simples que automatizam o processo de configuração, implantação, invocação e destruição do aplicativo.



O modelo de aplicativo usa um tipo de recurso AWS Serverless Application Model (AWS SAM) para definir o modelo. O AWS SAM simplifica a criação de modelos para aplicativos sem servidor automatizando a definição de funções de execução, APIs e outros recursos.

O modelo define os recursos na pilha de aplicativos. Isso inclui a função, sua função de execução e uma camada do Lambda que fornece as dependências da biblioteca da função. A pilha não inclui o bucket que a AWS CLI usa durante a implantação nem o grupo de logs do CloudWatch Logs.

Example [blank-nodejs/template.yml](#): recursos sem servidor

```
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:
      Handler: index.handler
      Runtime: nodejs12.x
      CodeUri: function/
      Description: Call the AWS Lambda API
      Timeout: 10
      # Function's execution role
      Policies:
        - AWSLambdaBasicExecutionRole
```

```
- AWSLambda_ReadOnlyAccess
- AWSXrayWriteOnlyAccess
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:
    - nodejs12.x
```

Quando você implanta o aplicativo, o AWS CloudFormation aplica a transformação do AWS SAM ao modelo para gerar um modelo do AWS CloudFormation com tipos padrão, como `AWS::Lambda::Function` e `AWS::IAM::Role`.

Example modelo processado

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "An AWS Lambda application that calls the Lambda API.",
  "Resources": {
    "function": {
      "Type": "AWS::Lambda::Function",
      "Properties": {
        "Layers": [
          {
            "Ref": "libs32xmpl61b2"
          }
        ],
        "TracingConfig": {
          "Mode": "Active"
        },
        "Code": {
          "S3Bucket": "lambda-artifacts-6b000xmpl1e9bf2a",
          "S3Key": "3d3axmpl473d249d039d2d7a37512db3"
        },
        "Description": "Call the AWS Lambda API",
        "Tags": [
          {
            "Value": "SAM",
            "Key": "lambda:createdBy"
          }
        ],
      }
    }
  }
}
```

Neste exemplo, a propriedade `Code` especifica um objeto em um bucket do Amazon S3. Isso corresponde ao caminho local na propriedade `CodeUri` no modelo de projeto:

```
CodeUri: function/.
```

Para fazer upload dos arquivos do projeto no Amazon S3, o script de implantação usa comandos na AWS CLI. O comando `cloudformation package` pré-processa o modelo, faz upload de artefatos e substitui caminhos locais por locais de objetos do Amazon S3. O comando `cloudformation deploy` implanta o modelo processado com um conjunto de alterações do AWS CloudFormation.

Example `blank-nodejs/3-deploy.sh`: empacotar e implantar

```
#!/bin/bash
```

```
set -eo pipefail
ARTIFACT_BUCKET=$(cat bucket-name.txt)
aws cloudformation package --template-file template.yml --s3-bucket $ARTIFACT_BUCKET --
output-template-file out.yml
aws cloudformation deploy --template-file out.yml --stack-name blank-nodejs --capabilities
CAPABILITY_NAMED_IAM
```

Da primeira vez que você executar esse script, ele criará uma pilha do AWS CloudFormation chamada `blank-nodejs`. Se fizer alterações no modelo ou no código da função, você poderá executá-la novamente para atualizar a pilha.

O script de limpeza ([blank-nodejs/5-cleanup.sh](#)) exclui a pilha e, opcionalmente, exclui o bucket de implantação e os logs de função.

Instrumentação com o AWS X-Ray

A função de exemplo é configurada para rastreamento com [AWS X-Ray](#). Com o modo de rastreamento definido como ativo, o Lambda registra informações de temporização para um subconjunto de invocações e as envia para o X-Ray. O X-Ray processa os dados para gerar um Mapa de serviço que mostra um nó de cliente e dois nós de serviço:



O primeiro nó de serviço (`AWS::Lambda`) representa o serviço do Lambda, que valida a solicitação de invocação e o envia para a função. O segundo nó, `AWS::Lambda::Function`, representa a própria função.

Para registrar detalhes adicionais, a função de exemplo usa o X-Ray SDK. Com alterações mínimas no código de função, o X-Ray SDK registra detalhes sobre chamadas feitas com o AWS SDK para serviços da AWS.

Example [blank-nodejs/function/index.js](#): instrumentação

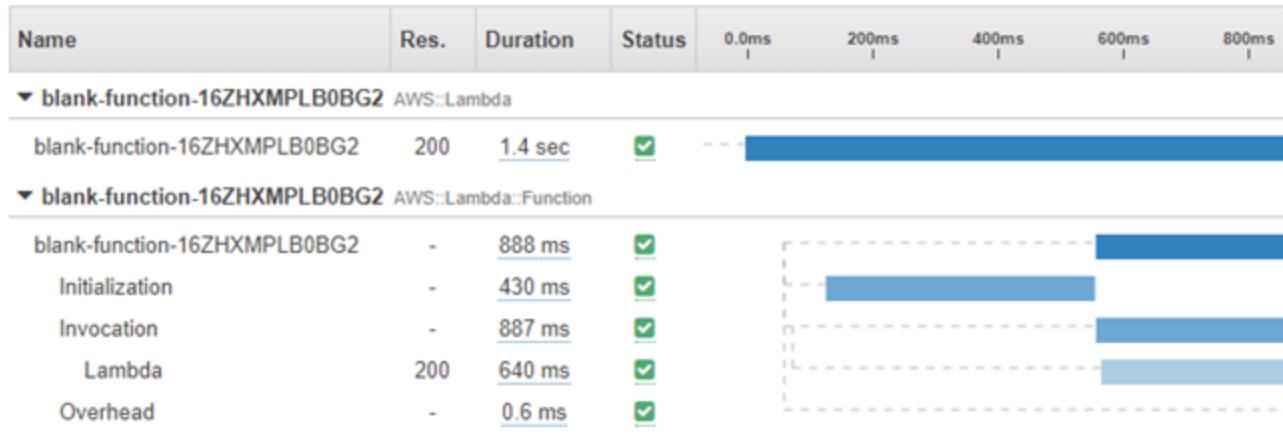
```
const AWSXRay = require('aws-xray-sdk-core')
const AWS = AWSXRay.captureAWS(require('aws-sdk'))

// Create client outside of handler to reuse
const lambda = new AWS.Lambda()
```

Instrumentar o cliente AWS SDK adiciona um nó extra ao mapa de serviços e mais detalhes em rastreamentos. Neste exemplo, o mapa de serviço mostra a função de exemplo chamando a API do Lambda para obter detalhes sobre armazenamento e uso de simultaneidade na região atual.



O rastreamento mostra detalhes de temporização para a invocação, com subsegmentos para inicialização da função, invocação e sobrecarga. O subsegmento de invocação tem um subsegmento para a chamada do AWS SDK para a operação de API GetAccountSettings.



Você pode incluir o X-Ray SDK e outras bibliotecas no pacote de implantação da função ou implantá-las separadamente em uma camada do Lambda. Para Node.js, Ruby e Python, o tempo de execução do Lambda inclui o AWS SDK no ambiente de execução.

Gerenciamento de dependências com camadas

Você pode instalar bibliotecas localmente e incluí-las no pacote de implantação que carregar no Lambda, mas isso tem suas desvantagens. Tamanhos de arquivo maiores geram tempos de implantação maiores e podem impedir que você teste alterações em seu código de função no console do Lambda. Para manter o pacote de implantação pequeno e evitar o upload de dependências que não foram alteradas, à aplicação de exemplo cria uma [camada do Lambda \(p. 85\)](#) e a associa à função.

Example [blank-nodejs/template.yml](#): camada de dependência

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs12.x
      CodeUri: function/
      Description: Call the AWS Lambda API
      Timeout: 10
```

```
# Function's execution role
Policies:
  - AWSLambdaBasicExecutionRole
  - AWSLambda_ReadOnlyAccess
  - AWSXrayWriteOnlyAccess
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:
    - nodejs12.x
```

O script `2-build-layer.sh` instala as dependências da função com o npm e as coloca em uma pasta com a [estrutura exigida pelo tempo de execução do Lambda \(p. 85\)](#).

Example [2-build-layer.sh](#): preparar a camada

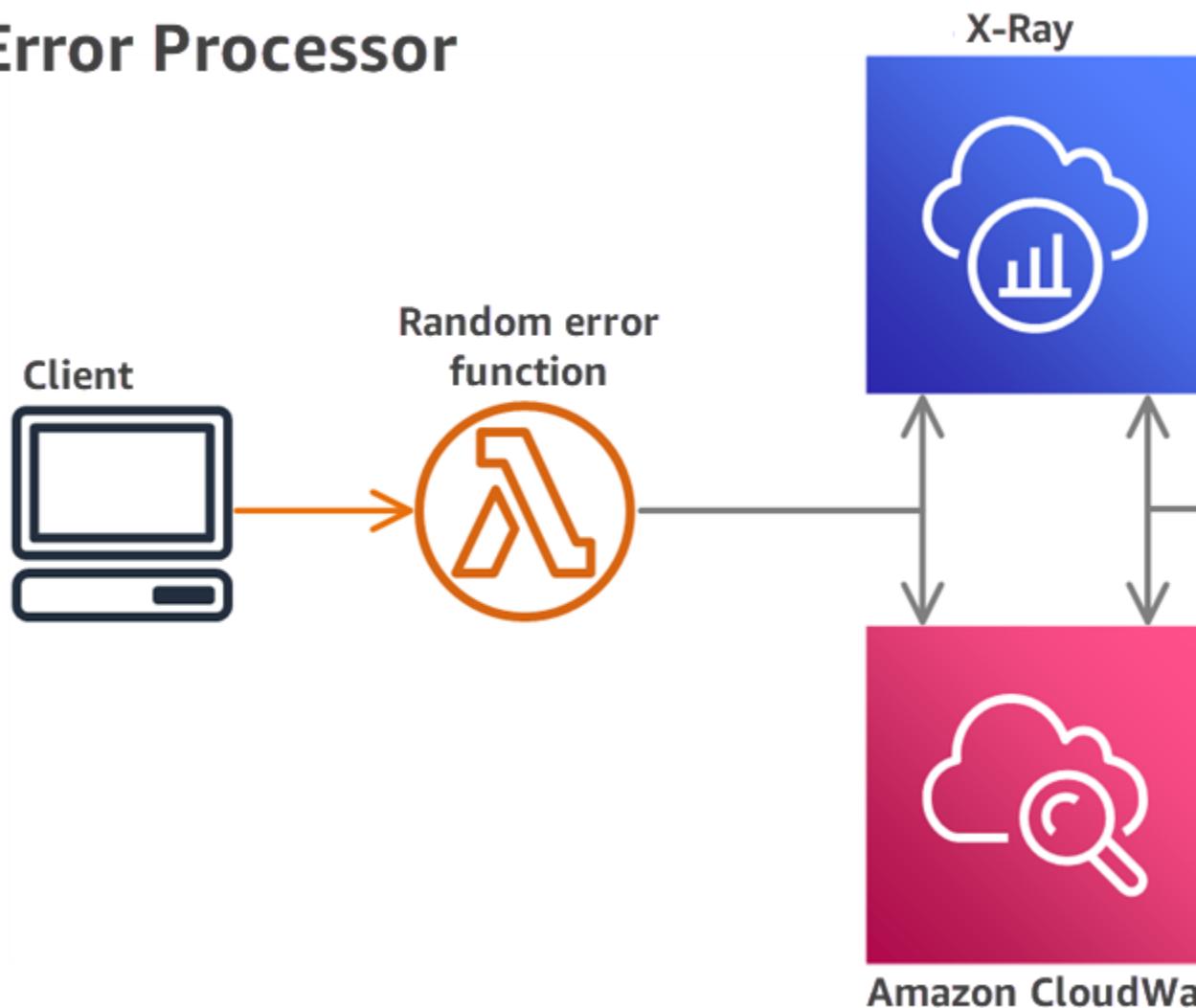
```
#!/bin/bash
set -eo pipefail
mkdir -p lib/nodejs
rm -rf node_modules lib/nodejs/node_modules
npm install --production
mv node_modules lib/nodejs/
```

Ao implantar o aplicativo de exemplo pela primeira vez, a AWS CLI empacota a camada separadamente do código de função e implanta ambos. Para implantações subsequentes, o arquivo de camada só será carregado se o conteúdo da pasta `lib` tiver sido alterado.

Aplicativo de exemplo de processador de erros para o AWS Lambda

A aplicação de exemplo do processador de erros demonstra o uso de AWS Lambda para processar eventos de uma [assinatura do Amazon CloudWatch Logs \(p. 322\)](#). O CloudWatch Logs permite invocar uma função do Lambda quando uma entrada de log corresponde a um padrão. A inscrição nesse aplicativo monitora o grupo de logs de uma função para verificar as entradas que contêm a palavra ERROR. Ela invoca uma função do Lambda do processador em resposta. A função de processador recupera o fluxo de logs e os dados de rastreamento completos para a solicitação que causou o erro e os armazena para uso posterior.

Error Processor



O código da função está disponível nos arquivos a seguir.

- Erro aleatório: [random-error/index.js](#)
- Processador: [processor/index.js](#)

Você pode implantar a amostra em alguns minutos com a AWS CLI e o AWS CloudFormation . Para fazer download, configurar e implantá-la na conta, siga as instruções no [README](#).

Seções

- [Estrutura de eventos e arquitetura \(p. 501\)](#)
- [Instrumentação com o AWS X-Ray \(p. 502\)](#)
- [AWS CloudFormationModelo do e recursos adicionais \(p. 503\)](#)

Estrutura de eventos e arquitetura

A aplicação de exemplo usa os seguintes serviços da AWS.

- AWS Lambda: executa o código da função, envia logs para o CloudWatch Logs e envia dados de rastreamento para o X-Ray.
- Amazon CloudWatch Logs: coleta logs e invoca uma função quando uma entrada de log corresponde a um padrão de filtro.
- AWS X-Ray: coleta de dados de rastreamento, indexa rastreamentos para pesquisa e gera um mapa de serviço.
- Amazon Simple Storage Service (Amazon S3) — Armazena artefatos de implantação e saída de aplicativo.

Aplicam-se taxas padrão para cada serviço.

Uma função do Lambda na aplicação gera erros aleatoriamente. Quando o CloudWatch Logs detecta a palavra `ERROR` nos logs da função, ele envia um evento para a função de processador para processamento.

Example CloudWatch Logs

```
{  
  "awslogs": {  
    "data": "H4sIAAAAAAAAHWQT0/DMAzFv0vEkbLYcdJkt4qVXmCDteIAm1DbZKjS  
+kdpB0Jo350MhsQFyVLsZ+unl/fJWje05asrPgbH5..."  
  }  
}
```

Quando decodificados, os dados contém detalhes sobre o evento de log. A função usa isso para identificar o fluxo de logs e analisa a mensagem de log para obter o ID da solicitação que causou o erro.

Example dados decodificado do evento CloudWatch Logs

```
{  
  "messageType": "DATA_MESSAGE",  
  "owner": "123456789012",  
  "logGroup": "/aws/lambda/lambda-error-processor-randomerror-1GD4SSDNACNP4",  
  "logStream": "2019/04/04/[$LATEST]63311769a9d742f19cedf8d2e38995b9",  
  "subscriptionFilters": [  
    "lambda-error-processor-subscription-15OPDVQ59CG07"  
  ],  
  "logEvents": [  
    {  
      "id": "34664632210239891980253245280462376874059932423703429141",  
      "timestamp": 1554415868243,  
      "message": "2019-04-04T22:11:08.243Z\t1d2c1444-efd1-43ec-  
b16e-8fb2d37508b8\terrOR\n"  
    }  
  ]  
}
```

}

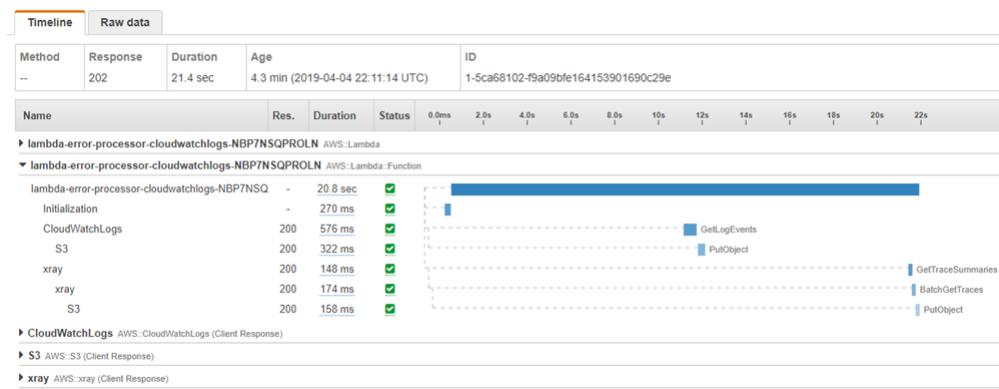
A função de processador usa informações do evento do CloudWatch Logs para fazer download da transmissão de logs e do rastreamento do X-Ray completos de uma solicitação que causou um erro. Ela armazena ambos em um bucket do Amazon S3. Para permitir que o fluxo de logs e o tempo de rastreamento sejam finalizados, a função aguarda por um curto período antes de acessar os dados.

Instrumentação com o AWS X-Ray

O aplicativo usa [AWS X-Ray \(p. 477\)](#) para rastrear chamadas de função e as chamadas que as funções fazem para [AWS Serviços da .](#) O X-Ray usa os dados de rastreamento que recebe das funções para criar um mapa de serviço que ajuda a identificar erros. O mapa de serviços a seguir mostra função de erro aleatório gerando erros para algumas solicitações. Ele também mostra a função do processador chamando X-Ray, CloudWatch Logs e Amazon S3.



As duas funções Node.js são configuradas para o rastreamento ativo no modelo, e são instrumentadas com o SDK do AWS X-Ray for Node.js no código. Com o rastreamento ativo, as tags do Lambda adicionam um cabeçalho de rastreamento a solicitações de entrada e envia um rastreamento com detalhes de tempo ao X-Ray. Além disso, a função de erro aleatório usa o X-Ray SDK para registrar o ID da solicitação e as informações do usuário nas anotações. As anotações são anexadas ao rastreamento e você pode usá-las para localizar o rastreamento de uma solicitação específica.



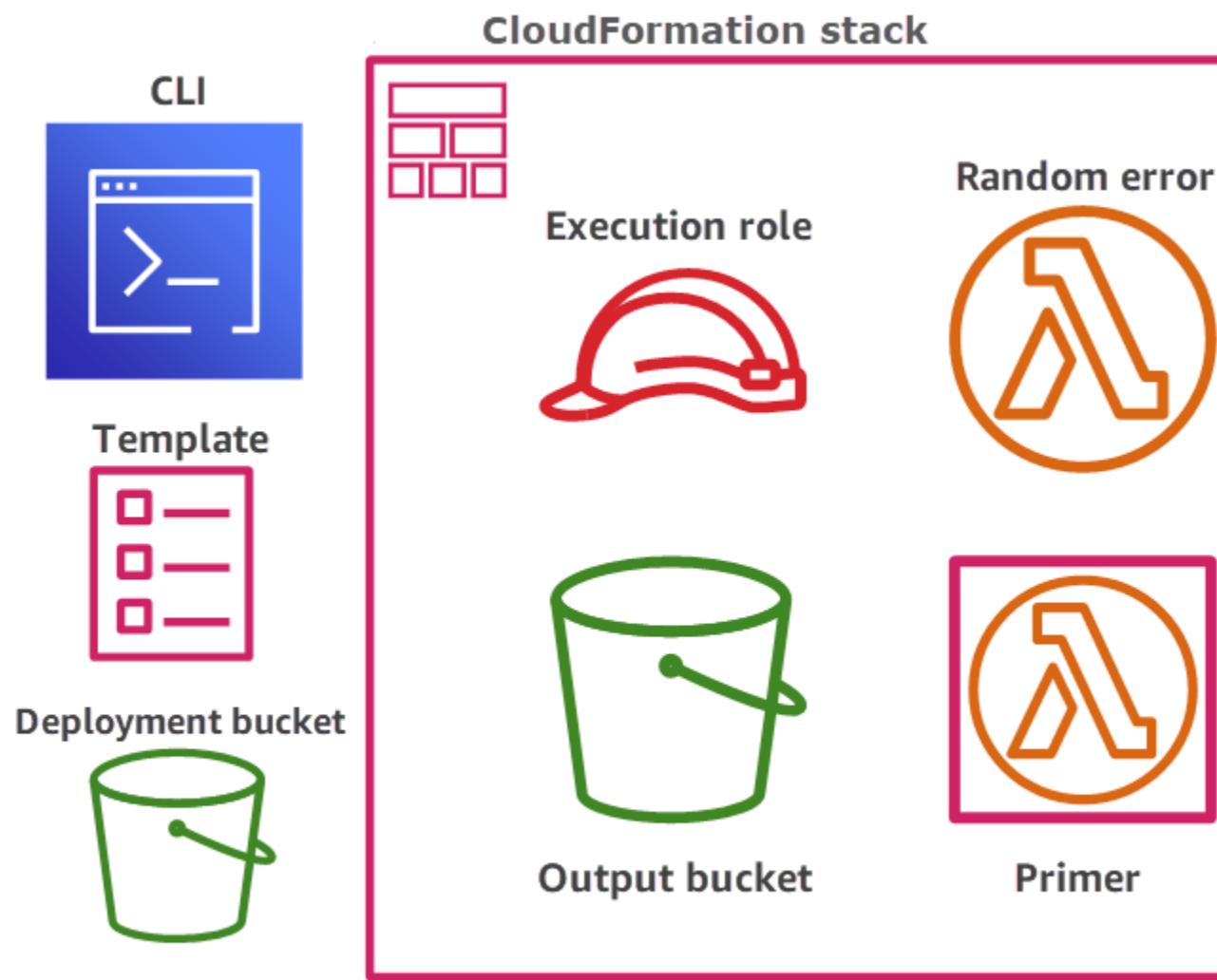
A função de processador obtém o ID da solicitação do evento do CloudWatch Logs e usa o AWS SDK for JavaScript para pesquisar essa solicitação no X-Ray. Ela usa clientes do AWS SDK, instrumentados com o X-Ray SDK, para fazer download do rastreamento e da transmissão de logs. Depois, ela os armazena no bucket de saída. O X-Ray SDK registra essas chamadas e elas aparecem como subsegmentos no rastreamento.

AWS CloudFormation Modelo do e recursos adicionais

O aplicativo é implementado em módulos do Node.js e implantado com um modelo do AWS CloudFormation e scripts de shell. O modelo cria a função de processador, a função de erro aleatório e os recursos de suporte a seguir.

- Função de execução: uma função do IAM que concede às funções permissão para acessar outros serviços da AWS.
- Função de introdução: uma função adicional que invoca a função de erro aleatório para criar um grupo de logs.
- Recurso personalizado: um recurso personalizado do AWS CloudFormation invoca a função de introdução durante a implantação para garantir que o grupo de logs exista.
- Inscrição no CloudWatch Logs: uma inscrição na transmissão de logs que aciona a função de processador quando a palavra ERROR é registrada.
- Política baseada em recursos: uma instrução de permissão na função de processador que permite que o CloudWatch Logs a invoque.
- Bucket do Amazon S3: um local de armazenamento para a saída da função de processador.

Visualize o [modelo de aplicação](#) no GitHub.



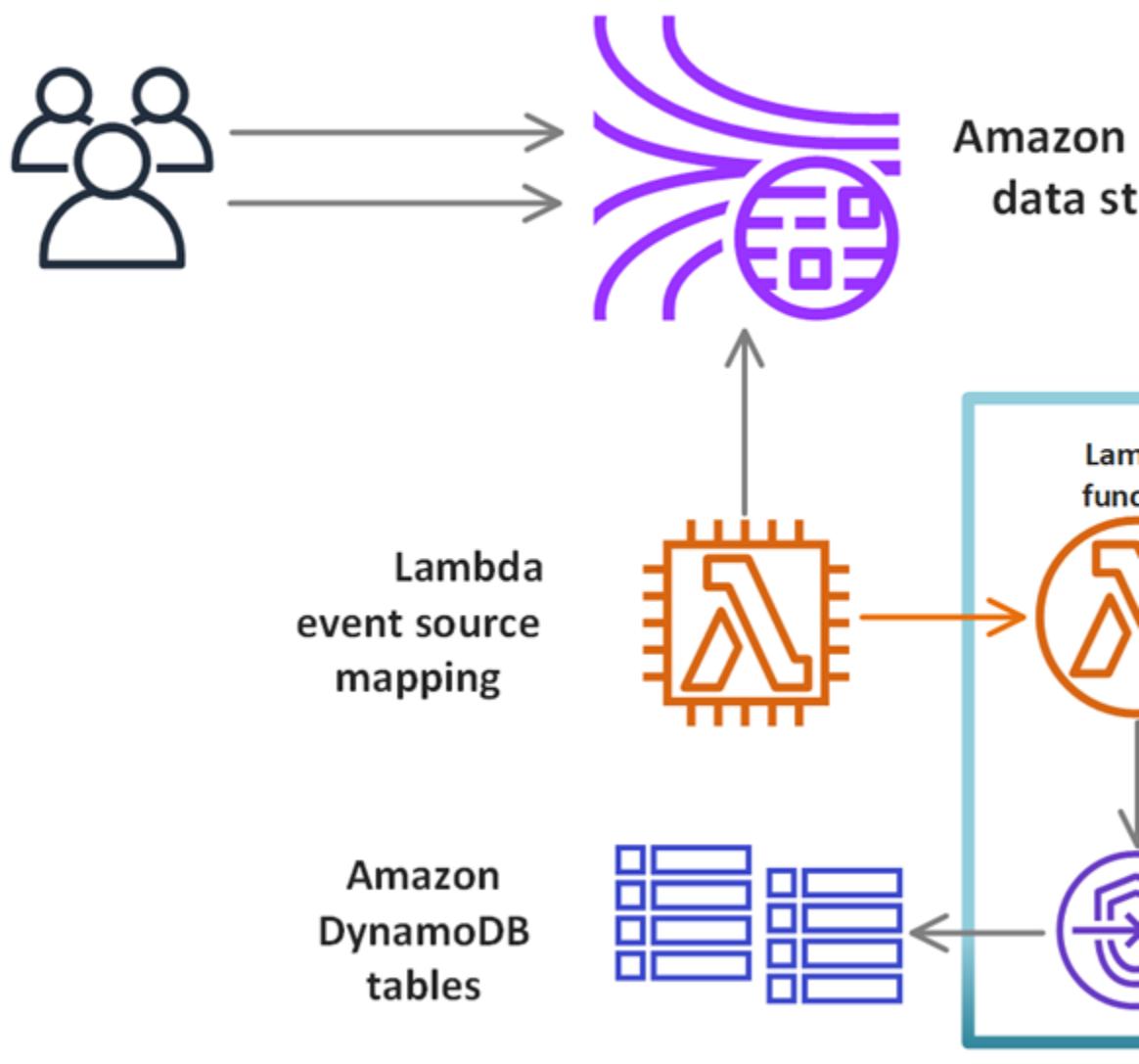
Para contornar uma limitação de integração do Lambda com o AWS CloudFormation, o modelo cria uma função adicional que é executada durante as implantações. Todas as funções do Lambda vêm com um grupo de logs do CloudWatch Logs que armazena a saída de execuções de funções. No entanto, o grupo de logs não é criado até que a função seja invocada pela primeira vez.

Para criar a inscrição, que depende da existência do grupo de logs, a aplicação usa uma terceira função do Lambda para invocar a função de erro aleatório. O modelo inclui o código para a função de introdução em linha. Um recurso personalizado do AWS CloudFormation a invoca durante a implantação. As propriedades `DependsOn` garantem que o fluxo de logs e a política baseada em recursos sejam criados antes da inscrição.

Aplicativo de exemplo do gerenciador de listas para o AWS Lambda

A aplicação de exemplo do gerenciador de listas demonstra o uso do AWS Lambda para processar registros em um fluxo de dados do Amazon Kinesis. Um mapeamento de fontes de eventos do Lambda lê registros da transmissão em lotes e invoca uma função do Lambda. A função usa informações dos registros para atualizar documentos no Amazon DynamoDB e armazena os registros que ela processa no Amazon Relational Database Service (Amazon RDS).

List manager application



Os clientes enviam registros para uma transmissão do Kinesis, que os armazena e os disponibiliza para processamento. A transmissão do Kinesis é usada como uma fila para armazenar os registros em buffer até que eles possam ser processados. Ao contrário de uma fila do Amazon SQS, os registros em uma transmissão do Kinesis não são excluídos após serem processados, portanto, vários consumidores podem processar os mesmos dados. Os registros no Kinesis também são processados em ordem e os itens da fila podem ser entregues fora de ordem. Os registros são excluídos do stream após 7 dias.

Além da função que processa eventos, o aplicativo inclui uma segunda função para executar tarefas administrativas no banco de dados. O código da função está disponível nos arquivos a seguir.

- Processador: [processor/index.js](#)
- Administrador de banco de dados: [dbadmin/index.js](#)

Você pode implantar a amostra em alguns minutos com a AWS CLI e o AWS CloudFormation . Para fazer download, configurar e implantá-la na conta, siga as instruções no [README](#).

Seções

- [Estrutura de eventos e arquitetura \(p. 506\)](#)
- [Instrumentação com o AWS X-Ray \(p. 507\)](#)
- [AWS CloudFormation Modelos de recursos adicionais \(p. 510\)](#)

Estrutura de eventos e arquitetura

A aplicação de exemplo usa os seguintes serviços da AWS:

- Kinesis: recebe eventos de clientes e os armazena temporariamente para processamento.
- AWS Lambda: lê a transmissão do Kinesis e envia eventos para o código do handler da função.
- DynamoDB: armazena listas geradas pela aplicação.
- Amazon RDS: armazena uma cópia dos registros processados em um banco de dados relacional.
- AWS Secrets Manager: armazena a senha do banco de dados.
- Amazon VPC: fornece uma rede local privada para comunicação entre a função e o banco de dados.

Pricing

Aplicam-se taxas padrão para cada serviço.

O aplicativo processa documentos JSON de clientes que contenham as informações necessárias para atualizar uma lista. Ele oferece suporte a dois tipos de lista: contagem e classificação. Uma contagem contém valores que serão adicionados ao valor atual da chave se ela existir. Cada entrada processada para um usuário aumenta o valor de uma chave na tabela especificada.

O exemplo a seguir mostra um documento que aumenta o valor de `xp` (pontos de experiência) para a lista de `stats` de um usuário.

Example registro: tipo de contagem

```
{  
  "title": "stats",  
  "user": "bill",  
  "type": "tally",  
  "entries": {  
    "xp": 83  
  }  
}
```

Uma classificação contém uma lista de entradas em que o valor é a ordem na qual eles são classificados. Uma classificação pode ser atualizada com valores diferentes que substituem o valor atual, em vez de incrementá-lo. O exemplo a seguir mostra uma classificação de filmes favoritos:

Example registro: tipo de classificação

```
{  
    "title": "favorite movies",  
    "user": "mike",  
    "type": "rank",  
    "entries": {  
        "blade runner": 1,  
        "the empire strikes back": 2,  
        "alien": 3  
    }  
}
```

Um [mapeamento de fontes de eventos \(p. 170\)](#) do Lambda lê registros da transmissão em lotes e invoca a função do processador. O evento recebido pelo manipulador da função contém uma matriz de objetos que contêm detalhes sobre um registro, como quando ele foi recebido, detalhes sobre o stream e uma representação codificada do documento de registro original.

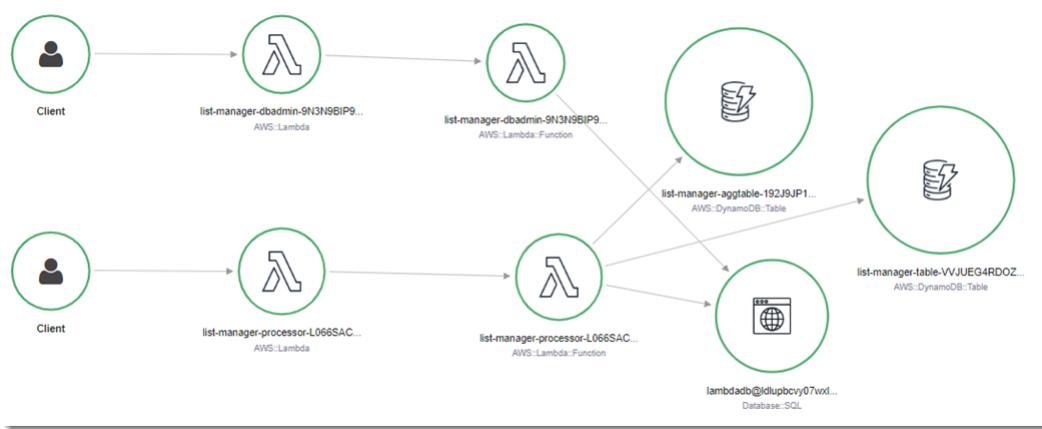
Example `events/kinesis.json`: registro

```
{  
    "Records": [  
        {  
            "kinesis": {  
                "kinesisSchemaVersion": "1.0",  
                "partitionKey": "0",  
                "sequenceNumber": "49598630142999655949899443842509554952738656579378741250",  
                "data": "  
eyJ0aXRsZSI6ICJmYXZvcml0ZSBtb3ZpZXMiLCAiDXNlcii6ICJyZGx5c2N0IiwgInR5cGUIoAiwmFuayIsICJlbnRyaWVzIjoge  
                "approximateArrivalTimestamp": 1570667770.615  
            },  
            "eventSource": "aws:kinesis",  
            "eventVersion": "1.0",  
            "eventID": "shardId-000000000000:49598630142999655949899443842509554952738656579378741250",  
            "eventName": "aws:kinesis:record",  
            "invokeIdentityArn": "arn:aws:iam::123456789012:role/list-manager-  
processorRole-7FYXMPFH7IUS",  
            "awsRegion": "us-east-2",  
            "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/list-manager-  
stream-87B3XMPFLF1AZ"  
        },  
        ...  
    ]  
}
```

Quando ele é decodificado, os dados contêm um registro. A função usa o registro para atualizar a lista do usuário e uma lista agregada que armazena valores acumulados em todos os usuários. Ele também armazena uma cópia do evento no banco de dados do aplicativo.

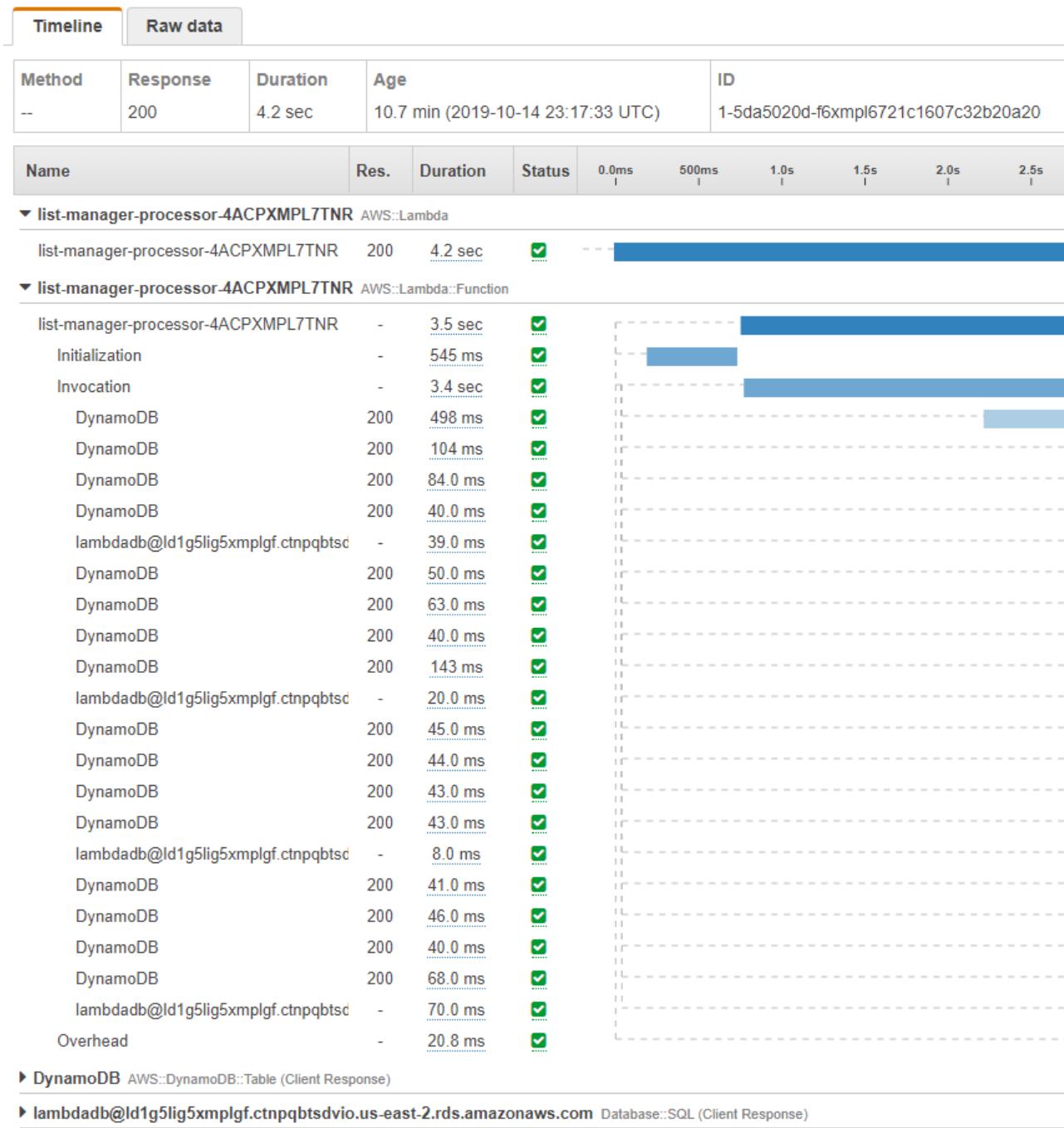
Instrumentação com o AWS X-Ray

O aplicativo usa [AWS X-Ray \(p. 477\)](#) para rastrear chamadas de função e as chamadas que as funções fazem para [AWS Serviços da .](#) O X-Ray usa os dados de rastreamento que recebe das funções para criar um mapa de serviço que ajuda a identificar erros. O mapa de serviço a seguir mostra a função que se comunica com duas tabelas do DynamoDB e um banco de dados MySQL.



A função do Node.js é configurada para o rastreamento ativo no modelo e é instrumentadas com o SDK do AWS X-Ray for Node.js no código. O X-Ray SDK registra um subsegmento para cada chamada feita com um AWS SDK ou um cliente MySQL.

Traces > Details



A função usa o AWS SDK for JavaScript in Node.js para ler e gravar nas duas tabelas para cada registro. A tabela principal armazena o estado atual de cada combinação de usuário e nome da lista. A tabela agregada armazena listas que combinam dados de vários usuários.

AWS CloudFormation Modelos de recursos adicionais

O aplicativo é implementado nos módulos do Node.js e implantado com um modelo do AWS CloudFormation e scripts de shell. O modelo de aplicação cria duas funções, uma transmissão do Kinesis, tabelas do DynamoDB e os recursos de suporte a seguir.

Recursos do aplicativo

- Função de execução: uma função do IAM que concede às funções permissão para acessar outros serviços da AWS.
- Mapeamento de origem do evento do Lambda: lê registros do fluxo de dados e invoca a função.

Visualize o [modelo de aplicação](#) no GitHub.

Um segundo modelo, [template-vpcrds.yml](#), cria a Amazon VPC e os recursos do banco de dados. Embora seja possível criar todos os recursos em um modelo, separá-los facilita a limpeza do aplicativo e permite que o banco de dados seja reutilizado com vários aplicativos.

Recursos de infraestrutura

- VPC: uma rede de nuvem privada virtual com sub-redes privadas, uma tabela de rotas e um endpoint da VPC que permite que a função se comunique com o DynamoDB sem uma conexão com a Internet.
- Banco de dados: uma instância de banco de dados do Amazon RDS e um grupo de sub-rede que a conecta à VPC.

Criar funções do Lambda com Node.js

Você pode executar o código JavaScript com Node.js no AWS Lambda. Lambda fornece [Tempos de execução do \(p. 214\)](#) para executar o seu código para processar eventos. Node.js Seu código é executado em um ambiente que inclui o AWS SDK for JavaScript, com as credenciais de uma função do AWS Identity and Access Management (IAM) que você gerencia.

O Lambda oferece suporte aos seguintes tempos de execução Node.js.

Tempos de execução Node.js

Nome	Identifier	SDK for JavaScript	Sistema operacional
Node.js 14	nodejs14.x	2.952.0	Amazon Linux 2
Node.js 12	nodejs12.x	2.952.0	Amazon Linux 2
Node.js 10	nodejs10.x	2.952.0	Amazon Linux 2

Note

Para obter informações sobre o fim do suporte a respeito do Node.js 10, consulte [the section called “Política de suporte ao tempo de execução” \(p. 217\)](#).

Funções do Lambda usam um [Função de execução do \(p. 57\)](#) para obter permissão para gravar logs no Amazon CloudWatch Logs e para acessar outros serviços e recursos. Se você ainda não tiver uma função de execução para o desenvolvimento de funções, crie uma.

Para criar uma função de execução

1. Abra a [página Roles](#) (Funções) no console do IAM.
2. Selecione Create role.
3. Crie uma função com as seguintes propriedades.
 - Entidade confiável—Lambda.
 - Permissions (Permissões): AWSLambdaBasicExecutionRole.
 - Nome da função – **lambda-role**.

A política AWSLambdaBasicExecutionRole tem as permissões necessárias para a função gravar logs no CloudWatch Logs.

Você pode adicionar permissões à função posteriormente ou trocá-la para uma função diferente específica de uma única função.

Para criar uma função do Node.js

1. Abra o [console do lambda](#).
2. Escolha Create function.

3. Configure as definições a seguir:
 - Nome – **my-function**.
 - Runtime (Tempo de execução): Node.js 14.x.
 - Role (Função): Choose an existing role (Escolher uma função existente).
 - Existing role (Função existente – **lambda-role**).
4. Escolha Create function.
5. Para configurar um evento de teste, escolha Test (Testar).
6. Em Event name (Nome do evento), insira **test**.
7. Selecione Save changes.
8. Escolha Test (Testar) para invocar a função.

O console cria uma função do Lambda com um único arquivo de origem chamado `index.js`. Você pode editar esse arquivo e adicionar mais arquivos no [editor de códigos \(p. 40\)](#) integrado. Para salvar suas alterações, selecione Save (Salvar). Em seguida, para executar seu código, escolha Teste.

Note

O console do Lambda usa o AWS Cloud9 para fornecer um ambiente de desenvolvimento integrado no navegador. Você também pode usar o AWS Cloud9 para desenvolver funções do Lambda em seu próprio ambiente. Para obter mais informações, consulte [Working with Lambda Functions](#) no guia do usuário do AWS Cloud9.

O arquivo `index.js` exporta uma função chamada `handler` que utiliza um objeto de evento e um objeto de contexto. Esta é a [função de manipulador \(p. 514\)](#) que o Lambda chama quando a função é invocada. O tempo de execução da função do Node.js recebe eventos de invocação do Lambda e os transmite ao manipulador. Na configuração da função, o valor do `handler` é `index.handler`.

Quando você salva seu código de função, o console do Lambda cria um pacote de implantação de arquivos.zip. Ao desenvolver seu código de função fora do console (usando um SDE), você precisa [Criar um pacote de implantação \(p. 517\)](#) para carregar seu código para a função do Lambda.

Note

Para começar com o desenvolvimento de aplicativos no ambiente local, implante um dos aplicativos de exemplo disponíveis no repositório do GitHub deste guia.

Aplicações de exemplo do Lambda em Node.js

- [blank-nodejs](#): uma função do Node.js que mostra o uso do registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK.
- [nodejs-apig](#): uma função com endpoint de API pública que processa um evento do API Gateway e retorna uma resposta HTTP.
- [rds-mysql](#): uma função que retransmite consultas para um banco de dados MySQL para RDS. Este exemplo inclui uma VPC privada e uma instância de banco de dados configurada com uma senha no AWS Secrets Manager.
- [efs-nodejs](#): uma função que usa um sistema de arquivos do Amazon EFS em uma Amazon VPC. Esse exemplo inclui uma VPC, um sistema de arquivos, destinos de montagem e ponto de acesso configurado para uso com o Lambda.
- [list-manager](#): uma função processa eventos de um fluxo de dados do Amazon Kinesis e atualiza listas agregadas no Amazon DynamoDB. A função armazena um registro de cada evento em um banco de dados MySQL para RDS em uma VPC privada. Este exemplo inclui uma VPC privada com um endpoint da VPC para o DynamoDB e uma instância de banco de dados.
- [error-processor](#): uma função do Node.js gera erros para uma porcentagem especificada de solicitações. Uma assinatura do CloudWatch Logs invoca uma segunda função quando um

erro é registrado. A função do processador usa o AWS SDK para coletar detalhes sobre a solicitação e os armazena em um bucket do Amazon S3.

O tempo de execução da função transmite um objeto de contexto para o handler, além do evento de invocação. O [objeto de contexto \(p. 522\)](#) contém informações adicionais sobre a invocação, a função e o ambiente de execução. Outras informações estão disponíveis de variáveis de ambiente.

Sua função do Lambda é fornecida com um grupo de logs do CloudWatch Logs. O tempo de execução da função envia detalhes sobre cada invocação para o CloudWatch Logs. Ele retransmite qualquer [logs que sua função produz \(p. 524\)](#) durante a invocação. Se a função [retornar um erro \(p. 529\)](#), o Lambda formatará o erro e o retornará para o chamador.

Tópicos

- [AWS Lambda Manipulador da função do em Node.js \(p. 514\)](#)
- [Implantar funções do Lambda em Node.js com arquivos .zip \(p. 517\)](#)
- [Implantar funções do Lambda em Node.js com imagens de contêiner \(p. 520\)](#)
- [AWS Lambda Objeto de contexto do em Node.js \(p. 522\)](#)
- [AWS Lambda Registro em log da função do em Node.js \(p. 524\)](#)
- [AWS Lambda Erros da função do em Node.js \(p. 529\)](#)
- [Instrumentação do código Node.js no AWS Lambda \(p. 533\)](#)

AWS LambdaManipulador da função do em Node.js

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. Quando o manipulador é encerrado ou retorna uma resposta, ele se torna disponível para tratar de outro evento.

O exemplo de função a seguir registra o conteúdo do objeto de evento e retorna o local dos logs.

Example index.js

```
exports.handler = async function(event, context) {
  console.log("EVENT: \n" + JSON.stringify(event, null, 2))
  return context.logStreamName
}
```

Quando você configura uma função, o valor da configuração do handler é o nome do arquivo e o nome do módulo do handler exportado, separados por um ponto. O padrão no console e nos exemplos deste guia é `index.handler`. Isso indica o método do handler que é exportado do arquivo `index.js`.

O tempo de execução transmite três argumentos para o método do manipulador. O primeiro argumento é o objeto `event`, que contém informações do chamador. O invocador passa essas informações como uma string no formato JSON ao chamar [Invoke \(p. 875\)](#), e o tempo de execução as converte em um objeto. Quando um serviço da AWS invoca a função, a estrutura do evento [varia de acordo com o serviço \(p. 277\)](#).

O segundo argumento é o [objeto de contexto \(p. 522\)](#), que contém informações sobre a invocação, a função e o ambiente de execução. No exemplo anterior, a função obtém o nome do [fluxo de log \(p. 524\)](#) do objeto de contexto e o retorna para o chamador.

O terceiro argumento, `callback`, é uma função que você pode chamar nos [manipuladores não assíncronos \(p. 515\)](#) para enviar uma resposta. A função de retorno de chamada usa dois argumentos: um `Error` e uma resposta. Quando você chamá-lo, o Lambda aguarda que o loop de eventos esteja vazio e, em seguida, retorna a resposta ou erro para o invocador. O objeto de resposta deve ser compatível com `JSON.stringify`.

Para manipuladores assíncronos, você retorna uma resposta, um erro ou uma promessa para o tempo de execução em vez de usar o `callback`.

Se sua função tiver dependências adicionais, [use o npm para incluí-las em seu pacote de implantação \(p. 518\)](#).

Manipuladores assíncronos

Para manipuladores assíncronos, você pode usar `return` e `throw` para enviar uma resposta ou um erro, respectivamente. As funções devem usar a palavra-chave `async` para usar esses métodos a fim de retornar uma resposta ou um erro.

Se o código executar uma tarefa assíncrona, retorne uma promessa para garantir que a execução seja finalizada. Quando você resolve ou rejeita a promessa, o Lambda envia a resposta ou o erro para o chamador.

Example Arquivo index.js: solicitação de HTTP com manipulador e promessas assíncronos

```
const https = require('https')
let url = "https://docs.aws.amazon.com/lambda/latest/dg/welcome.html"

exports.handler = async function(event) {
  const promise = new Promise(function(resolve, reject) {
```

```
    https.get(url, (res) => {
      resolve(res.statusCode)
    }).on('error', (e) => {
      reject(Error(e))
    })
  }
  return promise
}
```

Para bibliotecas que retornam uma promessa, você pode retornar essa promessa diretamente para o tempo de execução.

Example Arquivo index.js: AWS SDK com manipulador e promessas assíncronos

```
const AWS = require('aws-sdk')
const s3 = new AWS.S3()

exports.handler = async function(event) {
  return s3.listBuckets().promise()
}
```

Manipuladores não assíncronos

O exemplo de função a seguir verifica uma URL e retorna o código de status para o chamador.

Example Arquivo index.js: solicitação HTTP com chamada de retorno

```
const https = require('https')
let url = "https://docs.aws.amazon.com/lambda/latest/dg/welcome.html"

exports.handler = function(event, context, callback) {
  https.get(url, (res) => {
    callback(null, res.statusCode)
  }).on('error', (e) => {
    callback(Error(e))
  })
}
```

No caso de manipuladores não assíncronos, a execução da função continua até que o [loop de evento](#) esteja vazio ou o tempo da função se esgotar. A resposta não é enviada para o chamador até que todas as tarefas de loop de evento estejam concluídas. Se a função expirar, um erro será retornado. É possível configurar o tempo de execução para enviar a resposta imediatamente, definindo `context.callbackWaitsForEmptyEventLoop` (p. 522) como false.

No exemplo a seguir, a resposta do Amazon S3 é retornada ao chamador assim que fica disponível. O tempo limite em execução no loop de evento é congelado e continuará sendo executado na próxima vez que a função for invocada.

Example Arquivo index.js: callbackWaitsForEmptyEventLoop

```
const AWS = require('aws-sdk')
const s3 = new AWS.S3()

exports.handler = function(event, context, callback) {
  context.callbackWaitsForEmptyEventLoop = false
  s3.listBuckets(null, callback)
  setTimeout(function () {
    console.log('Timeout complete.')
  }, 5000)
```

}

Implantar funções do Lambda em Node.js com arquivos .zip

O código da função do AWS Lambda consiste em scripts ou programas compilados e as dependências deles. Você usa um pacote de implantação para implantar seu código de função no Lambda. O Lambda é compatível com dois tipos de pacotes de implantação: imagens de contêiner e arquivos .zip.

Para criar um pacote de implantação compactado em um arquivo .zip, você pode usar um utilitário de compactação em arquivo .zip integrado ou qualquer outro utilitário semelhante (como o [7zip](#)) para sua ferramenta de linha de comando. Lembre-se dos seguintes requisitos para usar um arquivo .zip como seu pacote de implantação:

- O arquivo .zip contém o código da sua função e quaisquer dependências usadas para executá-lo (se aplicável) no Lambda. Se sua função depender apenas de bibliotecas padrão ou de bibliotecas do SDK do AWS, você não precisará incluí-las em seu arquivo .zip. Essas bibliotecas são incluídas nos ambientes de [tempo de execução do Lambda](#) (p. 214) suportados.
- Se o arquivo.zip for maior que 50 MB, recomendamos enviá-lo para sua função a partir de um bucket do Amazon Simple Storage Service (Amazon S3).
- Se o pacote de implantação contiver bibliotecas nativas, você poderá criar o pacote de implantação com o AWS Serverless Application Model (AWS SAM). É possível usar o comando AWS SAM da CLI do `sam build` com `--use-container` para criar seu pacote de implantação. Essa opção cria um pacote de implantação dentro de uma imagem do Docker compatível com o ambiente de execução do Lambda.

Para obter mais informações, consulte [sam build](#) no Guia do desenvolvedor do AWS Serverless Application Model.

- O Lambda usa permissões de arquivo POSIX, então pode ser necessário [definir permissões para a pasta do pacote de implantação](#) antes de criar o arquivo .zip.

Seções

- [Prerequisites \(p. 517\)](#)
- [Atualizar uma função sem dependências \(p. 517\)](#)
- [Atualizar uma função com dependências adicionais \(p. 518\)](#)

Prerequisites

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Atualizar uma função sem dependências

Para atualizar uma função usando a API do Lambda, use a função [UpdateFunctionCode \(p. 965\)](#) operação. Crie um arquivo que contenha o código de função e faça upload dele usando a AWS Command Line Interface (AWS CLI).

Para atualizar uma função Node.js sem dependências

1. Crie um arquivo .zip.

```
zip function.zip index.js
```

2. Use o comando `update-function-code` para fazer upload do pacote.

```
aws lambda update-function-code --function-name my-function --zip-file fileb://function.zip
```

Você deve ver a saída a seguir:

```
{  
    "FunctionName": "my-function",  
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",  
    "Runtime": "nodejs12.x",  
    "Role": "arn:aws:iam::123456789012:role/lambda-role",  
    "Handler": "index.handler",  
    "CodeSha256": "Qf0hMc1I2di6YFMi9aXm3JtGTmcDbjniEuiYonYptAk=",  
    "Version": "$LATEST",  
    "TracingConfig": {  
        "Mode": "Active"  
    },  
    "RevisionId": "983ed1e3-ca8e-434b-8dc1-7d72ebadd83d",  
    ...  
}
```

Atualizar uma função com dependências adicionais

Se a função depende de outras bibliotecas que não o AWS SDK for JavaScript, use o [npm](#) para incluí-las em seu pacote de implantação. Verifique se a versão do Node.js no ambiente local corresponde à versão do Node.js da função. Se alguma das bibliotecas usar código nativo, [use um ambiente do Amazon Linux](#) para criar o pacote de implantação.

Você poderá adicionar o SDK for JavaScript ao pacote de implantação se precisar de uma versão mais recente do que a [incluída no tempo de execução \(p. 511\)](#), ou para garantir que a versão não seja alterada no futuro.

Se o pacote de implantação contiver bibliotecas nativas, você poderá criar o pacote de implantação com o AWS Serverless Application Model (AWS SAM). É possível usar o comando AWS SAM da CLI do `sam build` com `--use-container` para criar seu pacote de implantação. Essa opção cria um pacote de implantação dentro de uma imagem do Docker compatível com o ambiente de execução do Lambda.

Para obter mais informações, consulte [sam build](#) no Guia do desenvolvedor do AWS Serverless Application Model.

Como alternativa, você pode criar o pacote de implantação usando uma instância do Amazon EC2 que fornece um ambiente do Amazon Linux. Para obter instruções, consulte [Como usar pacotes e módulos nodejs nativos na AWS](#) no blog de computação da AWS.

Para atualizar uma função Node.js com dependências

1. Abra um shell ou um terminal da linha de comando. Verifique se a versão do Node.js no ambiente local corresponde à versão do Node.js da função.
2. Crie uma pasta para o pacote de implantação. As etapas a seguir assumem que a pasta é nomeada como `my-function`.
3. Instale bibliotecas no diretório `node_modules` com o comando `npm install`.

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

Isso cria uma estrutura de pastas que é semelhante à seguinte:

```
~/my-function
### index.js
### node_modules
### async
### async-listener
### atomic-batcher
### aws-sdk
### aws-xray-sdk
### aws-xray-sdk-core
```

- Crie um arquivo .zip com o conteúdo da pasta do projeto. Use a opção r (recursiva) para garantir que zip compacte as subpastas.

```
zip -r function.zip .
```

- Faça upload do pacote usando o comando update-function-code.

```
aws lambda update-function-code --function-name my-function --zip-file fileb://
function.zip
```

Você deve ver a saída a seguir:

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",
  "Runtime": "nodejs12.x",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Handler": "index.handler",
  "CodeSha256": "Qf0hMc1I2di6YFMi9aXm3JtGTmcDbjniEuiYonYptAk=",
  "Version": "$LATEST",
  "TracingConfig": {
    "Mode": "Active"
  },
  "RevisionId": "983ed1e3-ca8e-434b-8dc1-7d72ebadd83d",
  ...
}
```

Além do código e bibliotecas, o pacote de implantação também pode conter arquivos executáveis e outros recursos. Para obter mais informações, consulte [Como executar executáveis arbitrários no AWS Lambda](#) no blog de computação da AWS.

Implantar funções do Lambda em Node.js com imagens de contêiner

Você pode implantar seu código de função do Lambda como uma [imagem de contêiner \(p. 267\)](#). A AWS oferece os seguintes recursos para ajudar você a criar uma imagem de contêiner para sua função Node.js:

- [AWSImagens base para o Lambda](#)

Essas imagens base são pré-carregadas com um tempo de execução de linguagem e outros componentes necessários para executar a imagem no Lambda. AWS fornece um Dockerfile para cada uma das imagens de base para ajudar a criar sua imagem de contêiner.

- [Clientes de interface de tempo de execução de](#)

Se você usar uma imagem de base de comunidade ou de empresa privada, adicione um cliente de interface de tempo de execução à imagem base para torná-lo compatível com o Lambda.

O fluxo de trabalho de uma função definida como uma imagem de contêiner inclui as seguintes etapas:

1. Crie sua imagem de contêiner usando os recursos listados neste tópico.
2. Faça upload da imagem em seu registro do contêiner do Amazon ECR. Consulte os passos 7 a 9 no [Criar imagem \(p. 268\)](#).
3. [Crie \(p. 91\)](#) a função do Lambda ou [atualize o código da função \(p. 92\)](#) para implantar a imagem em uma função existente.

Tópicos

- [AWSImagens de base da para Node.js \(p. 520\)](#)
- [Usar uma imagem base Node.js \(p. 520\)](#)
- [Clientes de interface de tempo de execução do Node.js \(p. 521\)](#)
- [Implantar a imagem do contêiner \(p. 521\)](#)

AWSImagens de base da para Node.js

AWSA oferece as seguintes imagens de base para Node.js:

Tags	Tempo de execução	Sistema operacional	Dockerfile
14	NodeJS 14.x	Amazon Linux 2	Dockerfile para Node.js 14.x no GitHub
12	NodeJS 12.x	Amazon Linux 2	Dockerfile para Node.js 12.x no GitHub
10	NodeJS 10.x	Amazon Linux 2	Dockerfile para Node.js 10.x no GitHub

Repositório do Docker Hub: [amazon/aws-lambda-nodejs](#)

Repositório do Amazon ECR: [gallery.ecr.aws/lambda/nodejs](#)

Usar uma imagem base Node.js

Para obter instruções sobre como usar uma imagem base Node.js, escolha a guia `usage` (uso) em [Imagens base do AWS Lambda para Node.js](#) no repositório do Amazon ECR.

As instruções também estão disponíveis em [Imagens base do AWS Lambda para Node.js](#) no repositório do Docker Hub.

Cientes de interface de tempo de execução do Node.js

Instale o cliente de interface de tempo de execução para Node.js usando o gerenciador de pacotes npm:

```
npm install aws-lambda-ric
```

Para obter detalhes do pacote, consulte [Lambda RIC](#) no site do npm.

Você também pode fazer download do [cliente de interface de tempo de execução Node.js](#) no GitHub.

Implantar a imagem do contêiner

Para uma nova função, você implanta a imagem Node.js ao [criar a função \(p. 91\)](#). Para uma função existente, se você reconstruir a imagem do contêiner, precisará reimplantar a imagem ao [atualizar o código da função \(p. 92\)](#).

AWS LambdaObjeto de contexto do em Node.js

Quando o Lambda executa a função, ele transmite um objeto de contexto para o [handler \(p. 514\)](#). Esse objeto fornece métodos e propriedades que fornecem informações sobre a invocação, a função e o ambiente de execução.

Métodos de contexto

- `getRemainingTimeInMillis()`: retorna o número de milissegundos restantes antes do tempo limite da execução.

Propriedades de contexto

- `functionName`: o nome da função do Lambda.
- `functionVersion`: a [versão \(p. 106\)](#) da função.
- `invokedFunctionArn`: o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um alias ou número de versão.
- `memoryLimitInMB`: a quantidade de memória alocada para a função.
- `awsRequestId`: o identificador da solicitação de invocação.
- `logGroupName`: o grupo de logs da função.
- `logStreamName`: a transmissão de log para a instância da função.
- `identity`: (aplicativos móveis) informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
 - `cognitoIdentityId`: a identidade autenticada do Amazon Cognito.
 - `cognitoIdentityPoolId`: o grupo de identidades do Amazon Cognito que autorizou a invocação.
- `clientContext`: (aplicativos móveis) contexto do cliente fornecido ao Lambda pela aplicação cliente.
 - `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`: os valores personalizados que são definidos pela aplicação cliente.
- `callbackWaitsForEmptyEventLoop`: definido como falso para enviar a resposta imediatamente quando o [retorno de chamada \(p. 515\)](#) é executado, em vez de aguardar até que o loop de eventos do Node.js esteja vazio. Se for falso, todos os eventos pendentes continuarão a ser executados durante a próxima invocação.

O exemplo a seguir registra informações de contexto da função e retorna a localização dos logs.

Example Arquivo index.js

```
exports.handler = async function(event, context) {
  console.log('Remaining time: ', context.getRemainingTimeInMillis())
  console.log('Function name: ', context.functionName)
  return context.logStreamName
```

}

AWS Lambda Registro em log da função do em Node.js

O AWS Lambda monitora automaticamente as funções do Lambda em seu nome e envia métricas da função para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente do tempo de execução do Lambda envia detalhes sobre cada invocação à transmissão de logs e transmite os logs e outras saídas do código de sua função.

Esta página descreve como produzir a saída de logs usando o código de sua função do Lambda ou acessar os logs usando a AWS Command Line Interface, o console do Lambda ou o console do CloudWatch.

Seções

- [Criar uma função que retorna logs \(p. 524\)](#)
- [Usar o console do Lambda \(p. 525\)](#)
- [Usando o console do CloudWatch \(p. 525\)](#)
- [Usar a AWS Command Line Interface \(AWS CLI\) \(p. 526\)](#)
- [Excluir logs \(p. 528\)](#)

Criar uma função que retorna logs

Para gerar os logs do código de função, você pode usar métodos no [objeto do console](#) ou qualquer biblioteca de logs que grave no `stdout` ou `stderr`. O exemplo a seguir registra em log os valores das variáveis de ambiente e o objeto do evento.

Example Arquivo index.js: registro em 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 do 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:
```

```
200 ms Memory Size: 128 MB Max Memory Used: 74 MB Init Duration: 166.62 ms XRAY TraceId:
```

```
1-5d9d007f-0a8c7fd02xmp1480aed55ef0 SegmentId: 3d752xmplbbbe37e Sampled: true
```

O tempo de execução do Node.js registra as linhas START, END e REPORT para cada invocação. Ele adiciona um carimbo de data e hora, o ID da solicitação e o nível de log em cada entrada registrada pela função. A linha do relatório fornece os seguintes detalhes.

Registro em log de relatório

- RequestId: o ID de solicitação exclusivo para a invocação.
- Duração: a quantidade de tempo que o método de manipulador da função gastou processando o evento.
- Duração faturada: a quantia de tempo faturada para a invocação.
- Tamanho da memória: a quantidade de memória alocada para a função.
- Memória máxima utilizada: a quantidade de memória utilizada pela função.
- Duração inicial: para a primeira solicitação atendida, a quantidade de tempo que o tempo de execução levou para carregar a função e executar o código fora do método do manipulador.
- XRAY Traceld: para solicitações rastreadas, o [ID de rastreamento do AWS X-Ray \(p. 477\)](#).
- SegmentId: para solicitações rastreadas, o ID do segmento do X-Ray.
- Amostragem: para solicitações rastreadas, o resultado da amostragem.

Você pode visualizar logs no console do Lambda, no console do CloudWatch Logs ou na linha de comando.

Usar o console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda. Para obter mais informações, consulte [Acessar o Amazon CloudWatch Logs para o AWS Lambda \(p. 720\)](#).

Usando o console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a [página Log groups](#) (Grupos de log).
2. Escolha o grupo de logs de sua função (`/aws/lambda/nome-de-sua-função`).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma [instância da sua função \(p. 219\)](#). Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos instrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

Para usar uma aplicação de exemplo que correlaciona os logs e os rastreamentos com o X-Ray, consulte [Aplicativo de exemplo de processador de erros para o AWS Lambda \(p. 500\)](#).

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Você pode usar a [AWS CLI](#) para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Você deve ver a saída a seguir:

```
{  
    "StatusCode": 200,  
    "LogResult":  
        "U1RBULQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",  
    "ExecutedVersion": "$LATEST"  
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \  
--query 'LogResult' --output text | base64 -d
```

Você deve ver a saída a seguir:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e [Ubuntu no Windows](#). Os usuários do macOS precisam usar `base64 -D`.

Example get-logs.sh script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa `sed` para remover as aspas do arquivo de saída e fica inativo por 15 segundos para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como `get-logs.sh`.

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

```
#!/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 $(cat
out) --limit 5
```

Example macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod +x get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

Você deve ver a saída a seguir:

```
{
  "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"
}
```

Excluir logs

Os grupos de logs não são excluídos automaticamente quando você exclui uma função. Para evitar armazenar logs indefinidamente, exclua o grupo de logs ou[Configurar um período de retenção](#)após o qual os logs são excluídos automaticamente.

AWS Lambda Erros da função do em Node.js

Quando o código gera um erro, o Lambda gera uma representação JSON do erro. Esse documento de erro aparece no log de invocação e, para invocações síncronas, na saída.

Esta página descreve como exibir erros de invocação de função do Lambda para o tempo de execução do Node.js usando o console do Lambda e a AWS CLI.

Seções

- [Syntax \(p. 529\)](#)
- [Como funcionam \(p. 529\)](#)
- [Usar o console do Lambda \(p. 530\)](#)
- [Usar a AWS Command Line Interface \(AWS CLI\) \(p. 530\)](#)
- [Tratamento de erros em outros serviços da AWS \(p. 531\)](#)
- [Próximas etapas \(p. 532\)](#)

Syntax

Example Arquivo index.js: erro de referência

```
exports.handler = async function() {
    return x + 10
}
```

Esse código resulta em um erro de referência. O Lambda detecta o erro e gera um documento JSON com os campos para a mensagem de erro, o tipo e o rastreamento da pilha.

```
{
  "errorType": "ReferenceError",
  "errorMessage": "x is not defined",
  "trace": [
    "ReferenceError: x is not defined",
    "  at Runtime.exports.handler (/var/task/index.js:2:3)",
    "  at Runtime.handleOnce (/var/runtime/Runtime.js:63:25)",
    "    at process._tickCallback (internal/process/next_tick.js:68:7)"
  ]
}
```

Como funcionam

Ao invocar uma função do Lambda, o Lambda recebe a solicitação de invocação e valida as permissões de sua função de execução, verifica se o documento do evento é um documento JSON válido e verifica valores de parâmetros.

Se a solicitação for validada, o Lambda a envia para uma instância da função. O ambiente de [tempo de execução do Lambda \(p. 214\)](#) converte o documento do evento em um objeto e o transmite ao handler da função.

Se o Lambda encontra um erro, ele retorna um tipo de exceção, uma mensagem e o código HTTP do status que indica a causa do erro. O cliente ou o serviço que invocou a função do Lambda pode processar o erro de maneira programática ou transmiti-lo a um usuário final. O comportamento correto de tratamento de erros depende do tipo de aplicativo, do público e da origem do erro.

A lista a seguir descreve o intervalo de códigos de status que você pode receber do Lambda.

2xx

Um erro da série 2xx com um cabeçalho `X-Amz-Function-Error` na resposta indica um erro de tempo de execução ou de função do Lambda. Um código de status da série 2xx indica que o Lambda aceitou a solicitação, mas em vez de um código de erro, o Lambda indica o erro incluindo o cabeçalho `X-Amz-Function-Error` na resposta.

4xx

Um erro da série 4xx indica um erro que o cliente ou serviço que fez a invocação pode corrigir modificando a solicitação, solicitando permissão ou tentando a solicitação novamente. Os erros da série 4xx diferentes de 429 geralmente indicam um erro na solicitação.

5xx

Um erro da série 5xx indica um problema com o Lambda ou com a configuração/recursos da função. Os erros da série 5xx podem indicar uma condição temporária que pode ser resolvida sem nenhuma ação do usuário. O cliente ou serviço que fez a invocação não podem solucionar esses problemas, mas o proprietário de uma função do Lambda pode ser capaz de corrigi-los.

Para obter uma lista completa de erros de invocação, consulte [Erros de InvokeFunction \(p. 877\)](#).

Usar o console do Lambda

Você pode invocar sua função no console do Lambda configurando um evento de teste e visualizando a saída. A saída é capturada pelos logs de execução da função e, quando o [rastreamento ativo \(p. 477\)](#) está habilitado, pelo AWS X-Ray.

Para invocar uma função no console do Lambda

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Test (Testar).
4. Selecione New event (Novo evento) e escolha um Event template (Modelo de evento) na lista suspensa.
5. Insira um nome para o evento de teste.
6. Digite o JSON para o evento de teste.
7. Escolha Create event (Criar evento).
8. Escolha Invoke (Invocar).

O console do Lambda invoca sua função [de forma síncrona \(p. 158\)](#) e exibe o resultado. Para ver a resposta, os logs e outras informações, expanda a seção Details (Detalhes).

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Quando você invoca uma função do Lambda na AWS CLI, a AWS CLI divide a resposta em dois documentos. A resposta da AWS CLI é exibida no prompt de comando. Se ocorreu um erro, a resposta

contém um campo `FunctionError`. A resposta ou o erro de invocação retornado pela função é gravado no arquivo de saída. Por exemplo, o `output.json` ou o `output.txt`.

O exemplo de comando `invoke` a seguir demonstra como invocar uma função e escrever a resposta de invocação em um arquivo `output.txt`.

```
aws lambda invoke \
--function-name my-function \
--cli-binary-format raw-in-base64-out \
--payload '{"key1": "value1", "key2": "value2", "key3": "value3"}' output.txt
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

Você deve ver a resposta da AWS CLI em seu prompt de comando:

```
{
  "StatusCode": 200,
  "FunctionError": "Unhandled",
  "ExecutedVersion": "$LATEST"
}
```

Você deve ver a resposta de invocação de função no arquivo `output.txt`. Também é possível visualizar a saída no mesmo prompt de comando usando:

```
cat output.txt
```

Você deve ver a resposta de invocação no prompt de comando.

```
{"errorType": "ReferenceError", "errorMessage": "x is not defined", "trace": ["ReferenceError: x is not defined", " at Runtime.exports.handler (/var/task/index.js:2:3)", " at Runtime.handleOnce (/var/runtime/Runtime.js:63:25)", " at process._tickCallback (internal/process/next_tick.js:68:7)"]}
```

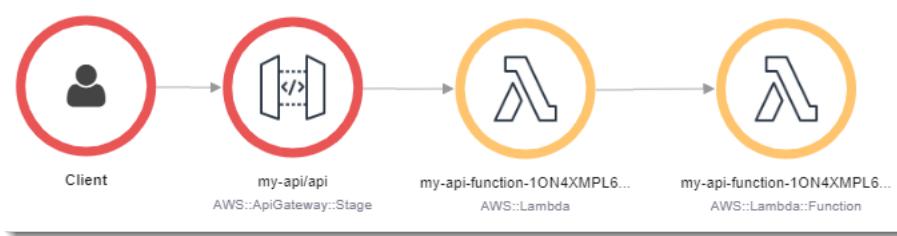
O Lambda também grava até 256 KB do objeto de erro nos logs de função. Para obter mais informações, consulte [AWS Lambda Registro em log da função do em Node.js \(p. 524\)](#).

Tratamento de erros em outros serviços da AWS

Quando outro serviço da AWS invoca sua função, o serviço escolhe o tipo de invocação e o comportamento de repetição. Os serviços da AWS podem invocar sua função em um agendamento, em resposta a um evento de ciclo de vida em um recurso ou para atender a uma solicitação de um usuário. Alguns serviços invocam funções de forma assíncrona e permitem que o Lambda trate erros, enquanto outros fazem novas tentativas ou transmitem os erros de volta ao usuário.

Por exemplo, o API Gateway trata todos os erros de invocação e função como erros internos. Se a API do Lambda rejeitar a solicitação de invocação, o API Gateway retornará um código de erro 500. Se a função é executada, mas retorna um erro ou retorna uma resposta no formato errado, o API Gateway retorna um código de erro 502. Para personalizar a resposta de erro, é necessário detectar erros no código e formatar uma resposta no formato necessário.

Recomendamos usar AWS X-Ray para determinar a fonte de um erro e a respectiva causa. O X-Ray permite localizar qual componente encontrou um erro e ver detalhes sobre os erros. O exemplo a seguir mostra um erro de função que resultou em uma resposta 502 do API Gateway.



Para obter mais informações, consulte [Instrumentação do código Node.js no AWS Lambda \(p. 533\)](#).

Próximas etapas

- Saiba como mostrar eventos de registro em log para sua função do Lambda na página [the section called “Registro em log” \(p. 524\)](#).

Instrumentação do código Node.js no AWS Lambda

O Lambda se integra ao AWS X-Ray para permitir que você rastreie, depure e otimize aplicativos do Lambda. É possível usar o X-Ray para rastrear uma solicitação à medida que ela atravessa recursos na aplicação, da API de frontend ao armazenamento e aos banco de dados no backend. Ao simplesmente adicionar a biblioteca do X-Ray SDK à configuração de compilação, é possível registrar erros e latência para qualquer chamada que a função faça para um serviço da AWS.

O X-Ray Mapa de serviço mostra o fluxo de solicitações através de seu aplicativo. O exemplo a seguir do aplicativo de exemplo [processador de erros \(p. 500\)](#) mostra um aplicativo com duas funções. A função principal processa eventos e, às vezes, retorna erros. A segunda função processa erros que aparecem no primeiro grupo de logs e usa o AWS SDK para chamar o X-Ray, o Amazon S3 e o Amazon CloudWatch Logs.



Para rastrear solicitações que não têm um cabeçalho de rastreamento, ative o rastreamento ativo na configuração da sua função.

Para ativar o rastreamento ativo

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Selecione **Configuração**, depois, escolha **Ferramentas de monitoramento**.
4. Selecione **Edit**.
5. Em X-Ray, habilite o **Active tracing (Rastreamento ativo)**.
6. Escolha **Save (Salvar)**.

Pricing

O X-Ray tem um nível gratuito vitalício. Além do limite do nível gratuito, o X-Ray cobra por armazenamento e recuperação de rastreamento. Para obter detalhes, consulte [Definição de preço do AWS X-Ray](#).

Sua função precisa de permissão para carregar dados de rastreamento no X-Ray. Quando você habilita o rastreamento ativo no console do Lambda, o Lambda adiciona as permissões necessárias à [função de](#)

[execução \(p. 57\)](#) da função. Caso contrário, adicione a política [AWSXRayDaemonWriteAccess](#) à função de execução.

O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento é eficiente, enquanto ainda fornece uma amostra representativa das solicitações que a sua aplicação serviu. A regra de amostragem padrão é uma solicitação por segundo e 5% de solicitações adicionais. Esta taxa de amostragem não pode ser configurada para funções do Lambda.

Quando o rastreamento ativo está habilitado, o Lambda registra um rastreamento para um subconjunto de invocações. Lambda registra doisSegmentos do, que cria dois nós no mapa de serviço. O primeiro nó representa o serviço Lambda que recebe a solicitação de invocação. O segundo nó é gravado pelo [tempo de execução \(p. 19\)](#)da função.



Configuração

O tempo de execução do Lambda define algumas variáveis de ambiente para configurar o X-Ray SDK, incluindo `AWS_XRAY_CONTEXT_MISSING`. Para definir uma estratégia de contexto ausente personalizada, substitua a variável de ambiente na configuração da função para que ela não tenha valores e, depois, defina a estratégia de contexto ausente de forma programática. Para obter mais informações, consulte [Variáveis de ambiente de tempo de execução \(p. 102\)](#).

Example Exemplo de código de inicialização

```
const AWSXRay = require('aws-xray-sdk-core');

// Configure the context missing strategy to do nothing
AWSXRay.setContextMissingStrategy(() => {});
```

É possível instrumentar o código do manipulador para gravar metadados e rastrear chamadas downstream. Para registrar detalhes sobre chamadas feitas pelo manipulador para outros recursos e serviços, use o X-Ray SDK for Node.js. Para obter o SDK, adicione o pacote `aws-xray-sdk-core` às dependências do aplicativo.

Example [blank-nodejs/package.json](#)

```
{
  "name": "blank-nodejs",
  "version": "1.0.0",
  "private": true,
  "devDependencies": {
    "aws-sdk": "2.631.0",
    "jest": "25.4.0"
  },
  "dependencies": {
    "aws-xray-sdk-core": "1.1.2"
```

```
    },
    "scripts": {
        "test": "jest"
    }
}
```

Para instrumentar clientes do AWS SDK, encapsule a biblioteca `aws-sdk` com o método `captureAWS`.

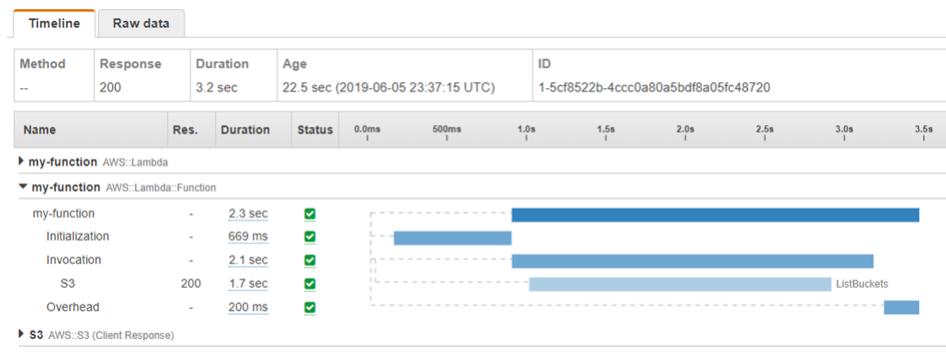
Example [blank-nodejs/function/index.js](#): rastrear um cliente do AWS SDK

```
const AWSXRay = require('aws-xray-sdk-core')
const AWS = AWSXRay.captureAWS(require('aws-sdk'))

// Create client outside of handler to reuse
const lambda = new AWS.Lambda()

// Handler
exports.handler = async function(event, context) {
    event.Records.forEach(record => {
        ...
    })
}
```

O exemplo a seguir mostra um rastreamento com 2 segmentos. Ambos são chamados `my-function`, mas um é do tipo `AWS::Lambda` e o outro é `AWS::Lambda::Function`. O segmento de função é expandido para mostrar seus subsegmentos.



O primeiro segmento representa a solicitação de invocação processada pelo serviço do Lambda. O segundo segmento registra o trabalho realizado pela sua função. O segmento de função tem 3 subsegmentos.

- Inicialização: representa o tempo gasto carregando a função e executando o [código de inicialização](#) (p. 30). Esse subsegmento só aparece para o primeiro evento processado por cada instância da função.
- Invocação: representa o trabalho realizado pelo código do handler. Ao instrumentar o código, é possível estender esse subsegmento com subsegmentos adicionais.
- Sobrecarga: representa o trabalho realizado pelo tempo de execução do Lambda como preparação para lidar com o próximo evento.

Você também pode instrumentar clientes HTTP, registrar consultas SQL e criar subsegmentos personalizados com anotações e metadados. Para obter mais informações, consulte [AWS X-Ray SDK for Node.js](#) no Guia do desenvolvedor do AWS X-Ray.

Seções

- [Habilitar o rastreamento ativo com a API do Lambda](#) (p. 536)
- [Habilitar o rastreamento ativo com o AWS CloudFormation](#) (p. 536)

- [Armazenar dependências de tempo de execução em uma camada \(p. 536\)](#)

Habilitar o rastreamento ativo com a API do Lambda

Para gerenciar a configuração de rastreamento com a AWS CLI ou com o AWS SDK, use as seguintes operações de API:

- [UpdateFunctionConfiguration \(p. 974\)](#)
- [GetFunctionConfiguration \(p. 851\)](#)
- [CreateFunction \(p. 796\)](#)

O exemplo de comando da AWS CLI a seguir habilita o rastreamento ativo em uma função chamada my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

O modo de rastreamento faz parte da configuração específica da versão que é bloqueada quando você publica uma versão da função. Não é possível alterar o modo de rastreamento em uma versão publicada.

Habilitar o rastreamento ativo com o AWS CloudFormation

Para habilitar o rastreamento ativo em um recurso `AWS::Lambda::Function` em um modelo do AWS CloudFormation, use a propriedade `TracingConfig`.

Example [function-inline.yml](#): configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
    ...
```

Para um recurso do AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, use a propriedade `Tracing`.

Example [template.yml](#): configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
      ...
    ...
```

Armazenar dependências de tempo de execução em uma camada

Se você usar o X-Ray SDK para instrumentar os clientes do AWS SDK com seu código de função, seu pacote de implantação poderá se tornar bastante grande. Para evitar o upload de dependências de

tempo de execução sempre que você atualizar seu código de funções, empacote-as em uma [camada do Lambda \(p. 85\)](#).

O exemplo a seguir mostra um recurso `AWS::Serverless::LayerVersion` que armazena o X-Ray SDK for Node.js.

Example [template.yml](#): camada de dependências

```
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:
        - nodejs12.x
```

Com essa configuração, você só atualizará a camada de biblioteca se alterar as dependências de tempo de execução. O pacote de implantação de função contém apenas o código. Ao atualizar o código de função, o tempo de upload será muito mais rápido do que se você incluir dependências no pacote de implantação.

A criação de uma camada de dependências requer alterações de compilação para gerar o arquivo da camada antes da implantação. Para obter um exemplo funcional, consulte o aplicativo de exemplo [blank-nodejs](#).

Criar funções do Lambda com Python

Você pode executar o código Python no AWS Lambda. Lambda fornece [Tempos de execução do \(p. 214\)](#) Python que executa o seu código para processar eventos. Seu código é executado em um ambiente que inclui o SDK for Python (Boto 3), com as credenciais de uma função do AWS Identity and Access Management (IAM) que você gerencia.

O Lambda oferece suporte aos seguintes tempos de execução Python.

Note

O término do suporte para o tempo de execução do Python 2.7 começou em 15 de julho de 2021. Para obter mais informações, consulte [the section called “Política de suporte ao tempo de execução” \(p. 217\).](#)

Tempos de execução do Python

Nome	Identifier	AWS SDK for Python	Sistema operacional	
Python 3.9	<code>python3.9</code>	boto3-1.17.100 botocore-1.20.100	Amazon Linux 2	
Python 3.8	<code>python3.8</code>	boto3-1.17.100 botocore-1.20.100	Amazon Linux 2	
Python 3.7	<code>python3.7</code>	boto3-1.17.100 botocore-1.20.100	Amazon Linux	
Python 3.6	<code>python3.6</code>	boto3-1.17.100 botocore-1.20.100	Amazon Linux	
Python 2.7	<code>python2.7</code>	boto3-1.17.100 botocore-1.20.100	Amazon Linux	

Para criar uma função do Python

1. Abra o [console do lambda](#).
2. Escolha Create function.
3. Configure as definições a seguir:
 - Nome – **my-function**.
 - Tempo de execução: Python 3.9.
 - Role (Função): Choose an existing role (Escolher uma função existente).
 - Existing role (Função existente – **lambda-role**).
4. Escolha Create function.
5. Para configurar um evento de teste, escolha Test (Testar).
6. Em Event name (Nome do evento), insira **test**.
7. Selecione Save changes.
8. Escolha Test (Testar) para invocar a função.

O console cria uma função do Lambda com um único arquivo de origem chamado `lambda_function`. Você pode editar esse arquivo e adicionar mais arquivos no [editor de códigos \(p. 40\)](#) integrado. Para salvar suas alterações, selecione Save (Salvar). Em seguida, para executar seu código, escolha Teste.

Note

O console do Lambda usa o AWS Cloud9 para fornecer um ambiente de desenvolvimento integrado no navegador. Você também pode usar o AWS Cloud9 para desenvolver funções do Lambda em seu próprio ambiente. Para obter mais informações, consulte [Working with Lambda Functions](#) no guia do usuário do AWS Cloud9.

Note

Para começar com o desenvolvimento de aplicativos no ambiente local, implante um dos aplicativos de exemplo disponíveis no repositório do GitHub deste guia.

Aplicativos Lambda de exemplo do em Python

- [blank-python](#): uma função Python que mostra o uso de registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK.

Sua função do Lambda é fornecida com um grupo de logs do CloudWatch Logs. O tempo de execução da função envia detalhes sobre cada invocação para o CloudWatch Logs. Ele retransmite qualquer [logs que sua função produz \(p. 553\)](#) durante a invocação. Se a função [retornar um erro \(p. 558\)](#), o Lambda formatará o erro e o retornará para o chamador.

Tópicos

- [Manipulador de função do Lambda em Python \(p. 540\)](#)
- [Implantar funções do Lambda em Python com arquivos .zip \(p. 543\)](#)
- [Implante funções do Lambda em Python com imagens de contêiner \(p. 548\)](#)
- [AWS LambdaObjeto de contexto em Python \(p. 551\)](#)
- [AWS LambdaRegistro em log da função do em Python \(p. 553\)](#)
- [AWS LambdaErros da função do em Python \(p. 558\)](#)
- [Instrumentação do código Python no AWS Lambda \(p. 562\)](#)

Manipulador de função do Lambda em Python

Note

O término do suporte para o tempo de execução do Python 2.7 começou em 15 de julho de 2021. Para obter mais informações, consulte [the section called “Política de suporte ao tempo de execução” \(p. 217\)](#).

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. Quando o manipulador é encerrado ou retorna uma resposta, ele se torna disponível para tratar de outro evento.

Você pode usar a seguinte sintaxe geral ao criar um manipulador de funções no Python:

```
def handler_name(event, context):
    ...
    return some_value
```

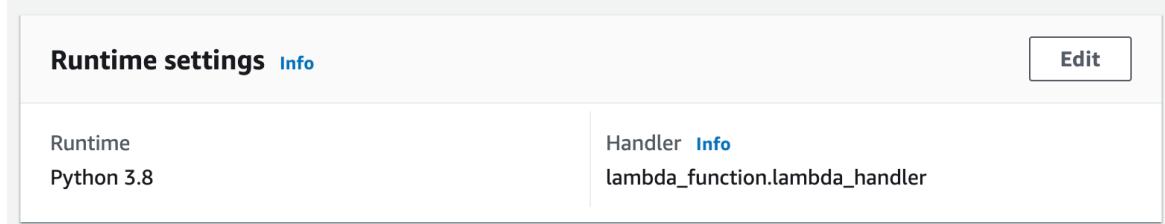
Naming

O nome do manipulador da função do Lambda especificado no momento em que você cria uma função do Lambda é derivado:

- do nome do arquivo no qual a função de manipulador do Lambda está localizada.
- do nome da função do manipulador do Python.

Um manipulador de funções pode ter qualquer nome; no entanto, o padrão no console do Lambda é `lambda_function.lambda_handler`. Esse nome de manipulador da função reflete o nome da função (`lambda_handler`) e o arquivo em que o código do manipulador está armazenado (`lambda_function.py`).

Para alterar o nome do manipulador de função no console do Lambda no [Configurações de execução](#) no painel, selecione [Edite](#).



Como funcionam

Quando o Lambda invoca seu manipulador de função, o [tempo de execução do Lambda \(p. 214\)](#) transmite dois argumentos ao manipulador da função:

- O primeiro argumento é o [objeto do evento](#). Um evento é um documento no formato JSON que contém dados para uma função do Lambda processar. O [tempo de execução do Lambda \(p. 214\)](#) converte o evento em um objeto e o transmite para o código da função. Ele geralmente é do tipo Python `dict`. Ele também pode ser do tipo `list`, `str`, `int`, `float`, ou `NoneType`.

O objeto do evento contém informações do serviço de chamada. Ao invocar uma função, você determina a estrutura e o conteúdo do evento. Quando um serviço da AWS invoca a função, ele define a estrutura

do evento. Para obter mais informações sobre eventos de serviços da AWS, consulte [Usar o AWS Lambda com outros serviços \(p. 277\)](#).

- O segundo argumento é o [objeto de contexto \(p. 551\)](#). Um objeto de contexto é passado para sua função pelo Lambda no tempo de execução. Esse objeto oferece métodos e propriedades que fornecem informações sobre a invocação, a função e o ambiente de tempo de execução.

Retornar um valor

Opcionalmente, um manipulador poderá retornar um valor. O que acontece com o valor retornado depende do [tipo de invocação \(p. 157\)](#) e do [serviço \(p. 277\)](#) que invocou a função. Por exemplo:

- Se você usar o tipo de invocação `RequestResponse`, como o [Invocação síncrona \(p. 158\)](#), o AWS Lambda retornará o resultado da chamada de função Python para o cliente que invoca a função do Lambda (na resposta HTTP à solicitação de invocação, serializada em JSON). Por exemplo, o console do AWS Lambda usa o tipo de invocação `RequestResponse`, assim, quando você invocar a função no console, ele exibirá o valor retornado.
- Se o manipulador retornar objetos que não podem ser serializados por `json.dumps`, o tempo de execução retornará um erro.
- Se o manipulador retornar `None`, assim como funções Python sem uma instrução `return` fazem implicitamente, o tempo de execução retornará `null`.
- Se você usar um `Event`, um tipo de invocação [Invocação assíncrona \(p. 161\)](#), o valor será descartado.

Note

No Python 3.9 e em versões posteriores, o Lambda inclui o `requestId` da invocação na resposta de erro.

Examples

A seção a seguir mostra exemplos de funções Python com as quais você pode usar o Lambda. Se você usar o console do Lambda para criar sua função, não será necessário associar um [arquivo .zip \(p. 543\)](#) para executar as funções nesta seção. Essas funções usam bibliotecas Python padrão que estão incluídas no tempo de execução do Lambda selecionado. Para obter mais informações, consulte [Pacotes de implantação do Lambda \(p. 37\)](#).

Retornar uma mensagem

O exemplo a seguir mostra uma função chamada `lambda_handler` que usa o `python3.8` [tempo de execução do Lambda \(p. 214\)](#). A função aceita a entrada do usuário de um nome e sobrenome, e retorna uma mensagem que contém dados do evento recebido como entrada.

```
def lambda_handler(event, context):
    message = 'Hello {} {}!'.format(event['first_name'], event['last_name'])
    return {
        'message' : message
    }
```

Você pode usar os seguintes dados de evento para [chamar a função](#):

```
{
    "first_name": "John",
    "last_name": "Smith"
}
```

A resposta mostra os dados do evento passados como entrada:

```
{  
    "message": "Hello John Smith!"  
}
```

Analisar uma resposta

O exemplo a seguir mostra uma função chamada `lambda_handler` que usa o `python3.8` [tempo de execução do Lambda \(p. 214\)](#). A função usa dados de evento passados pelo Lambda no tempo de execução. Ele analisa a [variável de ambiente \(p. 99\)](#) na `AWS_REGION` retornada na resposta JSON.

```
import os  
import json  
  
def lambda_handler(event, context):  
    json_region = os.environ['AWS_REGION']  
    return {  
        "statusCode": 200,  
        "headers": {  
            "Content-Type": "application/json"  
        },  
        "body": json.dumps({  
            "Region": json_region  
        })  
    }
```

Você pode usar quaisquer dados de eventos para [chamar a função](#):

```
{  
    "key1": "value1",  
    "key2": "value2",  
    "key3": "value3"  
}
```

Os tempos de execução do Lambda definem diversas variáveis de ambiente durante a inicialização. Para obter mais informações sobre as variáveis de ambiente retornadas na resposta no tempo de execução, consulte [Usar variáveis de ambiente do AWS Lambda \(p. 99\)](#).

A função neste exemplo depende de uma resposta bem-sucedida (em 200) da API Invoke. Para obter mais informações sobre o status da API de invocação, consulte a Sintaxe de resposta [Invoke \(p. 875\)](#).

Retornar um cálculo

O exemplo a seguir de [código de função do Lambda em Python no GitHub](#) mostra uma função chamada `lambda_handler` que usa o [tempo de execução do Lambda \(p. 214\)](#) `python3.6`. A função aceita a entrada do usuário e retorna um cálculo para ele.

Você pode usar os seguintes dados de evento para [chamar a função](#):

```
{  
    "action": "increment",  
    "number": 3  
}
```

Implantar funções do Lambda em Python com arquivos .zip

Note

O término do suporte para o tempo de execução do Python 2.7 começou em 15 de julho de 2021. Para obter mais informações, consulte [the section called “Política de suporte ao tempo de execução” \(p. 217\)](#).

O código da função do AWS Lambda consiste em scripts ou programas compilados e as dependências deles. Você usa um pacote de implantação para implantar seu código de função no Lambda. O Lambda é compatível com dois tipos de pacotes de implantação: imagens de contêiner e arquivos .zip.

Para criar um pacote de implantação compactado em um arquivo .zip, você pode usar um utilitário de compactação em arquivo .zip integrado ou qualquer outro utilitário semelhante (como o [7zip](#)) para sua ferramenta de linha de comando. Lembre-se dos seguintes requisitos para usar um arquivo .zip como seu pacote de implantação:

- O arquivo .zip contém o código da sua função e quaisquer dependências usadas para executá-lo (se aplicável) no Lambda. Se sua função depender apenas de bibliotecas padrão ou de bibliotecas do SDK do AWS, você não precisará incluí-las em seu arquivo .zip. Essas bibliotecas são incluídas nos ambientes de [tempo de execução do Lambda \(p. 214\)](#) suportados.
- Se o arquivo.zip for maior que 50 MB, recomendamos enviá-lo para sua função a partir de um bucket do Amazon Simple Storage Service (Amazon S3).
- Se o pacote de implantação contiver bibliotecas nativas, você poderá criar o pacote de implantação com o AWS Serverless Application Model (AWS SAM). É possível usar o comando AWS SAM da CLI do `sam build` com `--use-container` para criar seu pacote de implantação. Essa opção cria um pacote de implantação dentro de uma imagem do Docker compatível com o ambiente de execução do Lambda.

Para obter mais informações, consulte [sam build](#) no Guia do desenvolvedor do AWS Serverless Application Model.

- O Lambda usa permissões de arquivo POSIX, então pode ser necessário [definir permissões para a pasta do pacote de implantação](#) antes de criar o arquivo .zip.

Note

Um pacote python pode conter código de inicialização no arquivo `__init__.py`. Antes do Python 3.9, o Lambda não executava o código `__init__.py` para pacotes no diretório do manipulador de funções ou diretórios pai. No Python 3.9 e em versões posteriores, o Lambda executa o código `init` para pacotes nesses diretórios durante a inicialização.

Observe que o Lambda executa o código de inicialização somente quando o ambiente de execução é inicializado pela primeira vez, não para cada invocação de função nesse ambiente inicializado.

Tópicos

- [Prerequisites \(p. 544\)](#)
- [O que é uma dependência de runtime? \(p. 544\)](#)
- [Pacote de implantação sem dependências \(p. 544\)](#)
- [Pacote de implantação com dependências \(p. 545\)](#)
- [Usar um ambiente virtual \(p. 546\)](#)
- [Adicionar o arquivo .zip à função \(p. 546\)](#)

Prerequisites

Você precisa do AWS Command Line Interface (AWS CLI) para chamar operações da API de serviço. Para instalar a AWS CLI, consulte [Installing the AWS CLI](#) no Manual do usuário do AWS Command Line Interface.

O que é uma dependência de runtime?

Um [pacote de implantação \(p. 37\)](#) é necessário para a criação ou a atualização de uma função do Lambda com ou sem dependências de tempo de execução. O pacote de implantação atuará como o pacote de origem para a execução do código e das dependências (se aplicável) da sua função no Lambda.

Uma dependência pode ser qualquer pacote, módulo ou outra dependência de assembly que não esteja incluída com o [Tempo de execução do Lambda \(p. 214\)](#) para o código da sua função.

O exemplo a seguir descreve uma função do Lambda sem dependências de tempo de execução:

- Se o código da sua função estiver em Python 3.8 ou posterior e depender apenas das bibliotecas padrão de matemática e registro em log do Python, você não precisará inclui-las em seu arquivo .zip. Essas bibliotecas estão incluídas no tempo de execução do Python.
- Se o código da sua função depender do [AWS SDK for Python \(Boto3\)](#), você não precisará incluir a biblioteca boto3 em seu arquivo .zip. Essas bibliotecas estão incluídas no tempo de execução do Python 3.8 e posterior.

Observação: o Lambda atualiza automaticamente as bibliotecas Boto3 para habilitar o conjunto de recursos e atualizações de segurança mais recentes. Para ter controle total das dependências usadas por sua função, empacote todas as dependências em seu pacote de implantação.

Pacote de implantação sem dependências

Criar o arquivo .zip para seu pacote de implantação.

Para criar o pacote de implantação

1. Abra um prompt de comando e crie um diretório de projeto do `my-math-function`. Por exemplo, no macOS:

```
mkdir my-math-function
```

2. Navegue até o diretório de projeto do `my-math-function`.

```
cd my-math-function
```

3. Copie o conteúdo do [código Python de exemplo do GitHub](#) e salve-o em um novo arquivo chamado `lambda_function.py`. A estrutura do seu diretório deve ficar assim:

```
my-math-function$  
| lambda_function.py
```

4. Adicione o arquivo `lambda_function.py` à raiz do arquivo .zip.

```
zip my-deployment-package.zip lambda_function.py
```

Isso gerará um arquivo `my-deployment-package.zip` no diretório do projeto. O comando produzirá a saída a seguir:

```
adding: lambda_function.py (deflated 50%)
```

Pacote de implantação com dependências

Criar o arquivo .zip para seu pacote de implantação.

Para criar o pacote de implantação

1. Abra um prompt de comando e crie um diretório de projeto do `my-sourcecode-function`. Por exemplo, no macOS:

```
mkdir my-sourcecode-function
```

2. Navegue até o diretório de projeto do `my-sourcecode-function`.

```
cd my-sourcecode-function
```

3. Copie o conteúdo do seguinte código Python de exemplo e salve-o em um novo arquivo chamado `lambda_function.py`:

```
import requests
def main(event, context):
    response = requests.get("https://www.test.com/")
    print(response.text)
    return response.text
if __name__ == "__main__":
    main('', '')
```

A estrutura do seu diretório deve ficar assim:

```
my-sourcecode-function$  
| lambda_function.py
```

4. Instale a biblioteca `requests` em um novo diretório do package.

```
pip install --target ./package requests
```

5. Crie um pacote de implantação com a biblioteca instalada na raiz.

```
cd package
zip -r ../my-deployment-package.zip .
```

Isso gerará um arquivo `my-deployment-package.zip` no diretório do projeto. O comando produzirá a saída a seguir:

```
adding: chardet/ (stored 0%)
adding: chardet/enums.py (deflated 58%)
...
```

6. Adicione o arquivo `lambda_function.py` à raiz do arquivo zip.

```
cd ..
zip -g my-deployment-package.zip lambda_function.py
```

Usar um ambiente virtual

Como atualizar uma função do Python com um ambiente virtual

1. Ative o ambiente virtual. Por exemplo:

```
~/my-function$ source myvenv/bin/activate
```

2. Instale as bibliotecas com o pip.

```
(myvenv) ~/my-function$ pip install requests
```

3. Desative o ambiente virtual.

```
(myvenv) ~/my-function$ deactivate
```

4. Crie um pacote de implantação com as bibliotecas instaladas na raiz.

```
~/my-function$ cd myvenv/lib/python3.8/site-packages  
zip -r ../../../../my-deployment-package.zip .
```

O último comando salva o pacote de implantação na raiz do diretório `my-function`.

Tip

Uma biblioteca pode aparecer em `site-packages` ou `dist-packages` a primeira pasta `lib` ou `lib64`. Use o comando `pip show` para localizar um pacote específico.

5. Adicione arquivos de código de função à raiz do seu pacote de implantação.

```
~/my-function/myvenv/lib/python3.8/site-packages$ cd ../../../../../../  
~/my-function$ zip -g my-deployment-package.zip lambda_function.py
```

Depois de concluir esta etapa, você deverá ter seguinte estrutura de diretórios:

```
my-deployment-package.zip$  
# lambda_function.py  
# __pycache__  
# certifi/  
# certifi-2020.6.20.dist-info/  
# chardet/  
# chardet-3.0.4.dist-info/  
...
```

Adicionar o arquivo .zip à função

Para implantar o novo código em sua função, você carrega o novo pacote de implantação de arquivo.zip. Você pode usar o [console do Lambda](#) (p. 83) para carregar um arquivo.zip para a função, ou você pode usar o comando [UpdateFunctionCode](#) (p. 965) da CLI.

O exemplo a seguir carrega um arquivo chamado `my-deployment-package.zip`. Use o prefixo `fileb://` para carregar o arquivo .zip binário no Lambda.

```
~/my-function$ aws lambda update-function-code --function-name MyLambdaFunction --zip-file  
fileb://my-deployment-package.zip  
{
```

```
"FunctionName": "mylambdafunction",
"FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:mylambdafunction",
"Runtime": "python3.8",
"Role": "arn:aws:iam::123456789012:role/lambda-role",
"Handler": "lambda_function.lambda_handler",
"CodeSize": 5912988,
"CodeSha256": "A2P0NUWq1J+LtSbkuP8tm9uNYqs1TAa3M76ptmZCw5g=",
"Version": "$LATEST",
"RevisionId": "5afdc7dc-2fcb-4ca8-8f24-947939ca707f",
...
}
```

Implante funções do Lambda em Python com imagens de contêiner

Note

O término do suporte para o tempo de execução do Python 2.7 começou em 15 de julho de 2021. Para obter mais informações, consulte [the section called “Política de suporte ao tempo de execução” \(p. 217\)](#).

Você pode implantar seu código de função do Lambda como uma [imagem de contêiner \(p. 267\)](#). A AWS fornece os seguintes recursos para ajudá-lo a criar uma imagem de contêiner para sua função Python:

- AWSImagens base para Lambda

Essas imagens base são pré-carregadas com um tempo de execução de linguagem e outros componentes necessários para executar a imagem no Lambda. AWS fornece um Dockerfile para cada uma das imagens de base para ajudar a criar sua imagem de contêiner.

- Clientes de interface de tempo de execução de

Se você usar uma imagem de base de comunidade ou de empresa privada, adicione um cliente de interface de tempo de execução à imagem base para torná-lo compatível com o Lambda.

O fluxo de trabalho de uma função definida como uma imagem de contêiner inclui as seguintes etapas:

1. Crie sua imagem de contêiner usando os recursos listados neste tópico.
2. Faça upload da imagem em seu registro do contêiner do Amazon ECR. Consulte os passos 7 a 9 no [Criar imagem \(p. 268\)](#).
3. [Crie \(p. 91\)](#) a função do Lambda ou [atualize o código da função \(p. 92\)](#) para implantar a imagem em uma função existente.

Tópicos

- [AWS imagens base para Python \(p. 548\)](#)
- [Clientes de interface de tempo de execução Python \(p. 549\)](#)
- [Criar uma imagem Python a partir de uma imagem básica da AWS \(p. 549\)](#)
- [Criar uma imagem Python a partir de uma imagem base alternativa \(p. 550\)](#)
- [Implantar a imagem do contêiner \(p. 550\)](#)

AWS imagens base para Python

AWS fornece as seguintes imagens base para Python:

Tags	Tempo de execução	Sistema operacional	Dockerfile
3, 3.9	Python 3.9	Amazon Linux 2	Dockerfile para Python 3.9 no GitHub
3.8	Python 3.8	Amazon Linux 2	Dockerfile para Python 3.8 no GitHub
3.7	Python 3.7	Amazon Linux 2018.03	Dockerfile para Python 3.7 no GitHub

Tags	Tempo de execução	Sistema operacional	Dockerfile
3.6	Python 3.6	Amazon Linux 2018.03	Dockerfile para Python 3.6 no GitHub
2, 2.7	Python 2.7	Amazon Linux 2018.03	Dockerfile para Python 2.7 no GitHub

Repositório do Docker Hub: [amazon/aws-lambda-python](#)

Repositório do Amazon ECR: [gallery.ecr.aws/lambda/python](#)

Cientes de interface de tempo de execução Python

Instale o cliente de interface de tempo de execução para Python usando o gerenciador de pacotes pip:

```
pip install awslambdaric
```

Para obter detalhes do pacote, consulte [Lambda RIC no site Python Package Index \(PyPI\)](#).

Você também pode baixar o [cliente de interface de tempo de execução Python](#) no GitHub.

Criar uma imagem Python a partir de uma imagem básica da AWS

Quando você cria uma imagem de contêiner para Python usando uma imagem AWS base, você só precisa copiar o aplicativo python para o contêiner e instalar quaisquer dependências.

Para criar e implantar uma função Python com a imagem **python:3.8** base.

1. Em sua máquina local, crie um diretório de projeto para sua nova função.
2. No diretório do projeto, adicione um arquivo chamado `app.py` contendo seu código de função. O exemplo a seguir mostra um handler do Python simples.

```
import sys
def handler(event, context):
    return 'Hello from AWS Lambda using Python' + sys.version + '!'
```

3. No diretório de seu projeto, adicione um arquivo chamado `requirements.txt`. Liste cada biblioteca necessária como uma linha separada neste arquivo. Deixe o arquivo vazio se não houver dependências.
4. Use um editor de texto para criar um Dockerfile no diretório do projeto. O exemplo a seguir mostra o Dockerfile para o manipulador que você criou na etapa anterior. Instale qualquer dependência no diretório `#{LAMBDA_TASK_ROOT}` junto do manipulador de funções para garantir que o tempo de execução do Lambda possa localizá-la quando a função for invocada.

```
FROM public.ecr.aws/lambda/python:3.8

# Copy function code
COPY app.py ${LAMBDA_TASK_ROOT}

# Install the function's dependencies using file requirements.txt
# from your project folder.

COPY requirements.txt .
```

```
RUN pip3 install -r requirements.txt --target "${LAMBDA_TASK_ROOT}"  
  
# Set the CMD to your handler (could also be done as a parameter override outside of  
the Dockerfile)  
CMD [ "app.handler" ]
```

5. Para criar a imagem do contêiner, siga as etapas de 4 a 7 em [Criar uma imagem a partir de uma imagem básica da AWS para o Lambda \(p. 268\)](#).

Criar uma imagem Python a partir de uma imagem base alternativa

Para obter um exemplo de como criar uma imagem Python a partir de uma imagem base Alpine, consulte [Suporte de imagem de contêiner para o Lambda](#) no AWS Blog.

Implantar a imagem do contêiner

Para uma nova função, você implanta a imagem do Python ao [criar a função \(p. 91\)](#). Para uma função existente, se você reconstruir a imagem do contêiner, precisará reimplantar a imagem ao [atualizar o código da função \(p. 92\)](#).

AWS LambdaObjeto de contexto em Python

Note

O término do suporte para o tempo de execução do Python 2.7 começou em 15 de julho de 2021. Para obter mais informações, consulte [the section called “Política de suporte ao tempo de execução” \(p. 217\)](#).

Quando o Lambda executa a função, ele transmite um objeto de contexto para o [handler \(p. 540\)](#). Esse objeto fornece métodos e propriedades que fornecem informações sobre a invocação, a função e o ambiente de execução. Para obter mais informações sobre como o objeto de contexto é passado para o manipulador de funções, consulte [Manipulador de função do Lambda em Python \(p. 540\)](#).

Métodos de contexto

- `get_remaining_time_in_millis`: retorna o número de milissegundos restantes antes do tempo limite da execução.

Propriedades de contexto

- `function_name`: o nome da função do Lambda.
- `function_version`: a [versão \(p. 106\)](#) da função.
- `invoked_function_arn`: o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um alias ou número de versão.
- `memory_limit_in_mb`: a quantidade de memória alocada para a função.
- `aws_request_id`: o identificador da solicitação de invocação.
- `log_group_name`: o grupo de logs da função.
- `log_stream_name`: a transmissão de log para a instância da função.
- `identity`: (aplicativos móveis) informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
 - `cognito_identity_id`: a identidade autenticada do Amazon Cognito.
 - `cognito_identity_pool_id`: o grupo de identidades do Amazon Cognito que autorizou a invocação.
- `client_context`: (aplicativos móveis) contexto do cliente fornecido ao Lambda pela aplicação cliente.
 - `client.installation_id`
 - `client.app_title`
 - `client.app_version_name`
 - `client.app_version_code`
 - `client.app_package_name`
- `custom`: um dict de valores personalizados definidos pela aplicação cliente para dispositivos móveis.
- `env`: um dict de informações do ambiente fornecidas pelo AWS SDK.

O exemplo a seguir mostra uma função do handler que registra informações de contexto.

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_timeInMillis.
time.sleep(1)
print("Lambda time remaining in MS:", context.get_remaining_timeInMillis())
```

Além das opções listadas acima, você também pode usar o AWS X-Ray SDK para [Instrumentação do código Python no AWS Lambda \(p. 562\)](#) para identificar caminhos de código críticos, rastrear a performance e capturar os dados para análise.

AWS Lambda Registro em log da função do em Python

Note

O término do suporte para o tempo de execução do Python 2.7 começou em 15 de julho de 2021. Para obter mais informações, consulte [the section called “Política de suporte ao tempo de execução” \(p. 217\)](#).

O AWS Lambda monitora automaticamente as funções do Lambda em seu nome e envia métricas da função para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente do tempo de execução do Lambda envia detalhes sobre cada invocação à transmissão de logs e transmite os logs e outras saídas do código de sua função.

Esta página descreve como produzir a saída de logs usando o código de sua função do Lambda ou acessar os logs usando a AWS Command Line Interface, o console do Lambda ou o console do CloudWatch.

Seções

- [Criar uma função que retorna logs \(p. 553\)](#)
- [Usar o console do Lambda \(p. 554\)](#)
- [Usando o console do CloudWatch \(p. 554\)](#)
- [Usar a AWS Command Line Interface \(AWS CLI\) \(p. 554\)](#)
- [Excluir logs \(p. 557\)](#)
- [Biblioteca de registros em log \(p. 557\)](#)

Criar uma função que retorna logs

Para gerar os logs do seu código da função, você pode usar o método `print` ou qualquer biblioteca de logs que grava no `stdout` ou `stderr`. O exemplo a seguir registra em log os valores das variáveis de ambiente e o objeto do evento.

Example `lambda_function.py`

```
import os

def lambda_handler(event, context):
    print('## ENVIRONMENT VARIABLES')
    print(os.environ)
    print('## EVENT')
    print(event)
```

Example formato do 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: 16 ms Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 130.49 ms XRAY TraceId: 1-5e34a614-10bdxmplf1fb44f07bc535a1 SegmentId: 07f5xmpl2d1f6f85 Sampled: true
```

O tempo de execução do Python registra as linhas START, END e REPORT para cada invocação. A linha do relatório fornece os seguintes detalhes.

Registro em log de relatório

- RequestId: o ID de solicitação exclusivo para a invocação.
- Duração: a quantidade de tempo que o método de manipulador da função gastou processando o evento.
- Duração faturada: a quantia de tempo faturada para a invocação.
- Tamanho da memória: a quantidade de memória alocada para a função.
- Memória máxima utilizada: a quantidade de memória utilizada pela função.
- Duração inicial: para a primeira solicitação atendida, a quantidade de tempo que o tempo de execução levou para carregar a função e executar o código fora do método do manipulador.
- XRAY TraceId: para solicitações rastreadas, o [ID de rastreamento do AWS X-Ray \(p. 477\)](#).
- SegmentId: para solicitações rastreadas, o ID do segmento do X-Ray.
- Amostragem: para solicitações rastreadas, o resultado da amostragem.

Usar o console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda. Para obter mais informações, consulte [Acessar o Amazon CloudWatch Logs para o AWS Lambda \(p. 720\)](#).

Usando o console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a [página Log groups](#) (Grupos de log).
2. Escolha o grupo de logs de sua função (`/aws/lambda/nome-de-sua-função`).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma [instância da sua função \(p. 219\)](#). Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos instrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

Para usar uma aplicação de exemplo que correlaciona os logs e os rastreamentos com o X-Ray, consulte [Aplicativo de exemplo de processador de erros para o AWS Lambda \(p. 500\)](#).

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Você pode usar a [AWS CLI](#) para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Você deve ver a saída a seguir:

```
{  
    "StatusCode": 200,  
    "LogResult":  
        "U1RBULQgUmVxdWVzdElkOia4N2QwNDRiOC1mMTU0LTExZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb... ",  
    "ExecutedVersion": "$LATEST"  
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \  
--query 'LogResult' --output text | base64 -d
```

Você deve ver a saída a seguir:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e [Ubuntu no Windows](#). Os usuários do macOS precisam usar `base64 -D`.

Example `get-logs.sh` script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa `sed` para remover as aspas do arquivo de saída e fica inativo por 15 segundos para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como `get-logs.sh`.

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

```
#!/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 $(cat  
out) --limit 5
```

Example macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod -R 755 get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

Você deve ver a saída a seguir:

```
{  
    "StatusCode": 200,  
    "ExecutedVersion": "$LATEST"  
}  
{  
    "events": [  
        {  
            "timestamp": 1559763003171,  
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:  
$LATEST\n",  
            "ingestionTime": 1559763003309  
        },  
        {  
            "timestamp": 1559763003173,  
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf  
\tINFO\tENVIRONMENT VARIABLES\r{\r\t\t\"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\",  
            "ingestionTime": 1559763018353  
        },  
        {  
            "timestamp": 1559763003173,  
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf  
\tINFO\tEVENT\r{\r\t\t\"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  
        }  
}
```

Excluir logs

Os grupos de logs não são excluídos automaticamente excluídos quando você exclui uma função. Para evitar armazenar logs indefinidamente, exclua o grupo de logs ou[Configurar um período de retenção](#)após o qual os logs são excluídos automaticamente.

Biblioteca de registros em log

Para logs mais detalhados, use [a biblioteca de logs](#).

```
import os
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES')
    logger.info(os.environ)
    logger.info('## EVENT')
    logger.info(event)
```

A saída de logger inclui o nível do log, o timestamp e o ID da solicitação.

```
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: 3 ms Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 113.51 ms
XRAY TraceId: 1-5e34a66a-474xmpl7c2534a87870b4370 SegmentId: 073cxmpl3e442861 Sampled:
true
```

AWS Lambda Erros da função do em Python

Note

O término do suporte para o tempo de execução do Python 2.7 começou em 15 de julho de 2021. Para obter mais informações, consulte [the section called “Política de suporte ao tempo de execução” \(p. 217\)](#).

Quando o código gera um erro, o Lambda gera uma representação JSON do erro. Esse documento de erro aparece no log de invocação e, para invocações síncronas, na saída.

Esta página descreve como exibir erros de invocação de função do Lambda para o tempo de execução do Python usando o console do Lambda e a AWS CLI.

Seções

- [Como funcionam \(p. 558\)](#)
- [Usar o console do Lambda \(p. 559\)](#)
- [Usar a AWS Command Line Interface \(AWS CLI\) \(p. 559\)](#)
- [Tratamento de erros em outros serviços da AWS \(p. 560\)](#)
- [Exemplos de erro \(p. 560\)](#)
- [Aplicativos de exemplo \(p. 561\)](#)
- [Próximas etapas \(p. 532\)](#)

Como funcionam

Ao invocar uma função do Lambda, o Lambda recebe a solicitação de invocação e valida as permissões de sua função de execução, verifica se o documento do evento é um documento JSON válido e verifica valores de parâmetros.

Se a solicitação for validada, o Lambda a envia para uma instância da função. O ambiente de [tempo de execução do Lambda \(p. 214\)](#) converte o documento do evento em um objeto e o transmite ao handler da função.

Se o Lambda encontra um erro, ele retorna um tipo de exceção, uma mensagem e o código HTTP do status que indica a causa do erro. O cliente ou o serviço que invocou a função do Lambda pode processar o erro de maneira programática ou transmiti-lo a um usuário final. O comportamento correto de tratamento de erros depende do tipo de aplicativo, do público e da origem do erro.

A lista a seguir descreve o intervalo de códigos de status que você pode receber do Lambda.

2xx

Um erro da série 2xx com um cabeçalho `X-Amz-Function-Error` na resposta indica um erro de tempo de execução ou de função do Lambda. Um código de status da série 2xx indica que o Lambda aceitou a solicitação, mas em vez de um código de erro, o Lambda indica o erro incluindo o cabeçalho `X-Amz-Function-Error` na resposta.

4xx

Um erro da série 4xx indica um erro que o cliente ou serviço que fez a invocação pode corrigir modificando a solicitação, solicitando permissão ou tentando a solicitação novamente. Os erros da série 4xx diferentes de 429 geralmente indicam um erro na solicitação.

5xx

Um erro da série 5xx indica um problema com o Lambda ou com a configuração/recursos da função. Os erros da série 5xx podem indicar uma condição temporária que pode ser resolvida sem nenhuma

ação do usuário. O cliente ou serviço que fez a invocação não podem solucionar esses problemas, mas o proprietário de uma função do Lambda pode ser capaz de corrigi-los.

Para obter uma lista completa de erros de invocação, consulte [Erros de InvokeFunction \(p. 877\)](#).

Usar o console do Lambda

Você pode invocar sua função no console do Lambda configurando um evento de teste e visualizando a saída. A saída é capturada pelos logs de execução da função e, quando o [rastreamento ativo \(p. 477\)](#) está habilitado, pelo AWS X-Ray.

Para invocar uma função no console do Lambda

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Test (Testar).
4. Selecione New event (Novo evento) e escolha um Event template (Modelo de evento) na lista suspensa.
5. Insira um nome para o evento de teste.
6. Digite o JSON para o evento de teste.
7. Escolha Create event (Criar evento).
8. Escolha Invoke (Invocar).

O console do Lambda invoca sua função [de forma síncrona \(p. 158\)](#) e exibe o resultado. Para ver a resposta, os logs e outras informações, expanda a seção Details (Detalhes).

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Quando você invoca uma função do Lambda na AWS CLI, a AWS CLI divide a resposta em dois documentos. A resposta da AWS CLI é exibida no prompt de comando. Se ocorreu um erro, a resposta contém um campo `FunctionError`. A resposta ou o erro de invocação retornado pela função é gravado no arquivo de saída. Por exemplo, o `output.json` ou o `output.txt`.

O exemplo de comando `invoke` a seguir demonstra como invocar uma função e escrever a resposta de invocação em um arquivo `output.txt`.

```
aws lambda invoke \
--function-name my-function \
--cli-binary-format raw-in-base64-out \
--payload '{"key1": "value1", "key2": "value2", "key3": "value3"}' output.txt
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

Você deve ver a resposta da AWS CLI em seu prompt de comando:

```
{
```

```
"StatusCode": 200,  
"FunctionError": "Unhandled",  
"ExecutedVersion": "$LATEST"  
}
```

Você deve ver a resposta de invocação de função no arquivo `output.txt`. Também é possível visualizar a saída no mesmo prompt de comando usando:

```
cat output.txt
```

Você deve ver a resposta de invocação no prompt de comando.

```
{"errorMessage": "action'", "errorType": "KeyError", "stackTrace": ["  File \"/var/task/lambda_function.py\", line 36, in lambda_handler\n      result = ACTIONS[event['action']]  
(event['number'])\n"]}
```

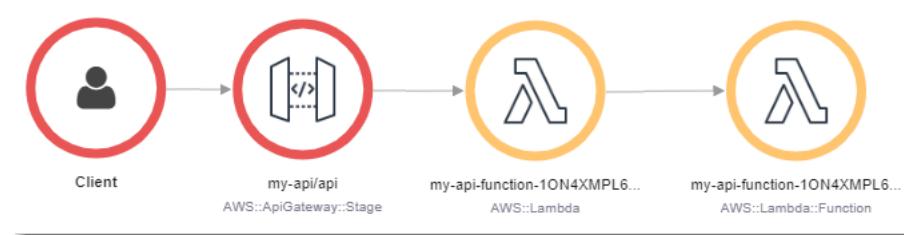
O Lambda também grava até 256 KB do objeto de erro nos logs de função. Para obter mais informações, consulte [AWS Lambda Registro em log da função em Python \(p. 553\)](#).

Tratamento de erros em outros serviços da AWS

Quando outro serviço da AWS invoca sua função, o serviço escolhe o tipo de invocação e o comportamento de repetição. Os serviços da AWS podem invocar sua função em um agendamento, em resposta a um evento de ciclo de vida em um recurso ou para atender a uma solicitação de um usuário. Alguns serviços invocam funções de forma assíncrona e permitem que o Lambda trate erros, enquanto outros fazem novas tentativas ou transmitem os erros de volta ao usuário.

Por exemplo, o API Gateway trata todos os erros de invocação e função como erros internos. Se a API do Lambda rejeitar a solicitação de invocação, o API Gateway retornará um código de erro 500. Se a função é executada, mas retorna um erro ou retorna uma resposta no formato errado, o API Gateway retorna um código de erro 502. Para personalizar a resposta de erro, é necessário detectar erros no código e formatar uma resposta no formato necessário.

Recomendamos usar AWS X-Ray para determinar a fonte de um erro e a respectiva causa. O X-Ray permite localizar qual componente encontrou um erro e ver detalhes sobre os erros. O exemplo a seguir mostra um erro de função que resultou em uma resposta 502 do API Gateway.



Para obter mais informações, consulte [Instrumentação do código Python no AWS Lambda \(p. 562\)](#).

Exemplos de erro

A seção a seguir mostra erros comuns que você pode receber ao criar, atualizar ou invocar sua função usando o Python [Tempos de execução do Lambda \(p. 214\)](#).

Example Exceção de tempo de execução: ImportError

```
{
```

```
{ "errorMessage": "Unable to import module 'lambda_function': Cannot import name '_imaging'\nfrom 'PIL' (/var/task/PIL/_init__.py)",\n  "errorType": "Runtime.ImportModuleError"\n}
```

Esse erro é resultado do uso da AWS Command Line Interface (AWS CLI) para fazer upload de um pacote de implantação que contém uma biblioteca C ou C++. Por exemplo, as bibliotecas [Pillow \(PIL\)](#), [numpy](#) ou [pandas](#).

Recomendamos usar o comando [sam build](#) da CLI do AWS SAM com a opção `--use-container` para criar seu pacote de implantação. Usar a AWS SAM CLI com essa opção cria um contêiner do Docker com um ambiente semelhante ao Lambda compatível com o Lambda.

Example Erro de serialização JSON: Runtime.MarshalError

```
{ "errorMessage": "Unable to marshal response: Object of type AttributeError is not JSON\nserializable",\n  "errorType": "Runtime.MarshalError"\n}
```

Esse erro pode ser o resultado do mecanismo de codificação base64 que você está usando em seu código de função. Por exemplo:

```
import base64\nencrypted_data = base64.b64encode(payload_enc).decode("utf-8")
```

Esse erro também pode ser o resultado de não especificar seu arquivo .zip como um arquivo binário quando você criou ou atualizou sua função. Recomendamos usar a opção de comando [fileb://](#) para fazer upload de seu pacote de implantação (arquivo .zip).

```
aws lambda create-function --function-name my-function --zip-file fileb://my-deployment-\npackage.zip --handler lambda_function.lambda_handler --runtime python3.8 --role\narn:aws:iam::your-account-id:role/lambda-ex
```

Aplicativos de exemplo

O repositório do GitHub para este guia inclui aplicativos de exemplo que demonstram o uso dos erros. Cada aplicativo de exemplo inclui scripts para fácil implantação e limpeza, um modelo do AWS Serverless Application Model (AWS SAM) e recursos de suporte.

Aplicativos de exemplo do Lambda em Python

- [blank-python](#): uma função Python que mostra o uso de registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK.

Próximas etapas

- Saiba como mostrar eventos de registro em log para sua função do Lambda na página [the section called “Registro em log” \(p. 553\)](#).

Instrumentação do código Python no AWS Lambda

Note

O término do suporte para o tempo de execução do Python 2.7 começou em 15 de julho de 2021. Para obter mais informações, consulte [the section called “Política de suporte ao tempo de execução” \(p. 217\)](#).

O Lambda se integra ao AWS X-Ray para permitir que você rastreie, depure e otimize aplicativos do Lambda. É possível usar o X-Ray para rastrear uma solicitação à medida que ela atravessa recursos na aplicação, da API de frontend ao armazenamento e aos banco de dados no backend. Ao simplesmente adicionar a biblioteca do X-Ray SDK à configuração de compilação, é possível registrar erros e latência para qualquer chamada que a função faça para um serviço da AWS.

O X-RayMapa de serviço mostra o fluxo de solicitações através de seu aplicativo. O exemplo a seguir do aplicativo de exemplo [processador de erros \(p. 500\)](#) mostra um aplicativo com duas funções. A função principal processa eventos e, às vezes, retorna erros. A segunda função processa erros que aparecem no primeiro grupo de logs e usa o AWS SDK para chamar o X-Ray, o Amazon S3 e o Amazon CloudWatch Logs.



Para rastrear solicitações que não têm um cabeçalho de rastreamento, ative o rastreamento ativo na configuração da sua função.

Para ativar o rastreamento ativo

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Selecione **Configuração**, depois, escolha **Ferramentas de monitoramento**.
4. Selecione **Edit**.
5. Em X-Ray, habilite o **Active tracing (Rastreamento ativo)**.
6. Escolha **Save (Salvar)**.

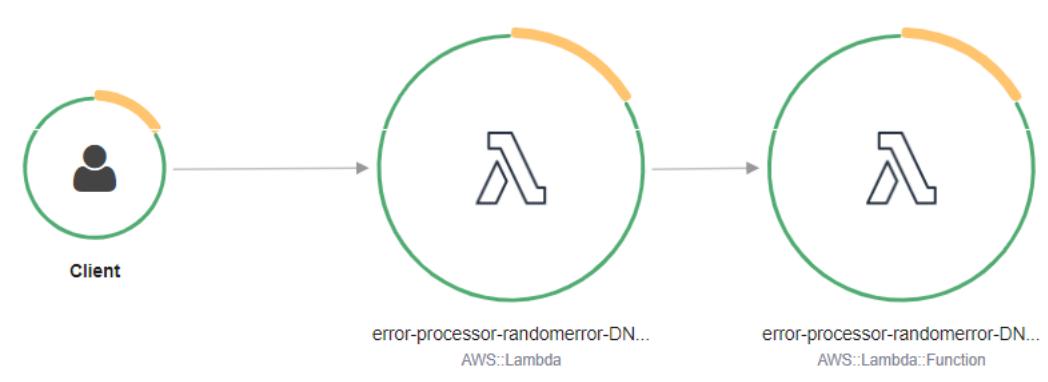
Pricing

O X-Ray tem um nível gratuito vitalício. Além do limite do nível gratuito, o X-Ray cobra por armazenamento e recuperação de rastreamento. Para obter detalhes, consulte [Definição de preço do AWS X-Ray](#).

Sua função precisa de permissão para carregar dados de rastreamento no X-Ray. Quando você habilita o rastreamento ativo no console do Lambda, o Lambda adiciona as permissões necessárias à [função de execução \(p. 57\)](#) da função. Caso contrário, adicione a política `AWSXRayDaemonWriteAccess` à função de execução.

O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento é eficiente, enquanto ainda fornece uma amostra representativa das solicitações que a sua aplicação serviu. A regra de amostragem padrão é uma solicitação por segundo e 5% de solicitações adicionais. Esta taxa de amostragem não pode ser configurada para funções do Lambda.

Quando o rastreamento ativo está habilitado, o Lambda registra um rastreamento para um subconjunto de invocações. Lambda registra doisSegmentos do, que cria dois nós no mapa de serviço. O primeiro nó representa o serviço Lambda que recebe a solicitação de invocação. O segundo nó é gravado pelo [tempo de execução \(p. 19\)](#)da função.



É possível instrumentar o código do manipulador para gravar metadados e rastrear chamadas downstream. Para registrar detalhes sobre chamadas feitas pelo manipulador para outros recursos e serviços, use o X-Ray SDK for Python. Para obter o SDK, adicione o pacote `aws-xray-sdk` às dependências do aplicativo.

Example [blank-python/function/requirements.txt](#)

```
jsonpickle==1.3
aws-xray-sdk==2.4.3
```

Para instrumentar clientes do AWS SDK, aplique um patch à biblioteca `boto3` com o módulo `aws_xray_sdk.core`.

Example [blank-python/function/lambda_function.py](#): rastrear um cliente do AWS SDK

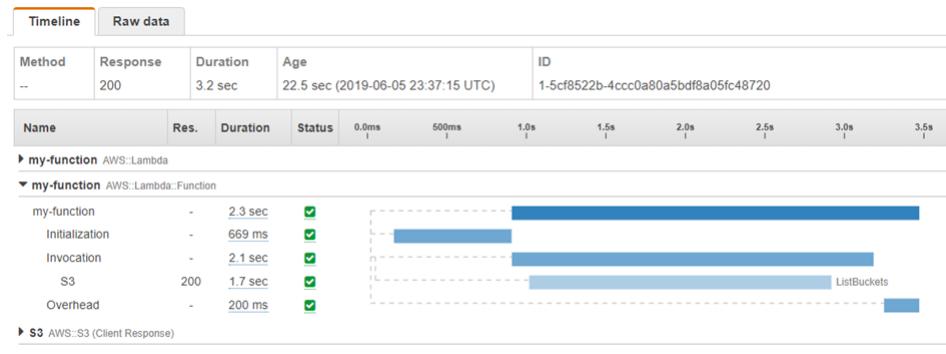
```
import boto3
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch_all

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

```
client = boto3.client('lambda')
client.get_account_settings()

def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES\r' + jsonpickle.encode(dict(**os.environ)))
    ...
```

O exemplo a seguir mostra um rastreamento com 2 segmentos. Ambos são chamados my-function, mas um é do tipo AWS::Lambda e o outro é AWS::Lambda::Function. O segmento de função é expandido para mostrar seus subsegmentos.



O primeiro segmento representa a solicitação de invocação processada pelo serviço do Lambda. O segundo segmento registra o trabalho realizado pela sua função. O segmento de função tem 3 subsegmentos.

- Inicialização: representa o tempo gasto carregando a função e executando o [código de inicialização \(p. 30\)](#). Esse subsegmento só aparece para o primeiro evento processado por cada instância da função.
- Invocação: representa o trabalho realizado pelo código do handler. Ao instrumentar o código, é possível estender esse subsegmento com subsegmentos adicionais.
- Sobrecarga: representa o trabalho realizado pelo tempo de execução do Lambda como preparação para lidar com o próximo evento.

Você também pode instrumentar clientes HTTP, registrar consultas SQL e criar subsegmentos personalizados com anotações e metadados. Para obter mais informações, consulte [The X-Ray SDK for Python](#) no Guia do desenvolvedor do AWS X-Ray.

Seções

- [Habilitar o rastreamento ativo com a API do Lambda \(p. 564\)](#)
- [Habilitar o rastreamento ativo com o AWS CloudFormation \(p. 565\)](#)
- [Armazenar dependências de tempo de execução em uma camada \(p. 565\)](#)

Habilitar o rastreamento ativo com a API do Lambda

Para gerenciar a configuração de rastreamento com a AWS CLI ou com o AWS SDK, use as seguintes operações de API:

- [UpdateFunctionConfiguration \(p. 974\)](#)
- [GetFunctionConfiguration \(p. 851\)](#)
- [CreateFunction \(p. 796\)](#)

O exemplo de comando da AWS CLI a seguir habilita o rastreamento ativo em uma função chamada my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

O modo de rastreamento faz parte da configuração específica da versão que é bloqueada quando você publica uma versão da função. Não é possível alterar o modo de rastreamento em uma versão publicada.

Habilitar o rastreamento ativo com o AWS CloudFormation

Para habilitar o rastreamento ativo em um recurso `AWS::Lambda::Function` em um modelo do AWS CloudFormation, use a propriedade `TracingConfig`.

Example [function-inline.yml](#): configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
      ...
    ...
```

Para um recurso do AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, use a propriedade `Tracing`.

Example [template.yml](#): configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Serverless::Function
    Properties:
      Tracing: Active
    ...
  ...
```

Armazenar dependências de tempo de execução em uma camada

Se você usar o X-Ray SDK para instrumentar os clientes do AWS SDK com seu código de função, seu pacote de implantação poderá se tornar bastante grande. Para evitar o upload de dependências de tempo de execução sempre que você atualizar seu código de funções, empacote-as em uma [camada do Lambda](#) (p. 85).

O exemplo a seguir mostra um recurso `AWS::Serverless::LayerVersion` que armazena o X-Ray SDK for Python.

Example [template.yml](#): camada de dependências

```
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:  
    - python3.8
```

Com essa configuração, você só atualizará a camada de biblioteca se alterar as dependências de tempo de execução. O pacote de implantação de função contém apenas o código. Ao atualizar o código de função, o tempo de upload será muito mais rápido do que se você incluir dependências no pacote de implantação.

A criação de uma camada de dependências requer alterações de compilação para gerar o arquivo da camada antes da implantação. Para obter um exemplo funcional, consulte o aplicativo de exemplo [blank-python](#).

Construir funções do Lambda com Ruby

Você pode executar o código Ruby no AWS Lambda. Lambda fornece [Tempos de execução do \(p. 214\)](#) Ruby que executa o seu código para processar eventos. Seu código é executado em um ambiente que inclui o AWS SDK for Ruby, com as credenciais de uma função do AWS Identity and Access Management (IAM) que você gerencia.

O Lambda oferece suporte aos seguintes tempos de execução do Ruby:

Tempos de execução do Ruby

Nome	Identifier	SDK for Ruby	Sistema operacional	
Ruby 2.7	<code>ruby2.7</code>	3.0.1	Amazon Linux 2	
Ruby 2.5	<code>ruby2.5</code>	3.0.1	Amazon Linux	

Note

Para obter informações sobre o fim do suporte sobre o Ruby 2.5, consulte [the section called “Política de suporte ao tempo de execução” \(p. 217\)](#).

Funções do Lambda usam um [Função de execução do \(p. 57\)](#) para obter permissão para gravar logs no Amazon CloudWatch Logs e para acessar outros serviços e recursos. Se você ainda não tiver uma função de execução para o desenvolvimento de funções, crie uma.

Para criar uma função de execução

1. Abra a [página Roles](#) (Funções) no console do IAM.
2. Selecione Create role.
3. Crie uma função com as seguintes propriedades.
 - Entidade confiável–Lambda.
 - Permissions (Permissões): AWSLambdaBasicExecutionRole.
 - Nome da função – **lambda-role**.

A política AWSLambdaBasicExecutionRole tem as permissões necessárias para a função gravar logs no CloudWatch Logs.

Você pode adicionar permissões à função posteriormente ou trocá-la para uma função diferente específica de uma única função.

Para criar uma função do Ruby

1. Abra o [console do lambda](#).
2. Escolha Create function (Criar função).
3. Configure as definições a seguir:

- Nome – **my-function**.
 - Runtime (Tempo de execução): Ruby 2.7.
 - Role (Função): Choose an existing role (Escolher uma função existente).
 - Existing role (Função existente – **lambda-role**).
4. Escolha Create function (Criar função).
 5. Para configurar um evento de teste, escolha Test (Testar).
 6. Em Event name (Nome do evento), insira **test**.
 7. Selecione Save changes.
 8. Escolha Test (Testar) para invocar a função.

O console cria uma função do Lambda com um único arquivo de origem chamado `lambda_function.rb`. Você pode editar esse arquivo e adicionar mais arquivos no [editor de códigos \(p. 40\)](#) integrado. Para salvar suas alterações, selecione Save (Salvar). Em seguida, para executar seu código, escolha Teste.

Note

O console do Lambda usa o AWS Cloud9 para fornecer um ambiente de desenvolvimento integrado no navegador. Você também pode usar o AWS Cloud9 para desenvolver funções do Lambda em seu próprio ambiente. Para obter mais informações, consulte [Working with Lambda Functions](#) no guia do usuário do AWS Cloud9.

O arquivo `lambda_function.rb` exporta uma função chamada `lambda_handler` que utiliza um objeto de evento e um objeto de contexto. Esta é a [função de manipulador \(p. 570\)](#) que o Lambda chama quando a função é invocada. O tempo de execução da função do Ruby obtém eventos de invocação do Lambda e os transmite ao handler. Na configuração da função, o valor do handler é `lambda_function.lambda_handler`.

Quando você salva seu código de função, o console do Lambda cria um pacote de implantação de arquivos.zip. Ao desenvolver seu código de função fora do console (usando um SDE), você precisa [Criar um pacote de implantação \(p. 571\)](#) para carregar seu código para a função do Lambda.

Note

Para começar com o desenvolvimento de aplicativos no ambiente local, implante um dos aplicativos de exemplo disponíveis no repositório do GitHub deste guia.

Aplicações de exemplo do Lambda em Ruby

- [blank-ruby](#): uma função do Ruby que mostra o uso de registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK.
- [Ruby Code Samples for AWS Lambda](#): exemplos de código escritos em Ruby que demonstram como interagir com o AWS Lambda.

O tempo de execução da função transmite um objeto de contexto para o handler, além do evento de invocação. O [objeto de contexto \(p. 577\)](#) contém informações adicionais sobre a invocação, a função e o ambiente de execução. Outras informações estão disponíveis de variáveis de ambiente.

Sua função do Lambda é fornecida com um grupo de logs do CloudWatch Logs. O tempo de execução da função envia detalhes sobre cada invocação para o CloudWatch Logs. Ele retransmite qualquer [logs que sua função produz \(p. 578\)](#) durante a invocação. Se a função [retornar um erro \(p. 583\)](#), o Lambda formatará o erro e o retornará para o chamador.

Tópicos

- [AWS Lambda Manipulador de função do em Ruby \(p. 570\)](#)

- [Implantar funções do Lambda em Ruby com arquivos .zip \(p. 571\)](#)
- [Implantar funções do Lambda em Ruby com imagens de contêiner \(p. 574\)](#)
- [AWS LambdaObjeto de contexto em Ruby \(p. 577\)](#)
- [AWS LambdaRegistro em log da função do em Ruby \(p. 578\)](#)
- [AWS LambdaErros da função do em Ruby \(p. 583\)](#)
- [Instrumentar o código Ruby no AWS Lambda \(p. 587\)](#)

AWS LambdaManipulador de função do em Ruby

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. Quando o manipulador é encerrado ou retorna uma resposta, ele se torna disponível para tratar de outro evento.

No exemplo a seguir, o arquivo `function.rb` define um método de handler chamado `handler`. A função do handler usa dois objetos como entrada e retorna um documento JSON.

Example `function.rb`

```
require 'json'

def handler(event:, context:)
    { event: JSON.generate(event), context: JSON.generate(context.inspect) }
end
```

Em sua configuração de função, a configuração `handler` informa o Lambda onde encontrar o manipulador. Para o exemplo anterior, o valor correto para essa configuração é `function.handler`. Ele inclui dois nomes separados por um ponto: o nome do arquivo e o nome do método do handler.

Você também pode definir o método de handler em uma classe. O exemplo a seguir define um método de handler chamado `process` em uma classe denominada `Handler` em um módulo chamado `LambdaFunctions`.

Example `source.rb`

```
module LambdaFunctions
  class Handler
    def self.process(event:,context:)
      "Hello!"
    end
  end
end
```

Nesse caso, a configuração do handler é `source.LambdaFunctions::Handler.process`.

Os dois objetos que o handler aceita são os eventos de invocação e contexto. O evento é um objeto Ruby que contém a carga que é fornecida pelo chamador. Se a carga útil for um documento JSON, o objeto de evento é um hash do Ruby. Caso contrário, é uma string. O [objeto de contexto \(p. 577\)](#) tem métodos e propriedades que fornecem informações sobre a invocação, a função e o ambiente de execução.

O manipulador da função é executado sempre que sua função do Lambda é invocada. O código estático fora do handler é executado uma vez por instância da função. Se o handler usar recursos como clientes SDK e conexões de banco de dados, você poderá criá-los fora do método de handler para reutilizá-los para várias invocações.

Cada instância de sua função pode processar vários eventos de invocação, mas ela processa apenas um evento por vez. O número de instâncias que processam um evento em um momento é a simultaneidade da sua função. Para obter mais informações sobre o ambiente de execução do Lambda, consulte [AWS LambdaAmbiente de execução do \(p. 219\)](#).

Implantar funções do Lambda em Ruby com arquivos .zip

O código da função do AWS Lambda consiste em scripts ou programas compilados e as dependências deles. Você usa um pacote de implantação para implantar seu código de função no Lambda. O Lambda é compatível com dois tipos de pacotes de implantação: imagens de contêiner e arquivos .zip.

Para criar um pacote de implantação compactado em um arquivo .zip, você pode usar um utilitário de compactação em arquivo .zip integrado ou qualquer outro utilitário semelhante (como o [7zip](#)) para sua ferramenta de linha de comando. Lembre-se dos seguintes requisitos para usar um arquivo .zip como seu pacote de implantação:

- O arquivo .zip contém o código da sua função e quaisquer dependências usadas para executá-lo (se aplicável) no Lambda. Se sua função depender apenas de bibliotecas padrão ou de bibliotecas do SDK do AWS, você não precisará inclui-las em seu arquivo .zip. Essas bibliotecas são incluídas nos ambientes de [tempo de execução do Lambda](#) (p. 214) suportados.
- Se o arquivo.zip for maior que 50 MB, recomendamos enviá-lo para sua função a partir de um bucket do Amazon Simple Storage Service (Amazon S3).
- Se o pacote de implantação contiver bibliotecas nativas, você poderá criar o pacote de implantação com o AWS Serverless Application Model (AWS SAM). É possível usar o comando AWS SAM da CLI do `sam build` com `--use-container` para criar seu pacote de implantação. Essa opção cria um pacote de implantação dentro de uma imagem do Docker compatível com o ambiente de execução do Lambda.

Para obter mais informações, consulte [sam build](#) no Guia do desenvolvedor do AWS Serverless Application Model.

- O Lambda usa permissões de arquivo POSIX, então pode ser necessário [definir permissões para a pasta do pacote de implantação](#) antes de criar o arquivo .zip.

Seções

- [Prerequisites \(p. 571\)](#)
- [Ferramentas e bibliotecas \(p. 571\)](#)
- [Atualizar uma função sem dependências \(p. 572\)](#)
- [Atualizar uma função com dependências adicionais \(p. 572\)](#)

Prerequisites

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Ferramentas e bibliotecas

O Lambda fornece as seguintes ferramentas e bibliotecas para o tempo de execução do Ruby:

Ferramentas e bibliotecas para Ruby

- [AWS SDK for Ruby](#): o AWS SDK oficial para a linguagem de programação Ruby.

Atualizar uma função sem dependências

Para atualizar uma função usando a API do Lambda, use a função [UpdateFunctionCode \(p. 965\)](#) operação. Crie um arquivo que contenha o código de função e faça upload dele usando a AWS Command Line Interface (AWS CLI).

Como atualizar uma função do Ruby sem dependências

1. Crie um arquivo .zip.

```
zip function.zip function.rb
```

2. Use o comando update-function-code para fazer upload do pacote.

```
aws lambda update-function-code --function-name my-function --zip-file fileb://
function.zip
```

Você deve ver a saída a seguir:

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
  "Runtime": "ruby2.5",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Handler": "function.handler",
  "CodeSha256": "Qf0hMc1I2di6YFMi9aXm3JtGTmcDbjniEuiYonYptAk=",
  "Version": "$LATEST",
  "TracingConfig": {
    "Mode": "Active"
  },
  "RevisionId": "983ed1e3-ca8e-434b-8dc1-7d72ebadd83d",
  ...
}
```

Atualizar uma função com dependências adicionais

Se a sua função depender de bibliotecas diferentes de AWS SDK for Ruby, instale-as em um diretório local com [Bundler](#) e as inclua em seu pacote de implantação.

Como atualizar uma função do Ruby com dependências

1. Instale bibliotecas no diretório do fornecedor com o comando bundle.

```
bundle config set --local path 'vendor/bundle' \
bundle install
```

Você deve ver a saída a seguir:

```
Fetching gem metadata from https://rubygems.org/.....
Resolving dependencies...
Fetching aws-eventstream 1.0.1
Installing aws-eventstream 1.0.1
...
```

Isso instala as gems no diretório do projeto em vez de no local do sistema e define vendor/bundle como o caminho padrão para futuras instalações. Para instalar gems posteriores globalmente, use bundle config set --local system 'true'.

2. Crie um arquivo .zip.

```
zip -r function.zip function.rb vendor
```

Você deve ver a saída a seguir:

```
adding: function.rb (deflated 37%)
adding: vendor/ (stored 0%)
adding: vendor/bundle/ (stored 0%)
adding: vendor/bundle/ruby/ (stored 0%)
adding: vendor/bundle/ruby/2.7.0/ (stored 0%)
adding: vendor/bundle/ruby/2.7.0/build_info/ (stored 0%)
adding: vendor/bundle/ruby/2.7.0/cache/ (stored 0%)
adding: vendor/bundle/ruby/2.7.0/cache/aws-eventstream-1.0.1.gem (deflated 36%)
...
```

3. Atualize o código da função.

```
aws lambda update-function-code --function-name my-function --zip-file fileb://
function.zip
```

Você deve ver a saída a seguir:

```
{
  "FunctionName": "my-function",
  "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
  "Runtime": "ruby2.5",
  "Role": "arn:aws:iam::123456789012:role/lambda-role",
  "Handler": "function.handler",
  "CodeSize": 300,
  "CodeSha256": "Qf0hMc1I2di6YFMi9aXm3JtGTmcDbjniEuiYonYptAk=",
  "Version": "$LATEST",
  "RevisionId": "983ed1e3-ca8e-434b-8dc1-7d72ebadd83d",
  ...
}
```

Implantar funções do Lambda em Ruby com imagens de contêiner

Você pode implantar seu código de função do Lambda como uma [imagem de contêiner \(p. 267\)](#). A AWS oferece os seguintes recursos para ajudar você a criar uma imagem de contêiner para sua função Ruby:

- [AWSImagens base para Lambda](#)

Essas imagens base são pré-carregadas com um tempo de execução de linguagem e outros componentes necessários para executar a imagem no Lambda. AWS fornece um Dockerfile para cada uma das imagens de base para ajudar a criar sua imagem de contêiner.

- Clientes de interface de tempo de execução de

Se você usar uma imagem de base de comunidade ou de empresa privada, adicione um cliente de interface de tempo de execução à imagem base para torná-lo compatível com o Lambda.

O fluxo de trabalho de uma função definida como uma imagem de contêiner inclui as seguintes etapas:

1. Crie sua imagem de contêiner usando os recursos listados neste tópico.
2. Faça upload da imagem em seu registro do contêiner do Amazon ECR. Consulte os passos 7 a 9 no [Criar imagem \(p. 268\)](#).
3. [Crie \(p. 91\)](#) a função do Lambda ou [atualize o código da função \(p. 92\)](#) para implantar a imagem em uma função existente.

Tópicos

- [AWSImagens de base da para Ruby \(p. 574\)](#)
- [Usar uma imagem base Ruby \(p. 574\)](#)
- [Clientes de interface de tempo de execução do Ruby \(p. 575\)](#)
- [Criar uma imagem do Ruby a partir de uma imagem básica da AWS \(p. 575\)](#)
- [Implantar a imagem do contêiner \(p. 576\)](#)

AWSImagens de base da para Ruby

AWSA oferece as seguintes imagens de base para Ruby:

Tags	Tempo de execução	Sistema operacional	Dockerfile
2, 2.7	Ruby 2.7	Amazon Linux 2	Dockerfile para Ruby 2.7 no GitHub
2.5	Ruby 2.5	Amazon Linux 2018.03	Dockerfile para Ruby 2.5 no GitHub

Repositório do Docker Hub: [amazon/aws-lambda-ruby](#)

Repositório do Amazon ECR: [gallery.ecr.aws/lambda/ruby](#)

Usar uma imagem base Ruby

Para obter instruções sobre como usar uma imagem base Ruby, escolha a guia usage (uso) em [Imagens base do AWS Lambda para Ruby](#) no repositório do Amazon ECR.

As instruções também estão disponíveis em [imagens base do AWS Lambda para Ruby](#) no repositório do Docker Hub.

Cientes de interface de tempo de execução do Ruby

Instale o cliente de interface de tempo de execução para Ruby usando o gerenciador de pacotes RubyGems.org:

```
gem install aws_lambda_ric
```

Para obter detalhes do pacote, consulte [Lambda RIC](#) em [RubyGems.org](#).

Você também pode fazer download do [cliente de interface de tempo de execução do Ruby](#) no GitHub.

Criar uma imagem do Ruby a partir de uma imagem básica da AWS

Quando você cria uma imagem de contêiner para Ruby usando uma imagem básica da AWS, só precisa copiar a aplicação ruby para o contêiner e instalar quaisquer dependências.

Para criar e implantar uma função do Ruby com a imagem básica da **ruby:2.7**:

1. Em sua máquina local, crie um diretório de projeto para sua nova função.
2. No diretório do projeto, adicione um arquivo chamado `app.rb` contendo seu código de função. O exemplo a seguir mostra um manipulador do Ruby simples.

```
module LambdaFunction
  class Handler
    def self.process(event:, context:)
      "Hello from Ruby 2.7 container image!"
    end
  end
end
```

3. Use um editor de texto para criar um Dockerfile no diretório do projeto. O exemplo a seguir mostra o Dockerfile para o manipulador que você criou na etapa anterior. Instale qualquer dependência no diretório `#{LAMBDA_TASK_ROOT}` junto do manipulador de funções para garantir que o tempo de execução do Lambda possa localizá-la quando a função for invocada.

```
FROM public.ecr.aws/lambda/ruby:2.7

# Copy function code
COPY app.rb ${LAMBDA_TASK_ROOT}

# Copy dependency management file
COPY Gemfile ${LAMBDA_TASK_ROOT}

# Install dependencies under LAMBDA_TASK_ROOT
ENV GEM_HOME=${LAMBDA_TASK_ROOT}
RUN bundle install

# Set the CMD to your handler (could also be done as a parameter override outside of
# the Dockerfile)
CMD [ "app.LambdaFunction::Handler.process" ]
```

4. Para criar a imagem do contêiner, siga as etapas de 4 a 7 em [Criar uma imagem a partir de uma imagem básica da AWS para o Lambda \(p. 268\)](#).

Implantar a imagem do contêiner

Para uma nova função, você implanta a imagem do Ruby ao [criar a função \(p. 91\)](#). Para uma função existente, se você reconstruir a imagem do contêiner, precisará reimplantar a imagem ao [atualizar o código da função \(p. 92\)](#).

AWS LambdaObjeto de contexto em Ruby

Quando o Lambda executa a função, ele transmite um objeto de contexto para o [handler \(p. 570\)](#). Esse objeto fornece métodos e propriedades que fornecem informações sobre a invocação, a função e o ambiente de execução.

Métodos de contexto

- `get_remaining_timeInMillis`: retorna o número de milissegundos restantes antes do tempo limite da execução.

Propriedades de contexto

- `function_name`: o nome da função do Lambda.
- `function_version`: a [versão \(p. 106\)](#) da função.
- `invoked_function_arn`: o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um alias ou número de versão.
- `memory_limit_in_mb`: a quantidade de memória alocada para a função.
- `aws_request_id`: o identificador da solicitação de invocação.
- `log_group_name`: o grupo de logs da função.
- `log_stream_name`: a transmissão de log para a instância da função.
- `deadline_ms`: a data em que a função expira em milissegundos de tempo do Unix.
- `identity`: (aplicativos móveis) informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
- `client_context`: (aplicativos móveis) contexto do cliente fornecido ao Lambda pela aplicação cliente.

AWS Lambda Registro em log da função do em Ruby

O AWS Lambda monitora automaticamente as funções do Lambda em seu nome e envia métricas da função para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente do tempo de execução do Lambda envia detalhes sobre cada invocação à transmissão de logs e transmite os logs e outras saídas do código de sua função.

Esta página descreve como produzir a saída de logs usando o código de sua função do Lambda ou acessar os logs usando a AWS Command Line Interface, o console do Lambda ou o console do CloudWatch.

Seções

- [Criar uma função que retorna logs \(p. 578\)](#)
- [Usar o console do Lambda \(p. 579\)](#)
- [Usando o console do CloudWatch \(p. 579\)](#)
- [Usar a AWS Command Line Interface \(AWS CLI\) \(p. 580\)](#)
- [Excluir logs \(p. 582\)](#)

Criar uma função que retorna logs

Para gerar os logs do seu código de função, você pode usar instruções `puts` ou qualquer biblioteca de logs que grava no `stdout` ou no `stderr`. O exemplo a seguir registra em log os valores das variáveis de ambiente e o objeto do 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
```

Para logs mais detalhados, use [a biblioteca de logger](#).

```
# 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
```

A saída de `logger` inclui o time stamp, o ID do processo, o nível do log e o ID da solicitação.

```
I, [2019-10-26T10:04:01.689856 #8] INFO 6573a3a0-2fb1-4e78-a582-2c769282e0bd -- : ## EVENT
I, [2019-10-26T10:04:01.689874 #8] INFO 6573a3a0-2fb1-4e78-a582-2c769282e0bd -- :
{"key1"=>"value1", "key2"=>"value2", "key3"=>"value3"}
```

Example formato do log

```
START RequestId: 50aba555-99c8-4b21-8358-644ee996a05f Version: $LATEST
## ENVIRONMENT VARIABLES
AWS_LAMBDA_FUNCTION_VERSION
$LATEST
AWS_LAMBDA_LOG_GROUP_NAME
/aws/lambda/my-function
AWS_LAMBDA_LOG_STREAM_NAME
2020/01/31/[ $LATEST ]3f34xmpl069f4018b4a773bcfe8ed3f9
AWS_EXECUTION_ENV
AWS_Lambda_ruby2.5
...
## EVENT
key
value
END RequestId: 50aba555-xmpl-4b21-8358-644ee996a05f
REPORT RequestId: 50aba555-xmpl-4b21-8358-644ee996a05f Duration: 12.96 ms Billed Duration:
13 ms Memory Size: 128 MB Max Memory Used: 48 MB Init Duration: 117.86 ms
XRAY TraceId: 1-5e34a246-2a04xmpl0fa44eb60ea08c5f SegmentId: 454xmpl46ca1c7d3 Sampled: true
```

O tempo de execução do Ruby registra em log as linhas START, END e REPORT para cada invocação. A linha do relatório fornece os seguintes detalhes.

Registro em log de relatório

- RequestId: o ID de solicitação exclusivo para a invocação.
- Duração: a quantidade de tempo que o método de manipulador da função gastou processando o evento.
- Duração faturada: a quantia de tempo faturada para a invocação.
- Tamanho da memória: a quantidade de memória alocada para a função.
- Memória máxima utilizada: a quantidade de memória utilizada pela função.
- Duração inicial: para a primeira solicitação atendida, a quantidade de tempo que o tempo de execução levou para carregar a função e executar o código fora do método do manipulador.
- XRAY Traceld: para solicitações rastreadas, o [ID de rastreamento do AWS X-Ray \(p. 477\)](#).
- SegmentId: para solicitações rastreadas, o ID do segmento do X-Ray.
- Amostragem: para solicitações rastreadas, o resultado da amostragem.

Usar o console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda. Para obter mais informações, consulte [Acessar o Amazon CloudWatch Logs para o AWS Lambda \(p. 720\)](#).

Usando o console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a [página Log groups](#) (Grupos de log).

2. Escolha o grupo de logs de sua função (/aws/lambda/**nome-de-sua-função**).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma [instância da sua função \(p. 219\)](#). Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos instrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

Para usar uma aplicação de exemplo que correlaciona os logs e os rastreamentos com o X-Ray, consulte [Aplicativo de exemplo de processador de erros para o AWS Lambda \(p. 500\)](#).

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Você pode usar a [AWS CLI](#) para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Você deve ver a saída a seguir:

```
{  
    "StatusCode": 200,  
    "LogResult":  
        "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",  
    "ExecutedVersion": "$LATEST"  
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \  
--query 'LogResult' --output text | base64 -d
```

Você deve ver a saída a seguir:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e [Ubuntu no Windows](#). Os usuários do macOS precisam usar `base64 -D`.

Example get-logs.sh script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa `sed` para remover as aspas do arquivo de saída e fica inativo por 15 segundos para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como `get-logs.sh`.

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

```
#!/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 $(cat
out) --limit 5
```

Example macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod +x get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

Você deve ver a saída a seguir:

```
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\+INFO\+ENVIRONMENT VARIABLES\r{\r\t\"AWS_LAMBDA_FUNCTION_VERSION\": \"$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"
}
```

Excluir logs

Os grupos de logs não são excluídos automaticamente quando você exclui uma função. Para evitar armazenar logs indefinidamente, exclua o grupo de logs ou[Configurar um período de retenção](#)após o qual os logs são excluídos automaticamente.

AWS Lambda Erros da função do em Ruby

Quando o código gera um erro, o Lambda gera uma representação JSON do erro. Esse documento de erro aparece no log de invocação e, para invocações síncronas, na saída.

Esta página descreve como exibir erros de invocação de função do Lambda para o tempo de execução do Ruby usando o console do Lambda e a AWS CLI.

Seções

- [Syntax \(p. 583\)](#)
- [Como funcionam \(p. 583\)](#)
- [Usar o console do Lambda \(p. 584\)](#)
- [Usar a AWS Command Line Interface \(AWS CLI\) \(p. 584\)](#)
- [Tratamento de erros em outros serviços da AWS \(p. 585\)](#)
- [Aplicativos de exemplo \(p. 586\)](#)
- [Próximas etapas \(p. 586\)](#)

Syntax

Example function.rb

```
def handler(event:, context:)
    puts "Processing event..."
    [1, 2, 3].first("two")
    "Success"
end
```

Esse código resulta em um erro de tipo. O Lambda detecta o erro e gera um documento JSON com os campos para a mensagem de erro, o tipo e o rastreamento da pilha.

```
{
  "errorMessage": "no implicit conversion of String into Integer",
  "errorType": "Function<TypeError>",
  "stackTrace": [
    "/var/task/function.rb:3:in `first'",
    "/var/task/function.rb:3:in `handler'"
  ]
}
```

Como funcionam

Ao invocar uma função do Lambda, o Lambda recebe a solicitação de invocação e valida as permissões de sua função de execução, verifica se o documento do evento é um documento JSON válido e verifica valores de parâmetros.

Se a solicitação for validada, o Lambda a envia para uma instância da função. O ambiente de [tempo de execução do Lambda \(p. 214\)](#) converte o documento do evento em um objeto e o transmite ao handler da função.

Se o Lambda encontra um erro, ele retorna um tipo de exceção, uma mensagem e o código HTTP do status que indica a causa do erro. O cliente ou o serviço que invocou a função do Lambda pode processar

o erro de maneira programática ou transmiti-lo a um usuário final. O comportamento correto de tratamento de erros depende do tipo de aplicativo, do público e da origem do erro.

A lista a seguir descreve o intervalo de códigos de status que você pode receber do Lambda.

2xx

Um erro da série 2xx com um cabeçalho `X-Amz-Function-Error` na resposta indica um erro de tempo de execução ou de função do Lambda. Um código de status da série 2xx indica que o Lambda aceitou a solicitação, mas em vez de um código de erro, o Lambda indica o erro incluindo o cabeçalho `X-Amz-Function-Error` na resposta.

4xx

Um erro da série 4xx indica um erro que o cliente ou serviço que fez a invocação pode corrigir modificando a solicitação, solicitando permissão ou tentando a solicitação novamente. Os erros da série 4xx diferentes de 429 geralmente indicam um erro na solicitação.

5xx

Um erro da série 5xx indica um problema com o Lambda ou com a configuração/recursos da função. Os erros da série 5xx podem indicar uma condição temporária que pode ser resolvida sem nenhuma ação do usuário. O cliente ou serviço que fez a invocação não podem solucionar esses problemas, mas o proprietário de uma função do Lambda pode ser capaz de corrigi-los.

Para obter uma lista completa de erros de invocação, consulte [Erros de InvokeFunction \(p. 877\)](#).

Usar o console do Lambda

Você pode invocar sua função no console do Lambda configurando um evento de teste e visualizando a saída. A saída é capturada pelos logs de execução da função e, quando o [rastreamento ativo \(p. 477\)](#) está habilitado, pelo AWS X-Ray.

Para invocar uma função no console do Lambda

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Test (Testar).
4. Selecione New event (Novo evento) e escolha um Event template (Modelo de evento) na lista suspensa.
5. Insira um nome para o evento de teste.
6. Digite o JSON para o evento de teste.
7. Escolha Create event (Criar evento).
8. Escolha Invoke (Invocar).

O console do Lambda invoca sua função [de forma síncrona \(p. 158\)](#) e exibe o resultado. Para ver a resposta, os logs e outras informações, expanda a seção Details (Detalhes).

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)

- [AWS CLI – Configuração rápida com aws configure](#)

Quando você invoca uma função do Lambda na AWS CLI, a AWS CLI divide a resposta em dois documentos. A resposta da AWS CLI é exibida no prompt de comando. Se ocorreu um erro, a resposta contém um campo `FunctionError`. A resposta ou o erro de invocação retornado pela função é gravado no arquivo de saída. Por exemplo, o `output.json` ou o `output.txt`.

O exemplo de comando `invoke` a seguir demonstra como invocar uma função e escrever a resposta de invocação em um arquivo `output.txt`.

```
aws lambda invoke \
--function-name my-function \
--cli-binary-format raw-in-base64-out \
--payload '{"key1": "value1", "key2": "value2", "key3": "value3"}' output.txt
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

Você deve ver a resposta da AWS CLI em seu prompt de comando:

```
{  
    "StatusCode": 200,  
    "FunctionError": "Unhandled",  
    "ExecutedVersion": "$LATEST"  
}
```

Você deve ver a resposta de invocação de função no arquivo `output.txt`. Também é possível visualizar a saída no mesmo prompt de comando usando:

```
cat output.txt
```

Você deve ver a resposta de invocação no prompt de comando.

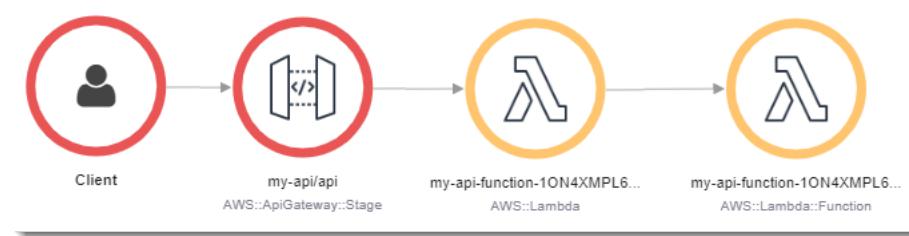
```
{"errorMessage": "no implicit conversion of String into  
Integer", "errorType": "Function<TypeError>", "stackTrace": ["/var/task/function.rb:3:in  
'first'", "/var/task/function.rb:3:in `handler'"]}
```

Tratamento de erros em outros serviços da AWS

Quando outro serviço da AWS invoca sua função, o serviço escolhe o tipo de invocação e o comportamento de repetição. Os serviços da AWS podem invocar sua função em um agendamento, em resposta a um evento de ciclo de vida em um recurso ou para atender a uma solicitação de um usuário. Alguns serviços invocam funções de forma assíncrona e permitem que o Lambda trate erros, enquanto outros fazem novas tentativas ou transmitem os erros de volta ao usuário.

Por exemplo, o API Gateway trata todos os erros de invocação e função como erros internos. Se a API do Lambda rejeitar a solicitação de invocação, o API Gateway retornará um código de erro 500. Se a função é executada, mas retorna um erro ou retorna uma resposta no formato errado, o API Gateway retorna um código de erro 502. Para personalizar a resposta de erro, é necessário detectar erros no código e formatar uma resposta no formato necessário.

Recomendamos usar AWS X-Ray para determinar a fonte de um erro e a respectiva causa. O X-Ray permite localizar qual componente encontrou um erro e ver detalhes sobre os erros. O exemplo a seguir mostra um erro de função que resultou em uma resposta 502 do API Gateway.



Para obter mais informações, consulte [Instrumentar o código Ruby no AWS Lambda \(p. 587\)](#).

Aplicativos de exemplo

O código de exemplo está disponível para o tempo de execução do Ruby.

Aplicações de exemplo do Lambda em Ruby

- [blank-ruby](#): uma função do Ruby que mostra o uso de registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, camadas, testes de unidade e do AWS SDK.
- [Ruby Code Samples for AWS Lambda](#): exemplos de código escritos em Ruby que demonstram como interagir com o AWS Lambda.

Próximas etapas

- Saiba como mostrar eventos de registro em log para sua função do Lambda na página [the section called “Registro em log” \(p. 578\)](#).

Instrumentar o código Ruby no AWS Lambda

O Lambda se integra ao AWS X-Ray para permitir que você rastreie, depure e otimize aplicativos do Lambda. É possível usar o X-Ray para rastrear uma solicitação à medida que ela atravessa recursos na aplicação, da API de frontend ao armazenamento e aos banco de dados no backend. Ao simplesmente adicionar a biblioteca do X-Ray SDK à configuração de compilação, é possível registrar erros e latência para qualquer chamada que a função faça para um serviço da AWS.

O X-Ray Mapa de serviço mostra o fluxo de solicitações através de seu aplicativo. O exemplo a seguir do aplicativo de exemplo [processador de erros \(p. 500\)](#) mostra um aplicativo com duas funções. A função principal processa eventos e, às vezes, retorna erros. A segunda função processa erros que aparecem no primeiro grupo de logs e usa o AWS SDK para chamar o X-Ray, o Amazon S3 e o Amazon CloudWatch Logs.



Para rastrear solicitações que não têm um cabeçalho de rastreamento, ative o rastreamento ativo na configuração da sua função.

Para ativar o rastreamento ativo

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Selecione **Configuração**, depois, escolha **Ferramentas de monitoramento**.
4. Selecione **Edit**.
5. Em X-Ray, habilite o **Active tracing (Rastreamento ativo)**.
6. Escolha **Save (Salvar)**.

Pricing

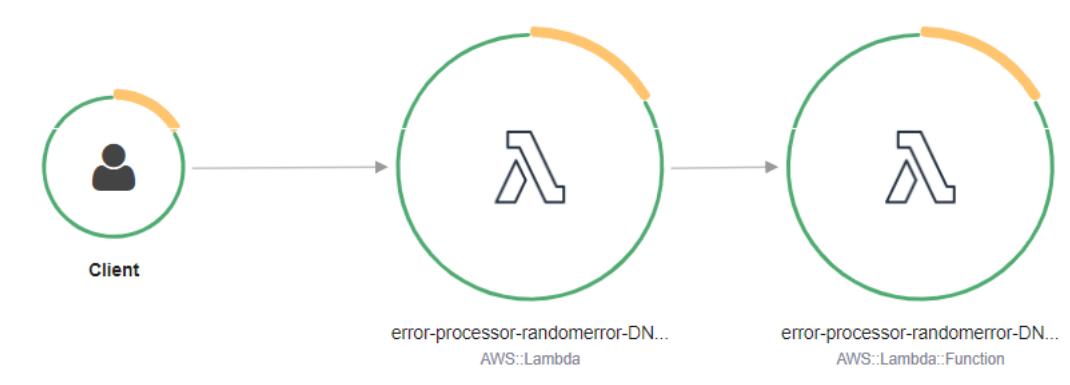
O X-Ray tem um nível gratuito vitalício. Além do limite do nível gratuito, o X-Ray cobra por armazenamento e recuperação de rastreamento. Para obter detalhes, consulte [Definição de preço do AWS X-Ray](#).

Sua função precisa de permissão para carregar dados de rastreamento no X-Ray. Quando você habilita o rastreamento ativo no console do Lambda, o Lambda adiciona as permissões necessárias à [função de](#)

[execução \(p. 57\)](#) da função. Caso contrário, adicione a política [AWSXRayDaemonWriteAccess](#) à função de execução.

O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento é eficiente, enquanto ainda fornece uma amostra representativa das solicitações que a sua aplicação serviu. A regra de amostragem padrão é uma solicitação por segundo e 5% de solicitações adicionais. Esta taxa de amostragem não pode ser configurada para funções do Lambda.

Quando o rastreamento ativo está habilitado, o Lambda registra um rastreamento para um subconjunto de invocações. Lambda registra doisSegmentos do, que cria dois nós no mapa de serviço. O primeiro representa o serviço Lambda que recebe a solicitação de invocação. O segundo nó é gravado pelo [tempo de execução \(p. 19\)](#)da função.



É possível instrumentar o código do manipulador para gravar metadados e rastrear chamadas downstream. Para registrar detalhes sobre chamadas feitas pelo manipulador para outros recursos e serviços, use o X-Ray SDK for Ruby. Para obter o SDK, adicione o pacote `aws-xray-sdk` às dependências do aplicativo.

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'
  
```

Para instrumentar clientes do AWS SDK, solicite o módulo `aws-xray-sdk/lambda` depois de criar um cliente no código de inicialização.

Example [blank-ruby/function/lambda_function.rb](#): rastrear um cliente do AWS SDK

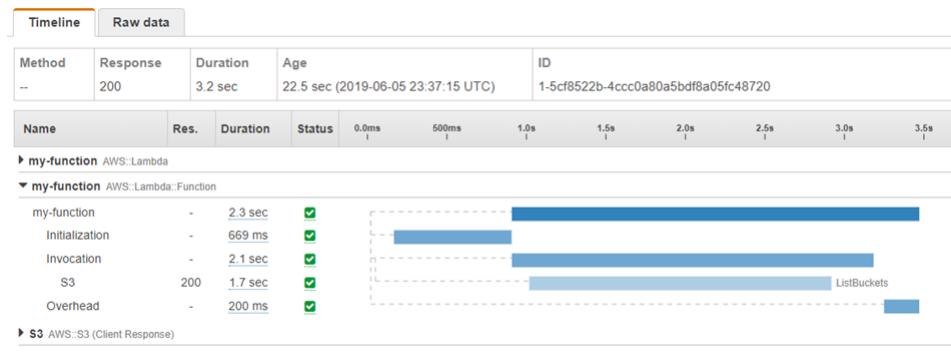
```

# lambda_function.rb
require 'logger'
require 'json'
require 'aws-sdk-lambda'
$client = Aws::Lambda::Client.new()
$client.get_account_settings()

require 'aws-xray-sdk/lambda'

def lambda_handler(event:, context:)
  logger = Logger.new($stdout)
  ...
  
```

O exemplo a seguir mostra um rastreamento com 2 segmentos. Ambos são chamados my-function, mas um é do tipo AWS::Lambda e o outro é AWS::Lambda::Function. O segmento de função é expandido para mostrar seus subsegmentos.



O primeiro segmento representa a solicitação de invocação processada pelo serviço do Lambda. O segundo segmento registra o trabalho realizado pela sua função. O segmento de função tem 3 subsegmentos.

- Inicialização: representa o tempo gasto carregando a função e executando o [código de inicialização \(p. 30\)](#). Esse subsegmento só aparece para o primeiro evento processado por cada instância da função.
- Invocação: representa o trabalho realizado pelo código do handler. Ao instrumentar o código, é possível estender esse subsegmento com subsegmentos adicionais.
- Sobrecarga: representa o trabalho realizado pelo tempo de execução do Lambda como preparação para lidar com o próximo evento.

Você também pode instrumentar clientes HTTP, registrar consultas SQL e criar subsegmentos personalizados com anotações e metadados. Para obter mais informações, consulte [The X-Ray SDK for Ruby](#) no Guia do desenvolvedor do AWS X-Ray.

Seções

- [Habilitar o rastreamento ativo com a API do Lambda \(p. 589\)](#)
- [Habilitar o rastreamento ativo com o AWS CloudFormation \(p. 590\)](#)
- [Armazenar dependências de tempo de execução em uma camada \(p. 590\)](#)

Habilitar o rastreamento ativo com a API do Lambda

Para gerenciar a configuração de rastreamento com a AWS CLI ou com o AWS SDK, use as seguintes operações de API:

- [UpdateFunctionConfiguration \(p. 974\)](#)
- [GetFunctionConfiguration \(p. 851\)](#)
- [CreateFunction \(p. 796\)](#)

O exemplo de comando da AWS CLI a seguir habilita o rastreamento ativo em uma função chamada my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

O modo de rastreamento faz parte da configuração específica da versão que é bloqueada quando você publica uma versão da função. Não é possível alterar o modo de rastreamento em uma versão publicada.

Habilitar o rastreamento ativo com o AWS CloudFormation

Para habilitar o rastreamento ativo em um recurso `AWS::Lambda::Function` em um modelo do AWS CloudFormation, use a propriedade `TracingConfig`.

Example [function-inline.yml](#): configuração de rastreamento

```
Resources:  
  function:  
    Type: AWS::Lambda::Function  
    Properties:  
      TracingConfig:  
        Mode: Active  
      ...
```

Para um recurso do AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, use a propriedade `Tracing`.

Example [template.yml](#): configuração de rastreamento

```
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      Tracing: Active  
    ...
```

Armazenar dependências de tempo de execução em uma camada

Se você usar o X-Ray SDK para instrumentar os clientes do AWS SDK com seu código de função, seu pacote de implantação poderá se tornar bastante grande. Para evitar o upload de dependências de tempo de execução sempre que você atualizar seu código de funções, empacote-as em uma [camada do Lambda \(p. 85\)](#).

O exemplo a seguir mostra um recurso `AWS::Serverless::LayerVersion` que armazena o X-Ray SDK for Ruby.

Example [template.yml](#): camada de dependências

```
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
```

Com essa configuração, você só atualizará a camada de biblioteca se alterar as dependências de tempo de execução. O pacote de implantação de função contém apenas o código. Ao atualizar o código de função, o tempo de upload será muito mais rápido do que se você incluir dependências no pacote de implantação.

A criação de uma camada de dependências requer alterações de compilação para gerar o arquivo da camada antes da implantação. Para obter um exemplo funcional, consulte o aplicativo de exemplo [blank-ruby](#).

Construir funções do Lambda com Java

Você pode executar o código Java no AWS Lambda. Lambda fornece [Tempos de execução do \(p. 214\)](#) para Java que executa o seu código para processar eventos. O código é executado em um ambiente do Amazon Linux que inclui credenciais da AWS de uma função do AWS Identity and Access Management (IAM) que você gerencia.

O Lambda oferece suporte aos seguintes tempos de execução Java.

Tempos de execução do Java

Nome	Identifier	JDK	Sistema operacional
Java 11	java11	amazon-corretto-11	Amazon Linux 2
Java 8	java8.al2	amazon-corretto-8	Amazon Linux 2
Java 8	java8	java-1.8.0-openjdk	Amazon Linux

O Lambda fornece as seguintes bibliotecas para funções em Java:

- [com.amazonaws:aws-lambda-java-core](#) (obrigatória): define as interfaces do método do manipulador e o objeto de contexto que o tempo de execução transmite ao manipulador. Se você definir seus próprios tipos de entrada, esta será a única biblioteca necessária.
- [com.amazonaws:aws-lambda-java-events](#): tipos de entrada para eventos de serviços que invocam funções do Lambda.
- [com.amazonaws:aws-lambda-java-log4j2](#): uma biblioteca appender do Apache Log4j 2 que você pode usar para adicionar o ID de solicitação da invocação atual aos [logs de função \(p. 611\)](#).
- [AWS SDK for Java 2.0](#): o AWS SDK oficial para a linguagem de programação Java.

Funções do Lambda usam um [Função de execução do \(p. 57\)](#) para obter permissão para gravar logs no Amazon CloudWatch Logs e para acessar outros serviços e recursos. Se você ainda não tiver uma função de execução para o desenvolvimento de funções, crie uma.

Para criar uma função de execução

1. Abra a [página Roles](#) (Funções) no console do IAM.
2. Selecione Create role.
3. Crie uma função com as seguintes propriedades.
 - Entidade confiável-Lambda.
 - Permissions (Permissões): AWSLambdaBasicExecutionRole.
 - Nome da função – **lambda-role**.

A política AWSLambdaBasicExecutionRole tem as permissões necessárias para a função gravar logs no CloudWatch Logs.

Você pode adicionar permissões à função posteriormente ou trocá-la para uma função diferente específica de uma única função.

Para criar uma função do Java

1. Abra o [console do lambda](#).
2. Escolha Create function.
3. Configure as definições a seguir:
 - Nome – **my-function**.
 - Runtime (Tempo de execução): Java 11.
 - Role (Função): Choose an existing role (Escolher uma função existente).
 - Existing role (Função existente – **lambda-role**).
4. Escolha Create function.
5. Para configurar um evento de teste, escolha Test (Testar).
6. Em Event name (Nome do evento), insira **test**.
7. Selecione Save changes.
8. Escolha Test (Testar) para invocar a função.

O console cria uma função do Lambda com uma classe de manipulador chamada `Hello`. Como Java é uma linguagem compilada, não é possível visualizar nem editar o código-fonte no console do Lambda, mas você pode modificar a configuração dele, invocá-lo e configurar acionadores.

Note

Para começar com o desenvolvimento de aplicativos no ambiente local, implante um dos [aplicativos de exemplo \(p. 631\)](#) disponíveis no repositório do GitHub deste guia.

A classe `Hello` tem uma função chamada `handleRequest` que utiliza um objeto de evento e um objeto de contexto. Esta é a [função de manipulador \(p. 595\)](#) que o Lambda chama quando a função é invocada. O tempo de execução da função do Java recebe eventos de invocação do Lambda e os transmite ao handler. Na configuração da função, o valor do handler é `example.Hello::handleRequest`.

Para atualizar o código da função, crie um pacote de implantação, que é um arquivo `.zip` que contém o código da função. À medida que o desenvolvimento da função progredir, você desejará armazenar o código da função no controle de origem, adicionar bibliotecas e automatizar implantações. Comece [criando um pacote de implantação \(p. 599\)](#) e atualizando o seu código na linha de comando.

O tempo de execução da função transmite um objeto de contexto para o handler, além do evento de invocação. O [objeto de contexto \(p. 608\)](#) contém informações adicionais sobre a invocação, a função e o ambiente de execução. Outras informações estão disponíveis de variáveis de ambiente.

Sua função do Lambda é fornecida com um grupo de logs do CloudWatch Logs. O tempo de execução da função envia detalhes sobre cada invocação para o CloudWatch Logs. Ele retransmite qualquer [logs que sua função produz \(p. 611\)](#) durante a invocação. Se a função [retornar um erro \(p. 618\)](#), o Lambda formatará o erro e o retornará para o chamador.

Tópicos

- [AWS LambdaManipulador de função do em Java \(p. 595\)](#)
- [Implantar funções do Lambda em Java com arquivos .zip ou JAR \(p. 599\)](#)
- [Implantar funções do Lambda em Java com imagens de contêiner \(p. 606\)](#)
- [AWS LambdaObjeto de contexto do em Java \(p. 608\)](#)
- [AWS LambdaRegistro em log da função do em Java \(p. 611\)](#)
- [AWS LambdaErros da função do em Java \(p. 618\)](#)

- [Instrumentação do código Java no AWS Lambda \(p. 623\)](#)
- [Criar um pacote de implantação usando o Eclipse \(p. 628\)](#)
- [Aplicativos de exemplo Java para o AWS Lambda \(p. 631\)](#)

AWS LambdaManipulador de função do em Java

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. Quando o manipulador é encerrado ou retorna uma resposta, ele se torna disponível para tratar de outro evento.

No exemplo a seguir, uma classe chamada `Handler` define um método do manipulador chamado `handleRequest`. O método do manipulador usa um evento e um objeto de contexto como entrada e retorna uma string.

Example `Handler.java`

```
package example;
import com.amazonaws.services.lambda.runtime.Context
import com.amazonaws.services.lambda.runtime.RequestHandler
import com.amazonaws.services.lambda.runtime.LambdaLogger
...

// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String, String>, String>{
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    @Override
    public String handleRequest(Map<String, String> event, Context context)
    {
        LambdaLogger logger = context.getLogger();
        String response = new String("200 OK");
        // 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));
        logger.log("EVENT TYPE: " + event.getClass().toString());
        return response;
    }
}
```

O tempo de execução do Lambda (p. 214) recebe um evento como uma string no formato JSON e o converte em um objeto. Ele transmite o objeto de evento ao manipulador de função juntamente com um objeto de contexto que fornece detalhes sobre a invocação e a função. Você indica ao tempo de execução qual método invocar definindo o parâmetro do manipulador na configuração da sua função.

Formatos do manipulador

- `package.Class::method`: formato completo. Por exemplo: `example.Handler::handleRequest`.
- `package.Class`: formato abreviado para funções que implantam uma interface do manipulador (p. 597). Por exemplo: `example.Handler`.

É possível adicionar o código de inicialização (p. 30) fora do método do manipulador para reutilizar recursos em várias invocações. Quando o tempo de execução carrega seu manipulador, ele executa o código estático e o construtor de classe. Os recursos criados durante a inicialização permanecem na memória entre as invocações e podem ser reutilizados pelo manipulador milhares de vezes.

No exemplo a seguir, o logger, o serializador e o cliente do AWS SDK são criados quando a função atende seu primeiro evento. Eventos subsequentes atendidos pela mesma instância de função são muito mais rápidos porque esses recursos já existem.

Example `Handler.java`: código de inicialização

```
// Handler value: example.Handler
```

```
public class Handler implements RequestHandler<SQSEvent, String>{
    private static final Logger logger = LoggerFactory.getLogger(Handler.class);
    private static final Gson gson = new GsonBuilder().setPrettyPrinting().create();
    private static final LambdaAsyncClient lambdaClient = LambdaAsyncClient.create();
    ...
    @Override
    public String handleRequest(SQSEvent event, Context context)
    {
        String response = new String();
        // call Lambda API
        logger.info("Getting account settings");
        CompletableFuture<GetAccountSettingsResponse> accountSettings =
            lambdaClient.getAccountSettings(GetAccountSettingsRequest.builder().build());
        // log execution details
        logger.info("ENVIRONMENT VARIABLES: " + gson.toJson(System.getenv()));
        ...
    }
}
```

O repositório do GitHub para este guia fornece aplicativos de exemplo fáceis de implantar que demonstram uma variedade de tipos de manipuladores. Para obter detalhes, consulte o [final deste tópico \(p. 598\)](#).

Seções

- [Escolher tipos de entrada e saída \(p. 596\)](#)
- [Interfaces do manipulador \(p. 597\)](#)
- [Código de exemplo do manipulador \(p. 598\)](#)

Escolher tipos de entrada e saída

Especifique o tipo de objeto para o qual o evento é mapeado na assinatura do método do manipulador. No exemplo anterior, o tempo de execução do Java desserializa o evento em um tipo que implementa a interface `Map<String, String>`. Mapas de string a string funcionam para eventos simples como o seguinte:

Example `Event.json`: dados climáticos

```
{
    "temperatureK": 281,
    "windKmh": -3,
    "humidityPct": 0.55,
    "pressureHPa": 1020
}
```

No entanto, o valor de cada campo deve ser uma string ou um número. Se o evento incluir um campo que tenha um objeto como um valor, o tempo de execução não poderá desserializá-lo e retornará um erro.

Escolha um tipo de entrada que funcione com os dados de evento que sua função processa. É possível usar um tipo básico, um tipo genérico ou um tipo bem definido.

Tipos de entrada

- `Integer`, `Long`, `Double`, etc. – o evento é um número sem formatação adicional, por exemplo, `3.5`. O tempo de execução converte o valor em um objeto do tipo especificado.
- `String` – o evento é uma string JSON, incluindo aspas, por exemplo, `"My string."`. O tempo de execução converte o valor (sem aspas) em um objeto `String`.
- `Type`, `Map<String, Type>` etc. – o evento é um objeto JSON. O tempo de execução o desserializa em um objeto da interface ou tipo especificado.
- `List<Integer>, List<String>, List<Object>`, etc. – o evento é uma matriz JSON. O tempo de execução o desserializa em um objeto da interface ou tipo especificado.

- **InputStream**: o evento é qualquer tipo JSON. O tempo de execução transmite um fluxo de bytes do documento ao manipulador sem modificação. Dessaírialize a entrada e grave a saída em um fluxo de saída.
- **Tipo de biblioteca**: para eventos enviados pelos serviços da AWS, use os tipos na biblioteca [aws-lambda-java-events \(p. 599\)](#).

Se você definir seu próprio tipo de entrada, ele deve ser um objeto Java simples e antigo (POJO) desserializável e mutável, com um construtor padrão e propriedades para cada campo no evento. Chaves no evento que não mapeiam para uma propriedade e propriedades que não estejam incluídas no evento são descartadas sem erro.

O tipo de saída pode ser um objeto ou `void`. O tempo de execução serializa valores de retorno em texto. Se a saída for um objeto com campos, o tempo de execução serializa-o em um documento JSON. Se for um tipo que envolve um valor primitivo, o tempo de execução retornará uma representação de texto desse valor.

Interfaces do manipulador

A biblioteca [aws-lambda-java-core](#) define duas interfaces para métodos do manipulador. Use as interfaces fornecidas para simplificar a configuração do manipulador e validar a assinatura do método do manipulador no momento da compilação.

- [com.amazonaws.services.lambda.runtime.requestHandler](#)
- [com.amazonaws.services.lambda.runtime.requestStreamHandler](#)

A interface `RequestHandler` é um tipo genérico que usa dois parâmetros: o tipo de entrada e o tipo de saída. Ambos os tipos devem ser objetos. Quando você usa essa interface, o tempo de execução Java desserializa o evento em um objeto com o tipo de entrada e serializa a saída em texto. Use essa interface quando a serialização interna funcionar com seus tipos de entrada e saída.

Example [Handler.java](#): interface do manipulador

```
// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String, String>, String>{
    @Override
    public String handleRequest(Map<String, String> event, Context context)
```

Para usar sua própria serialização, implemente a interface `RequestStreamHandler`. Com essa interface, o Lambda transmite ao seu manipulador uma transmissão de entrada e uma de saída. O manipulador lê bytes do fluxo de entrada, grava no fluxo de saída e retorna um `void`.

O exemplo a seguir usa tipos de leitor e gravador em buffer para trabalhar com os fluxos de entrada e saída. Ele usa a biblioteca [Gson](#) para serialização e desserialização.

Example [HandlerStream.java](#)

```
import com.amazonaws.services.lambda.runtime.Context
import com.amazonaws.services.lambda.runtime.RequestStreamHandler
import com.amazonaws.services.lambda.runtime.LambdaLogger
...
// Handler value: example.HandlerStream
public class HandlerStream implements RequestStreamHandler {
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    @Override
    public void handleRequest(InputStream inputStream, OutputStream outputStream, Context context) throws IOException
```

```
{  
    LambdaLogger logger = context.getLogger();  
    BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream,  
Charset.forName("US-ASCII")));  
    PrintWriter writer = new PrintWriter(new BufferedWriter(new  
OutputStreamWriter(outputStream, Charset.forName("US-ASCII"))));  
    try  
    {  
        HashMap<String, String> event = gson.fromJson(reader, HashMap.class);  
        logger.log("STREAM TYPE: " + inputStream.getClass().toString());  
        logger.log("EVENT TYPE: " + event.getClass().toString());  
        writer.write(gson.toJson(event));  
        if (writer.checkError())  
        {  
            logger.log("WARNING: Writer encountered an error.");  
        }  
    }  
    catch (IllegalStateException | JsonSyntaxException exception)  
    {  
        logger.log(exception.toString());  
    }  
    finally  
    {  
        reader.close();  
        writer.close();  
    }  
}  
}
```

Código de exemplo do manipulador

O repositório do GitHub para este guia inclui aplicativos de exemplo que demonstram o uso de vários tipos e interfaces de manipuladores. Cada aplicativo de exemplo inclui scripts para fácil implantação e limpeza, um modelo do AWS SAM e recursos de suporte.

Aplicações de exemplo do Lambda em Java

- [blank-java](#): uma função Java que mostra o uso das bibliotecas Java do Lambda, registro em log, variáveis de ambiente, camadas, rastreamento do AWS X-Ray, testes de unidade e do AWS SDK.
- [java-basic](#): uma função Java mínima com testes de unidade e configuração de registro em log variável.
- [java-events](#)— Uma função Java mínima que usa a versão mais recente (3.0.0 e mais recente) do [aws-lambda-java-events](#) (p. 599) biblioteca. Estes exemplos não exigem o AWSSDK como dependência.
- [s3-java](#): uma função Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.

As aplicações `blank-java` e `s3-java` usam um evento de serviço da AWS como entrada e retornam uma string. O aplicativo `java-basic` inclui vários tipos de manipuladores:

- [Handler.java](#): recebe `Map<String, String>` como entrada.
- [HandlerInteger.java](#): recebe `Integer` como entrada.
- [HandlerList.java](#): recebe `List<Integer>` como entrada.
- [HandlerStream.java](#): recebe `InputStream` e `OutputStream` como entrada.
- [HandlerString.java](#): recebe uma `String` como entrada.
- [HandlerWeatherData.java](#): recebe um tipo personalizado como entrada.

Para testar diferentes tipos de manipuladores, basta alterar o valor do manipulador no modelo do AWS SAM. Para obter instruções detalhadas, consulte o arquivo `readme` do aplicativo de exemplo.

Implantar funções do Lambda em Java com arquivos .zip ou JAR

O código da função do AWS Lambda consiste em scripts ou programas compilados e as dependências deles. Você usa um pacote de implantação para implantar seu código de função no Lambda. O Lambda é compatível com dois tipos de pacotes de implantação: imagens de contêiner e arquivos .zip.

Esta página descreve como criar o seu pacote de implantação como um arquivo .zip ou Jar e, em seguida, usar o pacote para implantar o seu código de função para o AWS Lambda usando a AWS Command Line Interface (AWS CLI).

Seções

- [Prerequisites \(p. 599\)](#)
- [Ferramentas e bibliotecas \(p. 599\)](#)

Prerequisites

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Ferramentas e bibliotecas

O Lambda fornece as seguintes bibliotecas para funções em Java:

- [com.amazonaws:aws-lambda-java-core](#) (obrigatória): define as interfaces do método do manipulador e o objeto de contexto que o tempo de execução transmite ao manipulador. Se você definir seus próprios tipos de entrada, esta será a única biblioteca necessária.
- [com.amazonaws:aws-lambda-java-events](#): tipos de entrada para eventos de serviços que invocam funções do Lambda.
- [com.amazonaws:aws-lambda-java-log4j2](#): uma biblioteca appender do Apache Log4j 2 que você pode usar para adicionar o ID de solicitação da invocação atual aos [logs de função \(p. 611\)](#).
- [AWS SDK for Java 2.0](#): o AWSSDK oficial para a linguagem de programação Java.

Essas bibliotecas estão disponíveis no [repositório central do Maven](#). Adicione-as à sua definição de compilação da seguinte forma:

Gradle

```
dependencies {
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.1'
    implementation 'com.amazonaws:aws-lambda-java-events:3.1.0'
    runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.2.0'
}
```

Maven

```
<dependencies>
  <dependency>
```

```
<groupId>com.amazonaws</groupId>
<artifactId>aws-lambda-java-core</artifactId>
<version>1.2.1</version>
</dependency>
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-events</artifactId>
    <version>3.1.0</version>
</dependency>
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-log4j2</artifactId>
    <version>1.2.0</version>
</dependency>
</dependencies>
```

Para criar um pacote de implantação, compile o código e as dependências da sua função em um único arquivo .zip ou Java Archive (JAR). Para o Gradle, [use o tipo de compilação Zip \(p. 600\)](#). Para o Apache Maven, [use o plugin Maven Shade \(p. 601\)](#).

Note

Para manter o pacote de implantação pequeno, empacote as dependências da função em camadas. As camadas permitem gerenciar as suas dependências de forma independente, podem ser usadas por várias funções e podem ser compartilhadas com outras contas. Para obter mais informações, consulte [Criar e compartilhar camadas do Lambda \(p. 85\)](#).

É possível fazer upload do seu pacote de implantação usando o console do Lambda, a API do Lambda ou o AWS Serverless Application Model (AWS SAM).

Para fazer upload de um pacote de implantação com o console do Lambda

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Em Code source (Fonte do código), escolha Upload from (Fazer upload de).
4. Faça upload do pacote de implantação.
5. Escolha Save (Salvar).

Seções

- [Compilar um pacote de implantação com o Gradle \(p. 600\)](#)
- [Compilar um pacote de implantação com o Maven \(p. 601\)](#)
- [Fazer upload de um pacote de implantação com a API do Lambda \(p. 603\)](#)
- [Fazer upload de um pacote de implantação com o AWS SAM \(p. 604\)](#)

Compilar um pacote de implantação com o Gradle

Para criar um pacote de implantação com o código e as dependências da função, use o tipo de compilação [Zip](#).

Example build.gradle: tarefa de compilação

```
task buildZip(type: Zip) {
    from compileJava
    from processResources
    into('lib') {
```

```
        from configurations.runtimeClasspath
    }
}
```

Essa configuração de compilação produz um pacote de implantação no diretório build/distributions. A tarefa compileJava compila as classes da função. A tarefa processResources copia os recursos do projeto Java no diretório de destino, processando potencialmente em seguida. A instrução into('lib') então copia bibliotecas de dependência do classpath da compilação em uma pasta chamada lib.

Example build.gradle: dependências

```
dependencies {
    implementation platform('software.amazon.awssdk:bom:2.10.73')
    implementation 'software.amazon.awssdk:lambda'
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.1'
    implementation 'com.amazonaws:aws-lambda-java-events:3.1.0'
    implementation 'com.google.code.gson:gson:2.8.6'
    implementation 'org.apache.logging.log4j:log4j-api:2.13.0'
    implementation 'org.apache.logging.log4j:log4j-core:2.13.0'
    runtimeOnly 'org.apache.logging.log4j:log4j-slf4j18-impl:2.13.0'
    runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.2.0'
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.6.0'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.6.0'
}
```

O Lambda carrega arquivos JAR em Unicode em ordem alfabética. Se vários arquivos JAR no diretório lib contiverem a mesma classe, a primeira será usada. Use o script de shell a seguir para identificar classes duplicadas.

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
```

Compilar um pacote de implantação com o Maven

Para compilar um pacote de implantação com o Maven, use o [plugin Maven Shade](#). O plugin cria um arquivo JAR que contém o código de função compilado e todas as suas dependências.

Example pom.xml: configuração do 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>
```

Para compilar o pacote de implantação, use o 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:1.2.1 in the shaded jar.
[INFO] Including com.amazonaws:aws-lambda-java-events:jar:3.1.0 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] -----
```

Esse comando gera um arquivo JAR no diretório target.

Se você usar a biblioteca appender (aws-lambda-java-log4j2), também será necessário configurar um transformador para o plugin Maven Shade. A biblioteca do transformador combina versões de um arquivo de cache que aparecem na biblioteca appender e no Log4j.

Example pom.xml: configuração do plugin com o appender do 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.PluginsCacheFileTransformer"
                        </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>
```

Fazer upload de um pacote de implantação com a API do Lambda

Para atualizar o código de uma função com a AWS Command Line Interface (AWS CLI) ou o AWS SDK, use a operação de API [UpdateFunctionCode \(p. 965\)](#). Na AWS CLI, use o comando `update-function-code`: O comando a seguir faz upload de um pacote de implantação chamado `my-function.zip` no diretório atual.

```
aws lambda update-function-code --function-name my-function --zip-file fileb://my-function.zip
```

Você deve ver a saída a seguir:

```
{  
    "FunctionName": "my-function",  
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",  
    "Runtime": "java8",  
    "Role": "arn:aws:iam::123456789012:role/lambda-role",  
    "Handler": "example.Handler",  
    "CodeSha256": "Qf0hMc1I2di6YFMi9aXm3JtGTmcDbjniEuiYonYptAk=",  
    "Version": "$LATEST",  
    "TracingConfig": {  
        "Mode": "Active"  
    },  
    "RevisionId": "983ed1e3-ca8e-434b-8dc1-7d72ebadd83d",  
    ...  
}
```

Se o pacote de implantação for maior que 50 MB, não será possível fazer upload diretamente. Faça upload dele para um bucket do Amazon Simple Storage Service (Amazon S3) e aponte o Lambda para o objeto. Os exemplos de comando a seguir fazem upload de um pacote de implantação em um bucket do S3 chamado `my-bucket` e o utilizam para atualizar o código de uma função:

```
aws s3 cp my-function.zip s3://my-bucket
```

Você deve ver a saída a seguir:

```
upload: my-function.zip to s3://my-bucket/my-function
```

```
aws lambda update-function-code --function-name my-function \  
    --s3-bucket my-bucket --s3-key my-function.zip
```

Você deve ver a saída a seguir:

```
{  
    "FunctionName": "my-function",  
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",  
    "Runtime": "java8",  
    "Role": "arn:aws:iam::123456789012:role/lambda-role",  
    "Handler": "example.Handler",  
    "CodeSha256": "Qf0hMc1I2di6YFMi9aXm3JtGTmcDbjniEuiYonYptAk=",  
    "Version": "$LATEST",  
    "TracingConfig": {  
        "Mode": "Active"  
    },  
    "RevisionId": "983ed1e3-ca8e-434b-8dc1-7d72ebadd83d",  
    ...  
}
```

```
    ...  
}
```

É possível usar esse método para fazer upload de pacotes de função até 250 MB (descompactado).

Fazer upload de um pacote de implantação com o AWS SAM

É possível usar o AWS SAM para automatizar implantações do código, da configuração e das dependências da sua função. O AWS SAM é uma extensão do AWS CloudFormation que fornece uma sintaxe simplificada para definir aplicações sem servidor. O seguinte exemplo de modelo define uma função com um pacote de implantação no diretório build/distributions que o Gradle usa:

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: java8  
      Description: Java function  
      MemorySize: 512  
      Timeout: 10  
      # Function's execution role  
      Policies:  
        - AWSLambdaBasicExecutionRole  
        - AWSLambda_ReadOnlyAccess  
        - AWSXrayWriteOnlyAccess  
        - AWSLambdaVPCAccessExecutionRole  
      Tracing: Active
```

Para criar a função, use os comandos package e deploy. Esses comandos são personalizações para a AWS CLI. Eles envolvem outros comandos para fazer upload do pacote de implantação no Amazon S3, reescrevem o modelo com o URI do objeto e atualizam o código da função.

O exemplo de script a seguir executa uma compilação do Gradle e faz upload do pacote de implantação que ele cria. Ele cria uma pilha do AWS CloudFormation na primeira vez que você executá-lo. Se a pilha já existir, o script a atualizará.

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
```

Para obter um modelo funcional completo, consulte as seguintes aplicações de exemplo:

Aplicações de exemplo do Lambda em Java

- [blank-java](#): uma função Java que mostra o uso das bibliotecas Java do Lambda, registro em log, variáveis de ambiente, camadas, rastreamento do AWS X-Ray, testes de unidade e do AWS SDK.
- [java-basic](#): uma função Java mínima com testes de unidade e configuração de registro em log variável.

- [java-events](#)— Uma função Java mínima que usa a versão mais recente (3.0.0 e mais recente) da[aws-lambda-java-events \(p. 599\)](#) biblioteca. Estes exemplos não exigem oAWS SDK como dependência.
- [s3-java](#): uma função Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.

Implantar funções do Lambda em Java com imagens de contêiner

Você pode implantar seu código de função do Lambda como uma [imagem de contêiner \(p. 267\)](#). A AWS oferece os seguintes recursos para ajudar você a criar uma imagem de contêiner para sua função Java:

- [AWSImagens base para o Lambda](#)

Essas imagens base são pré-carregadas com um tempo de execução de linguagem e outros componentes necessários para executar a imagem no Lambda. AWS fornece um Dockerfile para cada uma das imagens de base para ajudar a criar sua imagem de contêiner.

- Clientes de interface de tempo de execução de

Se você usar uma imagem de base de comunidade ou de empresa privada, adicione um cliente de interface de tempo de execução à imagem base para torná-lo compatível com o Lambda.

O fluxo de trabalho de uma função definida como uma imagem de contêiner inclui as seguintes etapas:

1. Crie sua imagem de contêiner usando os recursos listados neste tópico.
2. Faça upload da imagem em seu registro do contêiner do Amazon ECR. Consulte os passos 7 a 9 no [Criar imagem \(p. 268\)](#).
3. [Criar \(p. 91\)](#) a função do Lambda ou [atualizar o código da função \(p. 92\)](#) para implantar a imagem em uma função existente.

Tópicos

- [AWSImagens de base da para Java \(p. 606\)](#)
- [Usar uma imagem base Java \(p. 607\)](#)
- [Clientes de interface de tempo de execução para Java \(p. 607\)](#)
- [Implantar a imagem do contêiner \(p. 607\)](#)

[AWSImagens de base da para Java](#)

AWSA oferece as seguintes imagens de base para Java:

Tags	Tempo de execução	Sistema operacional	Dockerfile
11	Java 11 (Corretto)	Amazon Linux 2	Dockerfile para Java 11 no GitHub
8.al2	Java 8 (Corretto)	Amazon Linux 2	Dockerfile para Java 8.al2 no GitHub
8	Java 8 (OpenJDK)	Amazon Linux 2018.03	Dockerfile para Java 8 no GitHub

Repositório do Docker Hub: [amazon/aws-lambda-java](#)

Repositório do Amazon ECR: [gallery.ecr.aws/lambda/java](#)

Usar uma imagem base Java

Para obter instruções sobre como usar uma imagem base Java, escolha a guia usage (uso) em [Imagens base do Lambda para Java](#) no repositório do Amazon ECR.

As instruções também estão disponíveis em [imagens base do Lambda para Java](#) no repositório do Docker Hub.

Cientes de interface de tempo de execução para Java

Instale o cliente de interface de tempo de execução para Java usando o gerenciador de pacotes Apache Maven. Adicione o seguinte ao arquivo `pom.xml`:

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-runtime-interface-client</artifactId>
    <version>1.0.0</version>
</dependency>
```

Para obter detalhes do pacote, consulte [Lambda RIC no Maven Central Repository](#).

Você também pode visualizar o código-fonte do cliente Java no repositório de [Bibliotecas de Suporte Java do AWS Lambda](#) no GitHub.

Depois que a imagem de contêiner estiver no registro do contêiner do Amazon ECR, você poderá [criar e executar \(p. 90\)](#) a função do Lambda.

Implantar a imagem do contêiner

Para uma nova função, você implanta a imagem Java ao [criar a função \(p. 91\)](#). Para uma função existente, se você reconstruir a imagem do contêiner, precisará reimplantar a imagem ao [atualizar o código da função \(p. 92\)](#).

AWS LambdaObjeto de contexto do em Java

Quando o Lambda executa a função, ele transmite um objeto de contexto para o [handler \(p. 595\)](#). Esse objeto fornece métodos e propriedades que fornecem informações sobre a invocação, a função e o ambiente de execução.

Métodos de contexto

- `getRemainingTimeInMillis()`— Retorna o número de milissegundos restantes antes do tempo limite da execução.
- `getFunctionName()`: retorna o nome da função do Lambda.
- `getFunctionVersion()`— Retorna [oversion \(p. 106\)](#)da função do.
- `getInvokedFunctionArn()`: retorna o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um número de versão ou alias.
- `getMemoryLimitInMB()`— Retorna a quantidade de memória alocada para a função.
- `getAwsRequestId()`— Retorna o identificador de invocação da solicitação.
- `getLogGroupName()`— Retorna o grupo de logs para a função do.
- `getLogStreamName()`— Retorna o stream de log para a instância da função.
- `getIdentity()`— (aplicativos móveis) Retorna informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
- `getClientContext()`— (aplicativos móveis) Retorna o contexto do cliente que é fornecido ao Lambda pelo aplicativo cliente.
- `getLogger()`: retorna o [objeto logger \(p. 611\)](#) para a função.

O exemplo a seguir mostra uma função que usa o objeto de contexto para acessar o logger do Lambda.

Example Handler.java

```
package example;
import com.amazonaws.services.lambda.runtime.Context
import com.amazonaws.services.lambda.runtime.RequestHandler
import com.amazonaws.services.lambda.runtime.LambdaLogger
...
// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String, String>, String>{
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    @Override
    public String handleRequest(Map<String, String> event, Context context)
    {
        LambdaLogger logger = context.getLogger();
        String response = new String("200 OK");
        // 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));
        logger.log("EVENT TYPE: " + event.getClass().toString());
        return response;
    }
}
```

A função serializa o objeto de contexto em JSON e o registra em seu fluxo de log.

Example saída do log

```
START RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Version: $LATEST
...
CONTEXT:
{
    "memoryLimit": 512,
    "awsRequestId": "6bc28136-xmpl-4365-b021-0ce6b2e64ab0",
    "functionName": "java-console",
    ...
}
...
END RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0
REPORT RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Duration: 198.50 ms Billed Duration:
200 ms Memory Size: 512 MB Max Memory Used: 90 MB Init Duration: 524.75 ms
```

A interface para o objeto de contexto está disponível na biblioteca [aws-lambda-java-core](#). É possível implementar essa interface para criar uma classe de contexto para teste. O exemplo a seguir mostra uma classe de contexto que retorna valores fictícios para a maioria das propriedades e um logger de teste funcional.

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 LambdaLogger getLogger(){
        return new TestLogger();
    }
}
```

Para obter mais informações sobre registro em log, consulte [AWS Lambda Registro em log da função do em Java \(p. 611\)](#).

Contexto em aplicativos de exemplo

O repositório do GitHub para este guia inclui aplicativos de exemplo que demonstram o uso do objeto de contexto. Cada aplicativo de exemplo inclui scripts para fácil implantação e limpeza, um modelo do AWS Serverless Application Model (AWS SAM) e recursos de suporte.

Aplicações de exemplo do Lambda em Java

- [blank-java](#): uma função Java que mostra o uso das bibliotecas Java do Lambda, registro em log, variáveis de ambiente, camadas, rastreamento do AWS X-Ray, testes de unidade e do AWS SDK.
- [java-basic](#): uma função Java mínima com testes de unidade e configuração de registro em log variável.

- [java-events](#)— Uma função Java mínima que usa a versão mais recente (3.0.0 e mais recente) da[aws-lambda-java-events \(p. 599\)](#)biblioteca. Estes exemplos não exigem o AWSSDK como dependência.
- [s3-java](#): uma função Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.

Todos os aplicativos de exemplo têm uma classe de contexto de teste para testes de unidade. O aplicativo `java-basic` mostra como usar o objeto de contexto para obter um logger. Ele usa SLF4J e Log4J 2 para fornecer um logger que funcione para testes de unidade local.

AWS Lambda Registro em log da função do em Java

O AWS Lambda monitora automaticamente as funções do Lambda em seu nome e envia métricas da função para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente do tempo de execução do Lambda envia detalhes sobre cada invocação à transmissão de logs e transmite os logs e outras saídas do código de sua função.

Esta página descreve como produzir a saída de logs usando o código de sua função do Lambda ou acessar os logs usando a AWS Command Line Interface, o console do Lambda ou o console do CloudWatch.

Seções

- [Criar uma função que retorna logs \(p. 611\)](#)
- [Usar o console do Lambda \(p. 612\)](#)
- [Usando o console do CloudWatch \(p. 612\)](#)
- [Usar a AWS Command Line Interface \(AWS CLI\) \(p. 613\)](#)
- [Excluir logs \(p. 615\)](#)
- [Registro em log avançado com Log4j 2 e SLF4J \(p. 615\)](#)
- [Código de exemplo de registro em log \(p. 617\)](#)

Criar uma função que retorna logs

Para gerar os logs do código da função, use métodos no `java.lang.System` ou qualquer módulo de registro em log que grave no `stdout` ou no `stderr`. A biblioteca [aws-lambda-java-core \(p. 599\)](#) fornece uma classe do logger chamada `LambdaLogger` que você pode acessar a partir do objeto de contexto. A classe do logger oferece suporte a logs de várias linhas.

O exemplo a seguir usa o logger `LambdaLogger` fornecido pelo objeto de contexto.

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 do 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:  
200 ms Memory Size: 512 MB Max Memory Used: 90 MB Init Duration: 524.75 ms
```

O tempo de execução do Java registra em log as linhas START, END e REPORT para cada invocação. A linha do relatório fornece os seguintes detalhes:

Registro em log de relatório

- RequestId: o ID de solicitação exclusivo para a invocação.
- Duração: a quantidade de tempo que o método de manipulador da função gastou processando o evento.
- Duração faturada: a quantia de tempo faturada para a invocação.
- Tamanho da memória: a quantidade de memória alocada para a função.
- Memória máxima utilizada: a quantidade de memória utilizada pela função.
- Duração inicial: para a primeira solicitação atendida, a quantidade de tempo que o tempo de execução levou para carregar a função e executar o código fora do método do manipulador.
- XRAY Traceld: para solicitações rastreadas, o [ID de rastreamento do AWS X-Ray \(p. 477\)](#).
- SegmentId: para solicitações rastreadas, o ID do segmento do X-Ray.
- Amostragem: para solicitações rastreadas, o resultado da amostragem.

Usar o console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda. Para obter mais informações, consulte [Acessar o Amazon CloudWatch Logs para o AWS Lambda \(p. 720\)](#).

Usando o console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a [página Log groups](#) (Grupos de log).
2. Escolha o grupo de logs de sua função (`/aws/lambda/nome-de-sua-função`).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma [instância da sua função](#) (p. 219). Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos instrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

Para usar uma aplicação de exemplo que correlaciona os logs e os rastreamentos com o X-Ray, consulte [Aplicativo de exemplo de processador de erros para o AWS Lambda](#) (p. 500).

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Você pode usar a [AWS CLI](#) para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Você deve ver a saída a seguir:

```
{  
    "StatusCode": 200,  
    "LogResult":  
        "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",  
    "ExecutedVersion": "$LATEST"  
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \  
--query 'LogResult' --output text | base64 -d
```

Você deve ver a saída a seguir:

```
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
```

O utilitário base64 está disponível no Linux, macOS e [Ubuntu no Windows](#). Os usuários do macOS precisam usar `base64 -D`.

Example get-logs.sh script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa sed para remover as aspas do arquivo de saída e fica inativo por 15 segundos para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como `get-logs.sh`.

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

```
#!/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 $(cat
out) --limit 5
```

Example macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod -R 755 get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

Você deve ver a saída a seguir:

```
{
  "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"
}
```

Excluir logs

Os grupos de logs não são excluídos automaticamente quando você exclui uma função. Para evitar armazenar logs indefinidamente, exclua o grupo de logs ou[configurar um período de retenção](#)após o qual os logs são excluídos automaticamente.

Registro em log avançado com Log4j 2 e SLF4J

Para personalizar a saída de log, oferecer suporte ao registro em log durante testes de unidade e registrar chamadas do AWS SDK, use o Apache Log4j 2 com o SLF4J. O Log4j é uma biblioteca de registro em log para programas Java que permite configurar níveis de log e usar bibliotecas de appender. O SLF4J é uma biblioteca de fachada que permite alterar qual biblioteca você usa sem alterar o código da função.

Para adicionar o ID de solicitação aos logs da sua função, use o appender na biblioteca [aws-lambda-java-log4j2 \(p. 599\)](#). O exemplo a seguir mostra um arquivo de configuração Log4j 2 que adiciona um timestamp e o ID de solicitação a todos os logs.

Example [src/main/resources/log4j2.xml](#): configuração do appender

```
<Configuration status="WARN">
  <Appenders>
    <Lambda name="Lambda">
      <PatternLayout>
        <pattern>%d{yyyy-MM-dd HH:mm:ss} %X{AWSRequestId} %-5p %c{1} - %m%n</pattern>
      </PatternLayout>
    </Lambda>
  </Appenders>
  <Loggers>
    <Root level="INFO">
      <AppenderRef ref="Lambda"/>
    </Root>
    <Logger name="software.amazon.awssdk" level="WARN" />
    <Logger name="software.amazon.awssdk.request" level="DEBUG" />
  </Loggers>
</Configuration>
```

```
</Configuration>
```

Com essa configuração, cada linha é precedida com a data, a hora, o ID da solicitação, o nível do log e o nome da classe.

Example formato de log com o appender

```
START RequestId: 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 Version: $LATEST
2020-03-18 08:52:43 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 INFO Handler - ENVIRONMENT
VARIABLES:
{
    "_HANDLER": "example.Handler",
    "AWS_EXECUTION_ENV": "AWS_Lambda_java8",
    "AWS_LAMBDA_FUNCTION_MEMORY_SIZE": "512",
    ...
}
2020-03-18 08:52:43 6bc28136-xmpl-4365-b021-0ce6b2e64ab0 INFO Handler - CONTEXT:
{
    "memoryLimit": 512,
    "awsRequestId": "6bc28136-xmpl-4365-b021-0ce6b2e64ab0",
    "functionName": "java-console",
    ...
}
```

O SLF4J é uma biblioteca de fachada para registro em log em código Java. No código da função, use o logger factory SLF4J para recuperar um logger com métodos para níveis de log como `info()` e `warn()`. Na configuração da compilação, inclua a biblioteca de registro em log e o adaptador SLF4J no classpath. Alterando as bibliotecas na configuração de compilação, é possível alterar o tipo de logger sem alterar o código da função. O SLF4J é necessário para capturar logs do SDK for Java.

No exemplo a seguir, a classe de manipulador usa o SLF4J para recuperar um logger.

Example [src/main/java/example/Handler.java](#): registro em log com o SLF4J

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

// Handler value: example.Handler
public class Handler implements RequestHandler<SQSEvent, String>{
    private static final Logger logger = LoggerFactory.getLogger(Handler.class);
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    LambdaAsyncClient lambdaClient = LambdaAsyncClient.create();
    @Override
    public String handleRequest(SQSEvent event, Context context)
    {
        String response = new String();
        // call Lambda API
        logger.info("Getting account settings");
        CompletableFuture<GetAccountSettingsResponse> accountSettings =
            lambdaClient.getAccountSettings(GetAccountSettingsRequest.builder().build());
        // log execution details
        logger.info("ENVIRONMENT VARIABLES: {}", gson.toJson(System.getenv()));
    ...
}
```

A configuração de compilação usa dependências de tempo de execução no appender e no adaptador SLF4J do Lambda, além de dependências de implementação no Log4J 2.

Example [build.gradle](#): dependências de registro em log

```
dependencies {
```

```
implementation platform('software.amazon.awssdk:bom:2.10.73')
implementation platform('com.amazonaws:aws-xray-recorder-sdk-bom:2.4.0')
implementation 'software.amazon.awssdk:lambda'
implementation 'com.amazonaws:aws-xray-recorder-sdk-core'
implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-core'
implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-v2'
implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-v2-instrumentor'
implementation 'com.amazonaws:aws-lambda-java-core:1.2.1'
implementation 'com.amazonaws:aws-lambda-java-events:3.1.0'
implementation 'com.google.code.gson:gson:2.8.6'
implementation 'org.apache.logging.log4j:log4j-api:2.13.0'
implementation 'org.apache.logging.log4j:log4j-core:2.13.0'
runtimeOnly 'org.apache.logging.log4j:log4j-slf4j18-impl:2.13.0'
runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.2.0'
testImplementation 'org.junit.jupiter:junit-jupiter-api:5.6.0'
testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.6.0'
}
```

Quando você executa seu código localmente para testes, o objeto de contexto com o logger do Lambda não está disponível e não há nenhum ID de solicitação para o appender do Lambda usar. Para obter exemplos de configurações de teste, consulte as aplicações de exemplo na próxima seção.

Código de exemplo de registro em log

O repositório do GitHub para este guia inclui aplicativos de exemplo que demonstram o uso de várias configurações de registro em log. Cada aplicativo de exemplo inclui scripts para fácil implantação e limpeza, um modelo do AWS SAM e recursos de suporte.

Aplicações de exemplo do Lambda em Java

- [blank-java](#): uma função Java que mostra o uso das bibliotecas Java do Lambda, registro em log, variáveis de ambiente, camadas, rastreamento do AWS X-Ray, testes de unidade e do AWS SDK.
- [java-basic](#): uma função Java mínima com testes de unidade e configuração de registro em log variável.
- [java-events](#)— Uma função Java mínima que usa a versão mais recente (3.0.0 e mais recente) do[aws-lambda-java-events](#) ([p. 599](#))biblioteca. Estes exemplos não exigem oAWSSDK como dependência.
- [s3-java](#): uma função Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.

O aplicativo de exemplo `java-basic` mostra uma configuração mínima de registro em log que oferece suporte a testes de registro em log. O código do manipulador usa o logger `LambdaLogger` fornecido pelo objeto de contexto. Para testes, o aplicativo usa uma classe `TestLogger` personalizada que implementa a interface `LambdaLogger` com um logger Log4j 2. Ele usa o SLF4J como fachada para compatibilidade com o AWS SDK. As bibliotecas de registro em log são excluídas da saída de compilação para manter o pacote de implantação pequeno.

O exemplo de aplicação `blank-java` se baseia na configuração básica com o registro em log do AWS SDK e o appender Log4j 2 do Lambda. Ele usa o Log4j 2 no Lambda com o appender personalizado que adiciona o ID de solicitação de invocação à cada linha.

AWS Lambda Erros da função do em Java

Quando o código gera um erro, o Lambda gera uma representação JSON do erro. Esse documento de erro aparece no log de invocação e, para invocações síncronas, na saída.

Esta página descreve como exibir erros de invocação de função do Lambda para o tempo de execução do Java usando o console do Lambda e a AWS CLI.

Seções

- [Syntax \(p. 618\)](#)
- [Como funcionam \(p. 619\)](#)
- [Criar uma função que retorna exceções \(p. 619\)](#)
- [Usar o console do Lambda \(p. 620\)](#)
- [Usar a AWS Command Line Interface \(AWS CLI\) \(p. 621\)](#)
- [Tratamento de erros em outros serviços da AWS \(p. 621\)](#)
- [Aplicativos de exemplo \(p. 622\)](#)
- [Próximas etapas \(p. 622\)](#)

Syntax

No exemplo a seguir, o tempo de execução não consegue desserializar o evento em um objeto. A entrada é um tipo JSON válido, mas não corresponde ao tipo esperado pelo método do manipulador.

Example Erro Lambda de execução de

```
{  
  "errorMessage": "An error occurred during JSON parsing",  
  "errorType": "java.lang.RuntimeException",  
  "stackTrace": [],  
  "cause": {  
    "errorMessage": "com.fasterxml.jackson.databind.exc.InvalidFormatException: Can not  
construct instance of java.lang.Integer from String value '1000,10': not a valid Integer  
value\n at [Source: lambdainternal.util.NativeMemoryAsInputStream@35fc6dc4; line: 1,  
column: 1] (through reference chain: java.lang.Object[0])",  
    "errorType": "java.io.UncheckedIOException",  
    "stackTrace": [],  
    "cause": {  
      "errorMessage": "Can not construct instance of java.lang.Integer  
from String value '1000,10': not a valid Integer value\n at [Source:  
lambdainternal.util.NativeMemoryAsInputStream@35fc6dc4; line: 1, column: 1] (through  
reference chain: java.lang.Object[0])",  
      "errorType": "com.fasterxml.jackson.databind.exc.InvalidFormatException",  
      "stackTrace": [  
  
        "com.fasterxml.jackson.databind.exc.InvalidFormatException.from(InvalidFormatException.java:55)",  
  
        "com.fasterxml.jackson.databind.DeserializationContext.weirdStringException(DeserializationContext.java:  
          ...  
        ]  
      }  
    }  
  }  
}
```

Como funcionam

Ao invocar uma função do Lambda, o Lambda recebe a solicitação de invocação e valida as permissões de sua função de execução, verifica se o documento do evento é um documento JSON válido e verifica valores de parâmetros.

Se a solicitação for validada, o Lambda a envia para uma instância da função. O ambiente de [tempo de execução do Lambda \(p. 214\)](#) converte o documento do evento em um objeto e o transmite ao handler da função.

Se o Lambda encontra um erro, ele retorna um tipo de exceção, uma mensagem e o código HTTP do status que indica a causa do erro. O cliente ou o serviço que invocou a função do Lambda pode processar o erro de maneira programática ou transmiti-lo a um usuário final. O comportamento correto de tratamento de erros depende do tipo de aplicativo, do público e da origem do erro.

A lista a seguir descreve o intervalo de códigos de status que você pode receber do Lambda.

2xx

Um erro da série 2xx com um cabeçalho `X-Amz-Function-Error` na resposta indica um erro de tempo de execução ou de função do Lambda. Um código de status da série 2xx indica que o Lambda aceitou a solicitação, mas em vez de um código de erro, o Lambda indica o erro incluindo o cabeçalho `X-Amz-Function-Error` na resposta.

4xx

Um erro da série 4xx indica um erro que o cliente ou serviço que fez a invocação pode corrigir modificando a solicitação, solicitando permissão ou tentando a solicitação novamente. Os erros da série 4xx diferentes de 429 geralmente indicam um erro na solicitação.

5xx

Um erro da série 5xx indica um problema com o Lambda ou com a configuração/recursos da função. Os erros da série 5xx podem indicar uma condição temporária que pode ser resolvida sem nenhuma ação do usuário. O cliente ou serviço que fez a invocação não podem solucionar esses problemas, mas o proprietário de uma função do Lambda pode ser capaz de corrigi-los.

Para obter uma lista completa de erros de invocação, consulte [Erros de InvokeFunction \(p. 877\)](#).

Criar uma função que retorna exceções

Você pode criar uma função do Lambda que exiba mensagens de erro legíveis para os usuários.

Note

Para testar este código, você deve incluir `InputLengthException.java` na sua pasta src do projeto.

Example `src/main/java/example/HandlerDivide.java`: exceção de tempo de execução

```
import java.util.List;

// Handler value: example.HandlerDivide
public class HandlerDivide implements RequestHandler<List<Integer>, Integer>{
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    @Override
    public Integer handleRequest(List<Integer> event, Context context)
    {
        LambdaLogger logger = context.getLogger();
        // process event
        if (event.size() != 2)
```

```
{  
    throw new InputLengthException("Input must be an array that contains 2 numbers.");  
}  
int numerator = event.get(0);  
int denominator = event.get(1);  
logger.log("EVENT: " + gson.toJson(event));  
logger.log("EVENT TYPE: " + event.getClass().toString());  
return numerator/denominator;  
}  
}
```

Quando a função gera `InputLengthException`, o tempo de execução Java serializa-o no documento a seguir.

Example documento de erro (espaço em branco adicionado)

```
{  
    "errorMessage": "Input must contain 2 numbers.",  
    "errorType": "java.lang.InputLengthException",  
    "stackTrace": [  
        "example.HandlerDivide.handleRequest(HandlerDivide.java:23)",  
        "example.HandlerDivide.handleRequest(HandlerDivide.java:14)"  
    ]  
}
```

Neste exemplo, `InputLengthException` é uma `RuntimeException`. A `RequestHandler interface` (p. 597) não permite exceções verificadas. A interface `RequestStreamHandler` oferece suporte à geração de erros `IOException`.

A instrução de retorno no exemplo anterior também pode gerar uma exceção de tempo de execução.

```
return numerator/denominator;
```

Este código pode retornar um erro aritmético.

```
{"errorMessage": "/ by zero", "errorType": "java.lang.ArithmetricException", "stackTrace": [  
    "example.HandlerDivide.handleRequest(HandlerDivide.java:28)", "example.HandlerDivide.handleRequest(Hand
```

Usar o console do Lambda

Você pode invocar sua função no console do Lambda configurando um evento de teste e visualizando a saída. A saída é capturada pelos logs de execução da função e, quando o `rastreamento ativo` (p. 477) está habilitado, pelo AWS X-Ray.

Para invocar uma função no console do Lambda

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Test (Testar).
4. Selecione New event (Novo evento) e escolha um Event template (Modelo de evento) na lista suspensa.
5. Insira um nome para o evento de teste.
6. Digite o JSON para o evento de teste.
7. Escolha Create event (Criar evento).
8. Escolha Invoke (Invocar).

O console do Lambda invoca sua função [de forma síncrona \(p. 158\)](#) e exibe o resultado. Para ver a resposta, os logs e outras informações, expanda a seção Details (Detalhes).

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Quando você invoca uma função do Lambda na AWS CLI, a AWS CLI divide a resposta em dois documentos. A resposta da AWS CLI é exibida no prompt de comando. Se ocorreu um erro, a resposta contém um campo `FunctionError`. A resposta ou o erro de invocação retornado pela função é gravado no arquivo de saída. Por exemplo, o `output.json` ou o `output.txt`.

O exemplo de comando `invoke` a seguir demonstra como invocar uma função e escrever a resposta de invocação em um arquivo `output.txt`.

```
aws lambda invoke \
--function-name my-function \
--cli-binary-format raw-in-base64-out \
--payload '{"key1": "value1", "key2": "value2", "key3": "value3"}' output.txt
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

Você deve ver a resposta da AWS CLI em seu prompt de comando:

```
{  
    "StatusCode": 200,  
    "FunctionError": "Unhandled",  
    "ExecutedVersion": "$LATEST"  
}
```

Você deve ver a resposta de invocação de função no arquivo `output.txt`. Também é possível visualizar a saída no mesmo prompt de comando usando:

```
cat output.txt
```

Você deve ver a resposta de invocação no prompt de comando.

```
{"errorMessage": "Input must contain 2  
numbers.", "errorType": "java.lang.InputLengthException", "stackTrace":  
["example.HandlerDivide.handleRequest(HandlerDivide.java:23)", "example.HandlerDivide.handleRequest(Han
```

O Lambda também grava até 256 KB do objeto de erro nos logs de função. Para obter mais informações, consulte [AWS Lambda Registro em log da função do em Java \(p. 611\)](#).

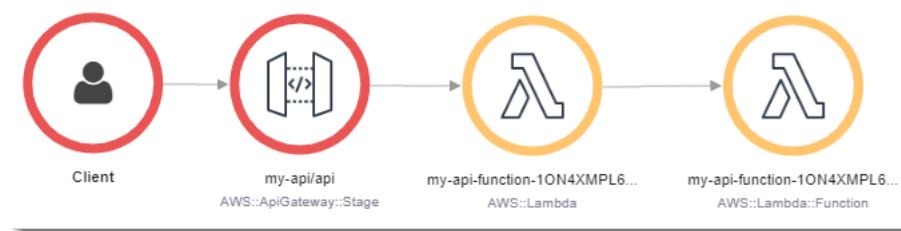
Tratamento de erros em outros serviços da AWS

Quando outro serviço da AWS invoca sua função, o serviço escolhe o tipo de invocação e o comportamento de repetição. Os serviços da AWS podem invocar sua função em um agendamento, em resposta a um evento de ciclo de vida em um recurso ou para atender a uma solicitação de um usuário.

Alguns serviços invocam funções de forma assíncrona e permitem que o Lambda trate erros, enquanto outros fazem novas tentativas ou transmitam os erros de volta ao usuário.

Por exemplo, o API Gateway trata todos os erros de invocação e função como erros internos. Se a API do Lambda rejeitar a solicitação de invocação, o API Gateway retornará um código de erro 500. Se a função é executada, mas retorna um erro ou retorna uma resposta no formato errado, o API Gateway retorna um código de erro 502. Para personalizar a resposta de erro, é necessário detectar erros no código e formatar uma resposta no formato necessário.

Recomendamos usar AWS X-Ray para determinar a fonte de um erro e a respectiva causa. O X-Ray permite localizar qual componente encontrou um erro e ver detalhes sobre os erros. O exemplo a seguir mostra um erro de função que resultou em uma resposta 502 do API Gateway.



Para obter mais informações, consulte [Instrumentação do código Java no AWS Lambda \(p. 623\)](#).

Aplicativos de exemplo

O repositório do GitHub para este guia inclui aplicativos de exemplo que demonstram o uso dos erros. Cada aplicativo de exemplo inclui scripts para fácil implantação e limpeza, um modelo do AWS Serverless Application Model (AWS SAM) e recursos de suporte.

Aplicações de exemplo do Lambda em Java

- [blank-java](#): uma função Java que mostra o uso das bibliotecas Java do Lambda, registro em log, variáveis de ambiente, camadas, rastreamento do AWS X-Ray, testes de unidade e do AWS SDK.
- [java-basic](#): uma função Java mínima com testes de unidade e configuração de registro em log variável.
- [java-events](#)— Uma função Java mínima que usa a versão mais recente (3.0.0 e mais recente) do[aws-lambda-java-events \(p. 599\)](#)biblioteca. Estes exemplos não exigem oAWSSDK como dependência.
- [s3-java](#): uma função Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.

A função `java-basic` inclui um manipulador (`HandlerDivide`) que retorna uma exceção de tempo de execução personalizada. O manipulador `HandlerStream` implementa o `RequestStreamHandler` e pode gerar uma exceção `IOException` marcada.

Próximas etapas

- Saiba como mostrar eventos de registro em log para sua função do Lambda na página [the section called “Registro em log” \(p. 611\)](#).

Instrumentação do código Java no AWS Lambda

O Lambda se integra ao AWS X-Ray para permitir que você rastreie, depure e otimize aplicativos do Lambda. É possível usar o X-Ray para rastrear uma solicitação à medida que ela atravessa recursos na aplicação, da API de frontend ao armazenamento e aos banco de dados no backend. Ao simplesmente adicionar a biblioteca do X-Ray SDK à configuração de compilação, é possível registrar erros e latência para qualquer chamada que a função faça para um serviço da AWS.

O X-Ray Mapa de serviço mostra o fluxo de solicitações através de seu aplicativo. O exemplo a seguir do aplicativo de exemplo [processador de erros \(p. 500\)](#) mostra um aplicativo com duas funções. A função principal processa eventos e, às vezes, retorna erros. A segunda função processa erros que aparecem no primeiro grupo de logs e usa o AWS SDK para chamar o X-Ray, o Amazon S3 e o Amazon CloudWatch Logs.



Para rastrear solicitações que não têm um cabeçalho de rastreamento, ative o rastreamento ativo na configuração da sua função.

Para ativar o rastreamento ativo

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Selecione **Configuração**, depois, escolha **Ferramentas de monitoramento**.
4. Selecione **Edit**.
5. Em X-Ray, habilite o **Active tracing (Rastreamento ativo)**.
6. Escolha **Save (Salvar)**.

Pricing

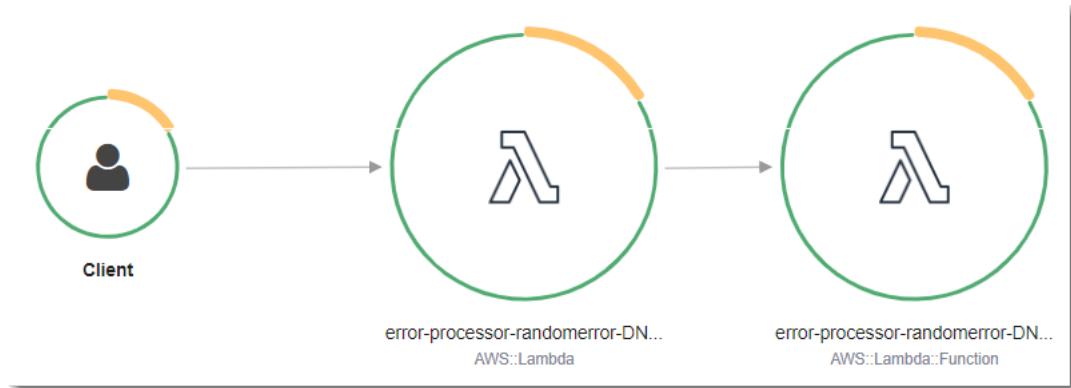
O X-Ray tem um nível gratuito vitalício. Além do limite do nível gratuito, o X-Ray cobra por armazenamento e recuperação de rastreamento. Para obter detalhes, consulte [Definição de preço do AWS X-Ray](#).

Sua função precisa de permissão para carregar dados de rastreamento no X-Ray. Quando você habilita o rastreamento ativo no console do Lambda, o Lambda adiciona as permissões necessárias à [função de](#)

execução (p. 57) da função. Caso contrário, adicione a política [AWSXRayDaemonWriteAccess](#) à função de execução.

O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento é eficiente, enquanto ainda fornece uma amostra representativa das solicitações que a sua aplicação serviu. A regra de amostragem padrão é uma solicitação por segundo e 5% de solicitações adicionais. Esta taxa de amostragem não pode ser configurada para funções do Lambda.

Quando o rastreamento ativo está habilitado, o Lambda registra um rastreamento para um subconjunto de invocações. Lambda registra doisSegmentos do, que cria dois nós no mapa de serviço. O primeiro nó representa o serviço Lambda que recebe a solicitação de invocação. O segundo nó é gravado pelo [tempo de execução \(p. 19\)](#)da função.

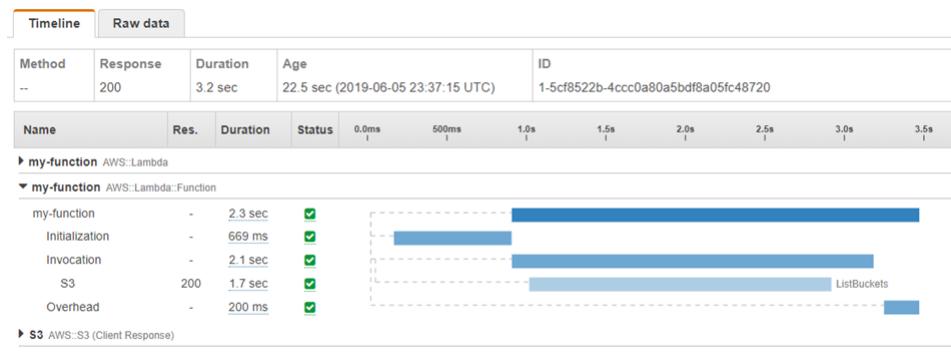


Para registrar detalhes sobre chamadas que sua função faz para outros recursos e serviços, adicione o X-Ray SDK for Java à sua configuração de compilação. O exemplo a seguir mostra uma configuração de compilação do Gradle que inclui as bibliotecas que permitem a instrumentação automática de clientes AWS SDK for Java 2.x.

Example [build.gradle](#): rastrear dependências

```
dependencies {  
    implementation platform('software.amazon.awssdk:bom:2.10.73')  
    implementation platform('com.amazonaws:aws-xray-recorder-sdk-bom:2.4.0')  
    implementation 'software.amazon.awssdk:lambda'  
    implementation 'com.amazonaws:aws-xray-recorder-sdk-core'  
    implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-core'  
    implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-v2'  
    implementation 'com.amazonaws:aws-xray-recorder-sdk-aws-sdk-v2-instrumentor'  
    ...  
}
```

O exemplo a seguir mostra um rastreamento com 2 segmentos. Ambos são chamados my-function, mas um é do tipo AWS::Lambda e o outro é AWS::Lambda::Function. O segmento de função é expandido para mostrar seus subsegmentos.



O primeiro segmento representa a solicitação de invocação processada pelo serviço do Lambda. O segundo segmento registra o trabalho realizado pela sua função. O segmento de função tem 3 subsegmentos.

- Inicialização: representa o tempo gasto carregando a função e executando o [código de inicialização \(p. 30\)](#). Esse subsegmento só aparece para o primeiro evento processado por cada instância da função.
- Invocação: representa o trabalho realizado pelo código do handler. Ao instrumentar o código, é possível estender esse subsegmento com subsegmentos adicionais.
- Sobrecarga: representa o trabalho realizado pelo tempo de execução do Lambda como preparação para lidar com o próximo evento.

Você também pode instrumentar clientes HTTP, registrar consultas SQL e criar subsegmentos personalizados com anotações e metadados. Para obter mais informações, consulte [AWS X-Ray SDK for Java](#) no Guia do desenvolvedor do AWS X-Ray.

Seções

- [Habilitar o rastreamento ativo com a API do Lambda \(p. 625\)](#)
- [Habilitar o rastreamento ativo com o AWS CloudFormation \(p. 626\)](#)
- [Armazenar dependências de tempo de execução em uma camada \(p. 626\)](#)
- [Rastreamento em aplicativos de exemplo \(p. 627\)](#)

Habilitar o rastreamento ativo com a API do Lambda

Para gerenciar a configuração de rastreamento com a AWS CLI ou com o AWS SDK, use as seguintes operações de API:

- [UpdateFunctionConfiguration \(p. 974\)](#)
- [GetFunctionConfiguration \(p. 851\)](#)
- [CreateFunction \(p. 796\)](#)

O exemplo de comando da AWS CLI a seguir habilita o rastreamento ativo em uma função chamada my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

O modo de rastreamento faz parte da configuração específica da versão que é bloqueada quando você publica uma versão da função. Não é possível alterar o modo de rastreamento em uma versão publicada.

Habilitar o rastreamento ativo com o AWS CloudFormation

Para habilitar o rastreamento ativo em um recurso `AWS::Lambda::Function` em um modelo do AWS CloudFormation, use a propriedade `TracingConfig`.

Example [function-inline.yml](#): configuração de rastreamento

```
Resources:  
  function:  
    Type: AWS::Lambda::Function  
    Properties:  
      TracingConfig:  
        Mode: Active  
      ...
```

Para um recurso do AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, use a propriedade `Tracing`.

Example [template.yml](#): configuração de rastreamento

```
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      Tracing: Active  
      ...
```

Armazenar dependências de tempo de execução em uma camada

Se você usar o X-Ray SDK para instrumentar os clientes do AWS SDK com seu código de função, seu pacote de implantação poderá se tornar bastante grande. Para evitar o upload de dependências de tempo de execução sempre que você atualizar seu código de funções, empacote-as em uma [camada do Lambda \(p. 85\)](#).

O exemplo a seguir mostra um recurso `AWS::Serverless::LayerVersion` que armazena o SDK for Java e o X-Ray SDK for Java.

Example [template.yml](#): camada de dependências

```
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:
- java8

Com essa configuração, você só atualizará a camada de biblioteca se alterar as dependências de tempo de execução. O pacote de implantação de função contém apenas o código. Ao atualizar o código de função, o tempo de upload será muito mais rápido do que se você incluir dependências no pacote de implantação.

A criação de uma camada de dependências requer alterações de configuração de compilação para gerar o arquivo de camada antes da implantação. Para obter um exemplo funcional, consulte o aplicativo de exemplo [java-basic](#).

Rastreamento em aplicativos de exemplo

O repositório do GitHub para este guia inclui aplicativos de exemplo que demonstram o uso do rastreamento. Cada aplicativo de exemplo inclui scripts para fácil implantação e limpeza, um modelo do AWS SAM e recursos de suporte.

Aplicações de exemplo do Lambda em Java

- [blank-java](#): uma função Java que mostra o uso das bibliotecas Java do Lambda, registro em log, variáveis de ambiente, camadas, rastreamento do AWS X-Ray, testes de unidade e do AWS SDK.
- [java-basic](#): uma função Java mínima com testes de unidade e configuração de registro em log variável.
- [java-events](#)— Uma função Java mínima que usa a versão mais recente (3.0.0 e mais recente) do[aws-lambda-java-events](#) ([p. 599](#))biblioteca. Estes exemplos não exigem oAWSSDK como dependência.
- [s3-java](#): uma função Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.

Todas as aplicações de exemplo têm rastreamento ativo habilitado para funções do Lambda. A aplicação [blank-java](#) mostra instrumentação automática de clientes AWS SDK for Java 2.x, gerenciamento de segmentos para testes, subsegmentos personalizados e o uso de camadas do Lambda para armazenar dependências de tempo de execução.



Este exemplo da aplicação [blank-java](#) mostra os nós de serviço do Lambda, uma função e a API do Lambda. A função chama a API do Lambda para monitorar o uso do armazenamento no Lambda.

Criar um pacote de implantação usando o Eclipse

Esta seção mostra como empacotar o código Java em um pacote de implantação usando o plugin Eclipse IDE e Maven para Eclipse.

Note

O AWS SDK Eclipse Toolkit fornece um plugin para você criar um pacote de implantação e fazer upload para criar uma função do Lambda. Se você pode usar o Eclipse IDE como seu ambiente de desenvolvimento, esse plugin permite-lhe criar códigos em Java, criar e fazer upload de um pacote de implantação e criar sua função do Lambda. Para obter mais informações, consulte o [AWS Toolkit for EclipseGuia de conceitos básicos do](#). Para obter um exemplo do uso do toolkit para a criação de funções do Lambda, consulte [Using Lambda with the AWS toolkit for Eclipse](#).

Tópicos

- [Prerequisites \(p. 628\)](#)
- [Criar e compilar um projeto \(p. 628\)](#)

Prerequisites

Instale o plugin do Maven para Eclipse.

1. Inicie o Eclipse. No menu Help no Eclipse, escolha Install New Software.
2. Na janela Install, digite `http://download.eclipse.org/technology/m2e/releases` na caixa Work with: e selecione Add.
3. Siga as etapas para concluir a instalação.

Criar e compilar um projeto

Nesta etapa, você inicia o Eclipse e cria um projeto em Maven. Você adicionará as dependências necessárias e criará o projeto. O build produzirá um .jar, que é o pacote de implantação.

1. Crie um novo projeto do Maven no Eclipse.
 - a. No menu File, escolha New e, em seguida, Project.
 - b. Na janela New Project, escolha Maven Project.
 - c. Na janela New Maven Project, escolha Create a simple project e deixe as outras seleções padrão.
 - d. Nas janelas New Maven Project, Configure project, digite as seguintes informações para Artifact:
 - Group Id: doc-examples
 - Artifact Id: lambda-java-example
 - Version: 0.0.1-SNAPSHOT
 - Packaging: jar
 - Name: lambda-java-example
2. Adicione a dependência `aws-lambda-java-core` ao arquivo `pom.xml`.

Isso fornece definições das interfaces `RequestHandler`, `RequestStreamHandler` e `Context`. Isso permite que você compile o código que pode usar com o AWS Lambda.

- a. Abra o menu de contexto (clique com o botão direito do mouse) do arquivo `pom.xml`, escolha Maven e escolha Add Dependency.

- b. Na janela Add Dependency, digite os seguintes valores:

Group Id: com.amazonaws

Artifact Id: aws-lambda-java-core

Versão: 1.2.1

Note

Se você estiver seguindo outros tópicos do tutorial neste guia, os tutoriais específicos podem exigir que você adicione mais dependências. Adicione essas dependências conforme o necessário.

3. Adicione a classe Java ao projeto.

- a. Abra o menu de contexto (clique com o botão direito do mouse) para o subdiretório `src/main/java` no projeto, escolha New e, em seguida, escolha Class.
- b. Na janela New Java Class, digite os seguintes valores:

- Pacote: **example**
- Nome: **Hello**

Note

Se você estiver seguindo outros tópicos de tutorial neste guia, os tutoriais específicos podem recomendar diferentes nomes de pacote ou de classe.

- c. Adicione seu código Java. Se você estiver seguindo outros tópicos de tutorial neste guia, adicione o código fornecido.

4. Crie o projeto.

Abra o menu de contexto (clique com o botão direito do mouse) para o projeto em Package Explorer, escolha Run As e escolha Maven Build.... Na janela Edit Configuration (Editar configuração), digite **package** na caixa Goals (Metas).

Note

O .jar resultante, `lambda-java-example-0.0.1-SNAPSHOT.jar`, não é o .jar final independente que você pode usar como pacote de implantação. Na próxima etapa, você adicionará o Apache maven-shade-plugin para criar o .jar independente. Para obter mais informações, acesse [Apache Maven Shade plugin](#).

5. Adicione o plugin maven-shade-plugin e refaça o build.

O maven-shade-plugin pegará os artefatos (jars) produzidos pela meta do pacote (produz o .jar com o código do cliente) e criará um .jar independente que contém o código do cliente compilado e as dependências resolvidas do `pom.xml`.

- a. Abra o menu de contexto (clique com o botão direito do mouse) do arquivo `pom.xml`, escolha Maven e escolha Add Plugin.
- b. Na janela Add Plugin, digite os seguintes valores:

- Group Id: org.apache.maven.plugins
- Artifact Id: maven-shade-plugin
- Versão: 3.2.2

- c. Agora faça o build novamente.

Desta vez, criaremos o jar como antes e, em seguida, usaremos o `maven-shade-plugin` para extrair dependências para tornar o .jar independente

- i. Abra o menu de contexto (clique com o botão direito do mouse) para o projeto Run As e, em seguida, selecione Maven build....
- ii. Nas janelas Edit Configuration (Editar configuração), digite **package shade:shade** na caixa Goals (Metas).
- iii. Escolha Run.

Você encontrará o .jar independente resultante (ou seja, seu pacote de implantação) no subdiretório `/target` .

Abra o menu de contexto (clique com o botão direito do mouse) do subdiretório `/target`, escolha Show In, escolha System Explorer e você encontrará o `lambda-java-example-0.0.1-SNAPSHOT.jar`.

Aplicativos de exemplo Java para o AWS Lambda

O repositório do GitHub para este guia inclui aplicativos de exemplo que demonstram o uso do Java no AWS Lambda. Cada aplicativo de exemplo inclui scripts para fácil implantação e limpeza, um modelo do AWS CloudFormation e recursos de suporte.

Aplicações de exemplo do Lambda em Java

- [blank-java](#): uma função Java que mostra o uso das bibliotecas Java do Lambda, registro em log, variáveis de ambiente, camadas, rastreamento do AWS X-Ray, testes de unidade e do AWS SDK.
- [java-basic](#): uma função Java mínima com testes de unidade e configuração de registro em log variável.
- [java-events](#)— Uma função Java mínima que usa a versão mais recente (3.0.0 e mais recente) do[aws-lambda-java-events](#) (p. 599)biblioteca. Estes exemplos não exigem oAWSSDK como dependência.
- [s3-java](#): uma função Java que processa eventos de notificação do Amazon S3 e usa a Java Class Library (JCL) para criar miniaturas de arquivos de imagem enviados por upload.

Use o aplicativo de exemplo `blank-java` para aprender o básico ou como ponto de partida para seu próprio aplicativo. Ele mostra o uso das bibliotecas Java do Lambda, das variáveis de ambiente, do AWS SDK e do AWS X-Ray SDK. Ele usa uma camada do Lambda para empacotar suas dependências separadamente do código de função, o que acelera os tempos de implantação quando você está iterando em seu código de função. O projeto requer configuração mínima e pode ser implantado a partir da linha de comando em menos de um minuto.



As outras aplicações de exemplo mostram outras configurações de compilação, interfaces do manipulador e casos de uso para serviços que se integram com o Lambda. O `java-basic` de exemplo mostra uma função com dependências mínimas. É possível usar esse exemplo para casos em que você não precisa de bibliotecas adicionais, como o AWS SDK, e pode representar a entrada e a saída da função com tipos de Java padrão. Para tentar um tipo de manipulador diferente, você pode simplesmente alterar a configuração do manipulador na função.

Example [java-basic/src/main/java/example/HandlerStream.java](#) – manipulador de transmissões

```
// Handler value: example.HandlerStream
public class HandlerStream implements RequestStreamHandler {
```

```
Gson gson = new GsonBuilder().setPrettyPrinting().create();
@Override
public void handleRequest(InputStream inputStream, OutputStream outputStream, Context
context) throws IOException
{
    LambdaLogger logger = context.getLogger();
    BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream,
Charset.forName("US-ASCII")));
    PrintWriter writer = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(outputStream, Charset.forName("US-ASCII"))));
    try
    {
        HashMap<String, String> event = gson.fromJson(reader, HashMap.class);
        logger.log("STREAM TYPE: " + inputStream.getClass().toString());
        logger.log("EVENT TYPE: " + event.getClass().toString());
        writer.write(gson.toJson(event));
    ...
}
```

Os exemplos `java-events/src/main/java/example/HandlerDynamoDB.java` mostram o uso dos tipos de evento fornecidos pela biblioteca `aws-lambda-java-events`. Esses tipos representam os documentos de evento que os [serviços da AWS \(p. 277\)](#) enviam para a função. O `java-events` inclui manipuladores para tipos que não exigem dependências adicionais.

Example [java-events/src/main/java/example/HandlerDynamoDB.java](#) – registros do DynamoDB

```
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;
...
// Handler value: example.HandlerDynamoDB
public class HandlerDynamoDB implements RequestHandler<DynamodbEvent, String>{
    private static final Logger logger = LoggerFactory.getLogger(HandlerDynamoDB.class);
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    @Override
    public String handleRequest(DynamodbEvent event, Context context)
    {
        String response = new String("200 OK");
        for (DynamodbStreamRecord record : event.getRecords()){
            logger.info(record.getEventID());
            logger.info(record.geteventName());
            logger.info(record.getDynamodb().toString());
        }
    ...
}
```

Para obter mais destaque, consulte os outros tópicos neste capítulo.

Criar funções do Lambda com Go

As seções a seguir explicam como padrões comuns de programação e conceitos fundamentais são aplicados ao criar o código de função do Lambda em [Go](#).

Tempos de execução do Go

Nome	Identifier	Sistema operacional
Go 1.x	go1.x	Amazon Linux

O Lambda fornece as seguintes ferramentas e bibliotecas para o tempo de execução do Go:

Ferramentas e bibliotecas para Go

- [AWS SDK for Go](#): o SDK oficial da AWS para a linguagem de programação Go.
- [github.com/aws/aws-lambda-go/lambda](#): a implementação do modelo de programação do Lambda para Go. Esse pacote é usado pelo AWS Lambda para invocar o [Handler \(p. 634\)](#).
- [github.com/aws/aws-lambda-go/lambdacontext](#): auxiliares para acesso a informações do [objeto de contexto \(p. 638\)](#).
- [github.com/aws/aws-lambda-go/events](#): esta biblioteca fornece definições de tipos para integrações comuns de origens de eventos.
- [github.com/aws/aws-lambda-go/cmd/build-lambda-zip](#): Esta ferramenta pode ser usada para criar um archive com arquivo .zip no Windows.

Para obter mais informações, consulte [aws-lambda-go](#) no GitHub.

O Lambda fornece as seguintes aplicações de exemplo para o tempo de execução do Go:

Aplicativos do Lambda de exemplo do em Go

- [blank-go](#): uma função que mostra o uso das bibliotecas do Go do Lambda, o registro em log, as variáveis de ambiente e o AWS SDK.

Tópicos

- [AWS LambdaManipulador de função do em Go \(p. 634\)](#)
- [AWS LambdaObjeto de contexto do em Go \(p. 638\)](#)
- [Implantar funções do Lambda em Go com arquivos .zip \(p. 640\)](#)
- [Implantar funções do Lambda em Go com imagens de contêiner \(p. 643\)](#)
- [AWS LambdaRegistro em log da função do em Go \(p. 647\)](#)
- [AWS LambdaErros da função do em Go \(p. 652\)](#)
- [Instrumentação do código Go no AWS Lambda \(p. 656\)](#)
- [Usar variáveis de ambiente do \(p. 660\)](#)

AWS LambdaManipulador de função do em Go

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. Quando o manipulador é encerrado ou retorna uma resposta, ele se torna disponível para tratar de outro evento.

Uma função do Lambda escrita em [Go](#) é criada como um executável do Go. No código da função do Lambda, você precisa incluir o pacote github.com/aws/aws-lambda-go/lambda que implementa o modelo de programação do Lambda para Go. Além disso, você precisa implementar o código da função do manipulador e uma função `main()`.

```
package main

import (
    "fmt"
    "context"
    "github.com/aws/aws-lambda-go/lambda"
)

type MyEvent struct {
    Name string `json:"name"`
}

func HandleRequest(ctx context.Context, name MyEvent) (string, error) {
    return fmt.Sprintf("Hello %s!", name.Name), nil
}

func main() {
    lambda.Start(HandleRequest)
}
```

Observe o seguinte:

- pacote `main`: no Go, o pacote que contém `func main()` sempre deve ser denominado `main`.
- `import`: use essa opção para incluir as bibliotecas necessárias para a função do Lambda. Neste caso, ela inclui:
 - `context`: [AWS LambdaObjeto de contexto do em Go \(p. 638\)](#).
 - `fmt`: o objeto de [formatação](#) do Go usado para formatar o valor de retorno da função.
 - `github.com/aws/aws-lambda-go/lambda`: conforme mencionado anteriormente, implementa o modelo de programação do Lambda para Go.
- `func HandleRequest(ctx context.Context, name MyEvent) (string, error)`: esta é a assinatura do manipulador do Lambda e inclui o código a ser executado. Além disso, os parâmetros incluídos indicam o seguinte:
 - `ctx context.Context`: fornece informações do tempo de execução da invocação da função do Lambda. `ctx` é uma variável que você declara para utilizar as informações disponíveis por meio do [AWS LambdaObjeto de contexto do em Go \(p. 638\)](#).
 - `name MyEvent`: Um tipo de entrada com um nome de variável de `name` cujo valor será retornado na instrução `return`.
 - `string, error`: retorna dois valores: `string` de êxito e informação de `erro` padrão. Para obter mais informações sobre como manipular erros personalizados, consulte [AWS LambdaErros da função do em Go \(p. 652\)](#).
 - `return fmt.Sprintf("Hello %s!", name), nil`: simplesmente retorna uma saudação "Hello" (Olá) formatada com o nome que você forneceu no evento de entrada. `nil` indica que não houve erros e que a função foi executada com êxito.
- `func main()`: o ponto de entrada que executa o código da função do Lambda. Isso é obrigatório.

Com a adição de `lambda.Start(HandleRequest)` entre colchetes de código `func main() {}`, a função do Lambda será executada. De acordo com os padrões da linguagem Go, colchete de abertura `{` deve ser colocado diretamente no fim da assinatura da função `main`.

Manipulador de função do Lambda usando tipos estruturados

No exemplo acima, o tipo de entrada era uma sequência simples. Mas você também pode passar eventos estruturados para o manipulador da função:

```
package main

import (
    "fmt"
    "github.com/aws/aws-lambda-go/lambda"
)

type MyEvent struct {
    Name string `json:"What is your name?"`
    Age int     `json:"How old are you?"`
}

type MyResponse struct {
    Message string `json:"Answer:"`
}

func HandleLambdaEvent(event MyEvent) (MyResponse, error) {
    return MyResponse{Message: fmt.Sprintf("%s is %d years old!", event.Name,
event.Age)}, nil
}

func main() {
    lambda.Start(HandleLambdaEvent)
}
```

A solicitação deverá ser semelhante a esta:

```
# request
{
    "What is your name?": "Jim",
    "How old are you?": 33
}
```

E a resposta deverá ser semelhante a esta:

```
# response
{
    "Answer": "Jim is 33 years old!"
}
```

Para serem exportados, os nomes do campo no struct do evento devem estar em letra maiúscula. Para obter mais informações sobre a manipulação de eventos de fontes de eventos da AWS, consulte [aws-lambda-go/events](#).

Assinaturas válidas de manipulador

Há várias opções ao criar um manipulador de função do Lambda no Go, mas você deve observar as seguintes regras:

- O manipulador deve ser uma função.
- O manipulador pode usar de 0 a 2 argumentos. Se houver dois argumentos, o primeiro argumento deverá implementar `context.Context`.
- O manipulador pode retornar de 0 a 2 argumentos. Se houver um único valor de retorno, ele deverá implementar `error`. Se houver dois valores de retorno, o segundo valor deverá implementar `error`. Para obter mais informações sobre como implementar informações de manipulação de erros, consulte [AWS Lambda Erros da função do em Go \(p. 652\)](#).

As assinaturas válidas de manipulador são listadas a seguir. `TIn` e `TOut` representam tipos compatíveis com a biblioteca `encoding/json` padrão. Para obter mais informações, consulte [func Unmarshal](#) para saber como esses tipos são desserializados.

- `func ()`
- `func () error`
- `func (TIn) error`
- `func () (TOut, error)`
- `func (context.Context) error`
- `func (context.Context, TIn) error`
- `func (context.Context) (TOut, error)`
- `func (context.Context, TIn) (TOut, error)`

Usar o estado global

Você pode declarar e modificar variáveis globais que são independentes do código do manipulador da função do Lambda. Além disso, o manipulador pode declarar uma função `init` que é executada quando o manipulador é carregado. Esse comportamento é o mesmo no AWS Lambda e nos programas Go padrão. Uma única instância de sua função do Lambda nunca manipulará vários eventos simultaneamente.

```
package main

import (
    "log"
    "github.com/aws/aws-lambda-go/lambda"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
    "github.com/aws/aws-sdk-go/aws"
)

var invokeCount = 0
var myObjects []*s3.Object
func init() {
```

```
svc := s3.New(session.New())
input := &s3.ListObjectsV2Input{
    Bucket: aws.String("examplebucket"),
}
result, _ := svc.ListObjectsV2(input)
myObjects = result.Contents
}

func LambdaHandler() (int, error) {
    invokeCount = invokeCount + 1
    log.Print(myObjects)
    return invokeCount, nil
}

func main() {
    lambda.Start(LambdaHandler)
}
```

AWS LambdaObjeto de contexto do em Go

Quando o Lambda executa a função, ele transmite um objeto de contexto para o [manipulador \(p. 634\)](#). Esse objeto fornece métodos e propriedades com informações sobre a invocação, a função e o ambiente de execução.

A biblioteca de contexto do Lambda fornece as seguintes variáveis globais, métodos e propriedades.

Variáveis globais

- `FunctionName`: o nome da função do Lambda.
- `FunctionVersion`: a [versão \(p. 106\)](#) da função.
- `MemoryLimitInMB`: a quantidade de memória alocada para a função.
- `LogGroupName`: o grupo de logs da função.
- `LogStreamName`: a transmissão de log para a instância da função.

Métodos de contexto

- `Deadline`— Retorna a data em que a execução expira o tempo em milissegundos do Unix.

Propriedades de contexto

- `InvokedFunctionArn`: o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um alias ou número de versão.
- `AwsRequestId`: o identificador da solicitação de invocação.
- `Identity`: (aplicativos móveis) informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
- `ClientContext`: (aplicativos móveis) contexto do cliente fornecido ao Lambda pela aplicação cliente.

Acessar informações do contexto de invocação

As funções do Lambda têm acesso aos metadados sobre seu ambiente e a solicitação da invocação. Isso pode ser acessado no [Contexto do pacote](#). Se o manipulador incluir `context.Context` como um parâmetro, o Lambda inserirá informações sobre sua função na propriedade `Value` do contexto. Observe que você precisa importar a biblioteca `lambdacontext` para acessar o conteúdo do objeto `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)
```

```
}
```

No exemplo acima, `lc` é a variável usada para consumir as informações que o objeto de contexto capturou, e `log.Println(lc.Identity.CognitoIdentityPoolID)` imprime essas informações, neste caso, o CognitoIdentityPoolID.

O exemplo a seguir mostra como usar o objeto de contexto para monitorar o tempo necessário para concluir a função do Lambda. Isso permite que você analise as expectativas de performance e ajuste seu código de função de maneira correspondente, se necessário.

```
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.Println("hello!")
            time.Sleep(50 * time.Millisecond)
        }
    }
}

func main() {
    lambda.Start(LongRunningHandler)
}
```

Implantar funções do Lambda em Go com arquivos .zip

O código da função do AWS Lambda consiste em scripts ou programas compilados e as dependências deles. Você usa um pacote de implantação para implantar seu código de função no Lambda. O Lambda é compatível com dois tipos de pacotes de implantação: imagens de contêiner e arquivos .zip.

Esta página descreve como criar um arquivo .zip como seu pacote de implantação do runtime do Go e, em seguida, usar o arquivo para implantar o código de sua função no AWS Lambda utilizando o AWS Command Line Interface (AWS CLI).

Seções

- [Prerequisites \(p. 640\)](#)
- [Ferramentas e bibliotecas \(p. 640\)](#)
- [Aplicativos de exemplo \(p. 640\)](#)
- [Criando um arquivo .zip no macOS e no Linux \(p. 641\)](#)
- [Criando um arquivo .zip no Windows \(p. 641\)](#)

Prerequisites

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Ferramentas e bibliotecas

O Lambda fornece as seguintes ferramentas e bibliotecas para o tempo de execução do Go:

Ferramentas e bibliotecas para Go

- [AWS SDK for Go](#): o SDK oficial da AWS para a linguagem de programação Go.
- [github.com/aws/aws-lambda-go/lambda](#): a implementação do modelo de programação do Lambda para Go. Esse pacote é usado pelo AWS Lambda para invocar o [handler \(p. 634\)](#).
- [github.com/aws/aws-lambda-go/lambdacontext](#): auxiliares para acesso a informações do [objeto de contexto \(p. 638\)](#).
- [github.com/aws/aws-lambda-go/events](#): esta biblioteca fornece definições de tipos para integrações comuns de origens de eventos.
- [github.com/aws/aws-lambda-go/cmd/build-lambda-zip](#): Esta ferramenta pode ser usada para criar um archive com arquivo .zip no Windows.

Para obter mais informações, consulte [aws-lambda-go](#) no GitHub.

Aplicativos de exemplo

O Lambda fornece as seguintes aplicações de exemplo para o tempo de execução do Go:

Aplicativos do Lambda de exemplo do em Go

- [blank-go](#): uma função do Go que mostra o uso das bibliotecas do Go do Lambda, o registro em log, as variáveis de ambiente e o AWS SDK.

Criando um arquivo .zip no macOS e no Linux

As etapas a seguir demonstram como baixar a biblioteca [lambda](#) do GitHub com o `go get` e compilar seu executável com o `go build`.

1. Baixe a biblioteca lambda do GitHub.

```
go get github.com/aws/aws-lambda-go/lambda
```

2. Compile seu executável.

```
GOOS=linux go build main.go
```

Configurar o `GOOS` para o `linux` garante que o executável compilado seja compatível com o [tempo de execução do Go \(p. 214\)](#), mesmo se você compilá-lo em um ambiente não Linux.

3. (Opcional) Se o seu pacote `main` for composto de vários arquivos, use o comando `go build` a seguir para compilar o pacote:

```
GOOS=linux go build main
```

4. (Opcional) Talvez seja necessário compilar pacotes com o `CGO_ENABLED=0` configurado no Linux:

```
GOOS=linux CGO_ENABLED=0 go build main.go
```

Este comando cria um pacote binário estável para versões padrão da biblioteca C (`libc`), que podem ser diferentes no Lambda e em outros dispositivos.

5. O Lambda usa permissões de arquivo POSIX, então pode ser necessário [definir permissões para a pasta do pacote de implantação](#) antes de criar o arquivo `.zip`.
6. Crie um pacote de implantação empacotando o executável em um arquivo `.zip`.

```
zip function.zip main
```

Criando um arquivo .zip no Windows

As etapas a seguir demonstram como baixar a ferramenta `build-lambda-zip` para Windows a partir do GitHub com o `go get` e compilar seu executável com o `go build`.

Note

Se ainda não tiver feito isso, você precisará instalar o [Git](#) e, em seguida, adicionar o executável `git` à variável de ambiente `%PATH%` do Windows.

1. Baixe a ferramenta `build-lambda-zip` do GitHub:

```
go.exe get -u github.com/aws/aws-lambda-go/cmd/build-lambda-zip
```

2. Use a ferramenta do seu `GOPATH` para criar um arquivo `.zip`. Se você tiver uma instalação padrão do Go, a ferramenta geralmente estará em `%USERPROFILE%\Go\bin`. Caso contrário, navegue até o local em que o tempo de execução do Go foi instalado e faça o seguinte:

cmd.exe

Em cmd.exe, execute o seguinte:

```
set GOOS=linux
go build -o main main.go
%USERPROFILE%\Go\bin\build-lambda-zip.exe -output main.zip main
```

PowerShell

No PowerShell, execute o seguinte:

```
$env:GOOS = "linux"
$env:CGO_ENABLED = "0"
$env:GOARCH = "amd64"
go build -o main main.go
~\Go\Bin\build-lambda-zip.exe -output main.zip main
```

Implantar funções do Lambda em Go com imagens de contêiner

Você pode implantar seu código de função do Lambda como uma [imagem de contêiner \(p. 267\)](#). A AWS oferece os seguintes recursos para ajudar você a criar uma imagem de contêiner para sua função do Go:

- [AWSImagens base para o Lambda](#)

Essas imagens base são pré-carregadas com um tempo de execução de linguagem e outros componentes necessários para executar a imagem no Lambda. AWS fornece um Dockerfile para cada uma das imagens de base para ajudar a criar sua imagem de contêiner.

- [Clientes de interface de tempo de execução de](#)

Se você usar uma imagem de base de comunidade ou de empresa privada, adicione um cliente de interface de tempo de execução à imagem base para torná-lo compatível com o Lambda.

O fluxo de trabalho de uma função definida como uma imagem de contêiner inclui as seguintes etapas:

1. Crie sua imagem de contêiner usando os recursos listados neste tópico.
2. Faça upload da imagem em seu registro do contêiner do Amazon ECR. Consulte os passos 7 a 9 no [Criar imagem \(p. 268\)](#).
3. [Crie \(p. 91\)](#) a função do Lambda ou [atualize o código da função \(p. 92\)](#) para implantar a imagem em uma função existente.

Tópicos

- [AWSImagens de base da para Go \(p. 643\)](#)
- [Clientes de interface de tempo de execução do Go \(p. 643\)](#)
- [Usar a imagem base Go:1.x \(p. 644\)](#)
- [Criar uma imagem Go a partir de uma imagem básica do provided.al2 \(p. 644\)](#)
- [Criar uma imagem Go a partir de uma imagem básica alternativa \(p. 645\)](#)
- [Implantar a imagem do contêiner \(p. 646\)](#)

AWSImagens de base da para Go

AWSA oferece a seguinte imagem de base para Go:

Tags	Tempo de execução	Sistema operacional	Dockerfile
1	Go 1.x	Amazon Linux 2018.03	Dockerfile para Go 1.x no GitHub

Repositório do Docker Hub: [amazon/aws-lambda-go](#)

Repositório do Amazon ECR: [gallery.ecr.aws/lambda/go](#)

Clientes de interface de tempo de execução do Go

AWSA não oferece um cliente de interface de tempo de execução separado para Go. O pacote `aws-lambda-go/lambda` inclui uma implementação da interface de tempo de execução.

Usar a imagem base Go:1.x

Para obter instruções sobre como usar a imagem base para Go:1.x, escolha a guia [usage \(uso\)](#) nas [imagens base do Lambda para Go](#) no repositório do Amazon ECR.

As instruções também estão disponíveis em [imagens base do Lambda para Go](#) no repositório do Docker Hub.

Criar uma imagem Go a partir de uma imagem básica do `provided.al2`

Para criar uma imagem de contêiner para Go que seja executada no Amazon Linux 2, use a imagem de base `provided.al2`. Para obter mais informações sobre esta imagem de base, consulte [fornecido](#) na galeria pública do Amazon ECR.

Você inclui o pacote `aws-lambda-go/lambda` com seu manipulador do Go. Este pacote implementa o modelo de programação para Go, incluindo o cliente de interface de tempo de execução. A imagem `provided.al2` de base também inclui o emulador de interface de tempo de execução.

Para criar e implantar uma função do Go com a imagem de base **`provided.al2`**.

Observe que as três primeiras etapas são idênticas se você implantar sua função como um arquivo de arquivo.zip ou como uma imagem de contêiner.

1. Em sua máquina local, crie um diretório de projeto para sua nova função.
2. Na pasta do projeto, execute o comando a seguir para instalar as bibliotecas do Lambda em Go necessárias.

```
go get github.com/aws/aws-lambda-go
```

Para obter uma descrição das bibliotecas do Lambda em Go, consulte [Criar funções do Lambda com o Go \(p. 633\)](#).

3. Crie seu [código de manipulador do Go \(p. 634\)](#) e inclua o pacote `aws-lambda-go/lambda`.
4. Use um editor de texto para criar um Dockerfile no diretório do projeto. O exemplo de Dockerfile a seguir usa a imagem de base AWS da `provided.al2`.

```
FROM public.ecr.aws/lambda/provided:al2 as build
# install compiler
RUN yum install -y golang
RUN go env -w GOPROXY=direct
# cache dependencies
ADD go.mod go.sum .
RUN go mod download
# build
ADD ..
RUN go build -o /main
# copy artifacts to a clean image
FROM public.ecr.aws/lambda/provided:al2
COPY --from=build /main /main
ENTRYPOINT [ "/main" ]
```

5. Crie sua imagem do Docker com o comando `docker build`. Insira um nome para a imagem. O exemplo a seguir nomeia a imagem como `hello-world`.

```
docker build -t hello-world .
```

6. Autentique a CLI do Docker no seu registro do Amazon ECR.

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-  
stdin 123456789012.dkr.ecr.us-east-1.amazonaws.com
```

7. Marque sua imagem para corresponder ao nome do repositório e implante a imagem no Amazon ECR usar o comando docker push.

```
docker tag hello-world:latest 123456789012.dkr.ecr.us-east-1.amazonaws.com/hello-  
world:latest  
docker push 123456789012.dkr.ecr.us-east-1.amazonaws.com/hello-world:latest
```

Agora que a imagem de contêiner reside no registro do contêiner do Amazon ECR, você pode criar ([p. 90](#)) a função do Lambda e implantar a imagem.

Criar uma imagem Go a partir de uma imagem básica alternativa

Você pode criar uma imagem de contêiner para Go a partir de uma imagem de base alternativa. O Dockerfile de exemplo a seguir usa [alpine](#) como a imagem de base.

```
FROM alpine as build  
# install build tools  
RUN apk add go git  
RUN go env -w GOPROXY=direct  
# cache dependencies  
ADD go.mod go.sum ./  
RUN go mod download  
# build  
ADD . .  
RUN go build -o /main  
# copy artifacts to a clean image  
FROM alpine  
COPY --from=build /main /main  
ENTRYPOINT [ "/main" ]
```

As etapas são as mesmas descritas para uma imagem de base provided.al2, com uma consideração adicional: se você quiser adicionar o RIE à sua imagem, será necessário seguir estas etapas adicionais antes de executar o comando docker build. Para obter mais informações sobre como testar sua imagem localmente com o RIE, consulte [the section called “Testar imagens” \(p. 274\)](#).

Para adicionar o RIE à imagem

1. No Dockerfile, substitua a instrução ENTRYPOINT pelo seguinte conteúdo:

```
# (Optional) Add Lambda Runtime Interface Emulator and use a script in the ENTRYPOINT  
for simpler local runs  
ADD https://github.com/aws/aws-lambda-runtime-interface-emulator/releases/latest/  
download/aws-lambda-rie /usr/bin/aws-lambda-rie  
RUN chmod 755 /usr/bin/aws-lambda-rie  
COPY entry.sh /  
RUN chmod 755 /entry.sh  
ENTRYPOINT [ "/entry.sh" ]
```

2. Use um editor de texto para criar arquivo entry.sh no diretório do projeto, com o seguinte conteúdo:

```
#!/bin/sh
```

```
if [ -z "${AWS_LAMBDA_RUNTIME_API}" ]; then
    exec /usr/bin/aws-lambda-rie "$@"
else
    exec "$@"
fi
```

Se não quiser adicionar o RIE à sua imagem, você poderá testá-la localmente sem realizar essa ação.

Para testar localmente sem adicionar RIE à imagem

1. No diretório do projeto, execute o comando a seguir para fazer download do RIE do GitHub e instalá-lo em sua máquina local.

```
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
```

2. Execute sua função de imagem do Lambda usando o comando `docker run`. No exemplo a seguir, `/main` é o caminho para o ponto de entrada da função.

```
docker run -d -v ~/.aws-lambda-rie:/aws-lambda --entrypoint /aws-lambda/aws-lambda-rie
-p 9000:8080 myfunction:latest /main
```

Isso executa a imagem como um contêiner e inicia um endpoint localmente em `localhost:9000/2015-03-31/functions/function/invocations`.

3. Publique um evento no seguinte endpoint usando um comando `curl`:

```
curl -XPOST "http://localhost:9000/2015-03-31/functions/function/invocations" -d '{}'
```

Este comando invoca a função do em execução na imagem do contêiner e retorna uma resposta.

Agora que a imagem de contêiner reside no registro do contêiner do Amazon ECR, você pode [criar \(p. 90\)](#) a função do Lambda e implantar a imagem.

Implantar a imagem do contêiner

Para uma nova função, você implanta a imagem Go ao [criar a função \(p. 91\)](#). Para uma função existente, se você reconstruir a imagem do contêiner, precisará reimplantar a imagem ao [atualizar o código da função \(p. 92\)](#).

AWS Lambda Registro em log da função do em Go

O AWS Lambda monitora automaticamente as funções do Lambda em seu nome e envia métricas da função para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente do tempo de execução do Lambda envia detalhes sobre cada invocação à transmissão de logs e transmite os logs e outras saídas do código de sua função.

Esta página descreve como produzir a saída de logs usando o código de sua função do Lambda ou acessar os logs usando a AWS Command Line Interface, o console do Lambda ou o console do CloudWatch.

Seções

- [Criar uma função que retorna logs \(p. 647\)](#)
- [Usar o console do Lambda \(p. 648\)](#)
- [Usando o console do CloudWatch \(p. 648\)](#)
- [Usar a AWS Command Line Interface \(AWS CLI\) \(p. 649\)](#)
- [Excluir logs \(p. 651\)](#)

Criar uma função que retorna logs

Para gerar os logs do código de função, você pode usar métodos no [pacote fmt](#) ou qualquer biblioteca de logs que grave em `stdout` ou em `stderr`. O exemplo a seguir usa [o pacote de logs](#).

Example `main.go`: registro em log

```
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 do 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/  
[$LATEST]569cxmplc3c34c7489e6a97ad08b4419  
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_NAME=blank-go-function-9DV3XMP6XBC  
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_MEMORY_SIZE=128  
2020/03/27 03:40:05 AWS_LAMBDA_FUNCTION_VERSION=$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:  
39 ms Memory Size: 128 MB Max Memory Used: 54 MB Init Duration: 203.69 ms  
XRAY TraceId: 1-5e7d7595-212fxmpl9ee07c4884191322 SegmentId: 42ffxmpl0645f474 Sampled: true
```

O tempo de execução do Go registra em log as linhas **START**, **END** e **REPORT** para cada invocação. A linha do relatório fornece os seguintes detalhes.

Registro em log de relatório

- **RequestId**: o ID de solicitação exclusivo para a invocação.
- **Duração**: a quantidade de tempo que o método de manipulador da função gastou processando o evento.
- **Duração faturada**: a quantia de tempo faturada para a invocação.
- **Tamanho da memória**: a quantidade de memória alocada para a função.
- **Memória máxima utilizada**: a quantidade de memória utilizada pela função.
- **Duração inicial**: para a primeira solicitação atendida, a quantidade de tempo que o tempo de execução levou para carregar a função e executar o código fora do método do manipulador.
- **XRAY Traceld**: para solicitações rastreadas, o [ID de rastreamento do AWS X-Ray \(p. 477\)](#).
- **SegmentId**: para solicitações rastreadas, o ID do segmento do X-Ray.
- **Amostragem**: para solicitações rastreadas, o resultado da amostragem.

Usar o console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda. Para obter mais informações, consulte [Acessar o Amazon CloudWatch Logs para o AWS Lambda \(p. 720\)](#).

Usando o console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a [página Log groups](#) (Grupos de log).
2. Escolha o grupo de logs de sua função (`/aws/lambda/nome-de-sua-função`).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma [instância da sua função \(p. 219\)](#). Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos instrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

Para usar uma aplicação de exemplo que correlaciona os logs e os rastreamentos com o X-Ray, consulte [Aplicativo de exemplo de processador de erros para o AWS Lambda \(p. 500\)](#).

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Você pode usar a [AWS CLI](#) para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Você deve ver a saída a seguir:

```
{  
    "StatusCode": 200,  
    "LogResult":  
        "U1RBULQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",  
    "ExecutedVersion": "$LATEST"  
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \  
--query 'LogResult' --output text | base64 -d
```

Você deve ver a saída a seguir:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e [Ubuntu no Windows](#). Os usuários do macOS precisam usar `base64 -D`.

Example get-logs.sh script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa `sed` para remover as aspas do arquivo de saída e fica inativo por 15 segundos para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como `get-logs.sh`.

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

```
#!/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 $(cat
out) --limit 5
```

Example macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod +R 755 get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

Você deve ver a saída a seguir:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version: $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf \tINFO\tENVIRONMENT VARIABLES\r{\r\t\t\"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\", \r\t\t...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf \tINFO\tEVENT\r{\r\t\t\"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"
}
```

Excluir logs

Os grupos de logs não são excluídos automaticamente quando você exclui uma função. Para evitar armazenar logs indefinidamente, exclua o grupo de logs ou[Configurar um período de retenção](#)após o qual os logs são excluídos automaticamente.

AWS Lambda Erros da função do em Go

Quando o código gera um erro, o Lambda gera uma representação JSON do erro. Esse documento de erro aparece no log de invocação e, para invocações síncronas, na saída.

Esta página descreve como exibir erros de invocação de função do Lambda para o tempo de execução do Go usando o console do Lambda e a AWS CLI.

Seções

- [Criar uma função que retorna exceções \(p. 652\)](#)
- [Como funcionam \(p. 652\)](#)
- [Usar o console do Lambda \(p. 653\)](#)
- [Usar a AWS Command Line Interface \(AWS CLI\) \(p. 653\)](#)
- [Tratamento de erros em outros serviços da AWS \(p. 654\)](#)
- [Próximas etapas \(p. 655\)](#)

Criar uma função que retorna exceções

O exemplo de código a seguir demonstra o tratamento de erros personalizado que gera uma exceção diretamente de uma função do Lambda e a manipula de maneira direta. Observe que os erros personalizados no Go devem importar o módulo `errors`.

```
package main

import (
    "errors"
    "github.com/aws/aws-lambda-go/lambda"
)

func OnlyErrors() error {
    return errors.New("something went wrong!")
}

func main() {
    lambda.Start(OnlyErrors)
}
```

O que retorna o seguinte:

```
{
    "errorMessage": "something went wrong!",
    "errorType": "errorString"
}
```

Como funcionam

Ao invocar uma função do Lambda, o Lambda recebe a solicitação de invocação e valida as permissões de sua função de execução, verifica se o documento do evento é um documento JSON válido e verifica valores de parâmetros.

Se a solicitação for validada, o Lambda a envia para uma instância da função. O ambiente de [tempo de execução do Lambda \(p. 214\)](#) converte o documento do evento em um objeto e o transmite ao handler da função.

Se o Lambda encontra um erro, ele retorna um tipo de exceção, uma mensagem e o código HTTP do status que indica a causa do erro. O cliente ou o serviço que invocou a função do Lambda pode processar o erro de maneira programática ou transmiti-lo a um usuário final. O comportamento correto de tratamento de erros depende do tipo de aplicativo, do público e da origem do erro.

A lista a seguir descreve o intervalo de códigos de status que você pode receber do Lambda.

2xx

Um erro da série 2xx com um cabeçalho `X-Amz-Function-Error` na resposta indica um erro de tempo de execução ou de função do Lambda. Um código de status da série 2xx indica que o Lambda aceitou a solicitação, mas em vez de um código de erro, o Lambda indica o erro incluindo o cabeçalho `X-Amz-Function-Error` na resposta.

4xx

Um erro da série 4xx indica um erro que o cliente ou serviço que fez a invocação pode corrigir modificando a solicitação, solicitando permissão ou tentando a solicitação novamente. Os erros da série 4xx diferentes de 429 geralmente indicam um erro na solicitação.

5xx

Um erro da série 5xx indica um problema com o Lambda ou com a configuração/recursos da função. Os erros da série 5xx podem indicar uma condição temporária que pode ser resolvida sem nenhuma ação do usuário. O cliente ou serviço que fez a invocação não podem solucionar esses problemas, mas o proprietário de uma função do Lambda pode ser capaz de corrigi-los.

Para obter uma lista completa de erros de invocação, consulte [Erros de InvokeFunction \(p. 877\)](#).

Usar o console do Lambda

Você pode invocar sua função no console do Lambda configurando um evento de teste e visualizando a saída. A saída é capturada pelos logs de execução da função e, quando o [rastreamento ativo \(p. 477\)](#) está habilitado, pelo AWS X-Ray.

Para invocar uma função no console do Lambda

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Test (Testar).
4. Selecione New event (Novo evento) e escolha um Event template (Modelo de evento) na lista suspensa.
5. Insira um nome para o evento de teste.
6. Digite o JSON para o evento de teste.
7. Escolha Create event (Criar evento).
8. Escolha Invoke (Invocar).

O console do Lambda invoca sua função [de forma síncrona \(p. 158\)](#) e exibe o resultado. Para ver a resposta, os logs e outras informações, expanda a seção Details (Detalhes).

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Quando você invoca uma função do Lambda na AWS CLI, a AWS CLI divide a resposta em dois documentos. A resposta da AWS CLI é exibida no prompt de comando. Se ocorreu um erro, a resposta contém um campo `FunctionError`. A resposta ou o erro de invocação retornado pela função é gravado no arquivo de saída. Por exemplo, o `output.json` ou o `output.txt`.

O exemplo de comando `invoke` a seguir demonstra como invocar uma função e escrever a resposta de invocação em um arquivo `output.txt`.

```
aws lambda invoke \
--function-name my-function \
--cli-binary-format raw-in-base64-out \
--payload '{"key1": "value1", "key2": "value2", "key3": "value3"}' output.txt
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

Você deve ver a resposta da AWS CLI em seu prompt de comando:

```
{
  "StatusCode": 200,
  "FunctionError": "Unhandled",
  "ExecutedVersion": "$LATEST"
}
```

Você deve ver a resposta de invocação de função no arquivo `output.txt`. Também é possível visualizar a saída no mesmo prompt de comando usando:

```
cat output.txt
```

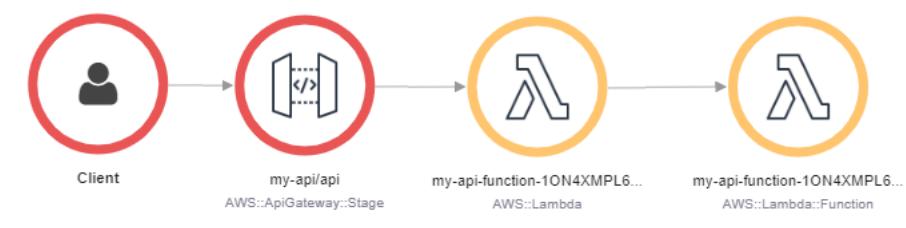
Você deve ver a resposta de invocação no prompt de comando.

Tratamento de erros em outros serviços da AWS

Quando outro serviço da AWS invoca sua função, o serviço escolhe o tipo de invocação e o comportamento de repetição. Os serviços da AWS podem invocar sua função em um agendamento, em resposta a um evento de ciclo de vida em um recurso ou para atender a uma solicitação de um usuário. Alguns serviços invocam funções de forma assíncrona e permitem que o Lambda trate erros, enquanto outros fazem novas tentativas ou transmitam os erros de volta ao usuário.

Por exemplo, o API Gateway trata todos os erros de invocação e função como erros internos. Se a API do Lambda rejeitar a solicitação de invocação, o API Gateway retornará um código de erro 500. Se a função é executada, mas retorna um erro ou retorna uma resposta no formato errado, o API Gateway retorna um código de erro 502. Para personalizar a resposta de erro, é necessário detectar erros no código e formatar uma resposta no formato necessário.

Recomendamos usar AWS X-Ray para determinar a fonte de um erro e a respectiva causa. O X-Ray permite localizar qual componente encontrou um erro e ver detalhes sobre os erros. O exemplo a seguir mostra um erro de função que resultou em uma resposta 502 do API Gateway.



Para obter mais informações, consulte [Instrumentação do código Go no AWS Lambda \(p. 656\)](#).

Próximas etapas

- Saiba como mostrar eventos de registro em log para sua função do Lambda na página [the section called “Registro em log” \(p. 647\)](#).

Instrumentação do código Go no AWS Lambda

O Lambda se integra ao AWS X-Ray para permitir que você rastreie, depure e otimize aplicativos do Lambda. É possível usar o X-Ray para rastrear uma solicitação à medida que ela atravessa recursos na aplicação, da API de frontend ao armazenamento e aos banco de dados no backend. Ao simplesmente adicionar a biblioteca do X-Ray SDK à configuração de compilação, é possível registrar erros e latência para qualquer chamada que a função faça para um serviço da AWS.

O X-Ray Mapa de serviço mostra o fluxo de solicitações através de seu aplicativo. O exemplo a seguir do aplicativo de exemplo [processador de erros \(p. 500\)](#) mostra um aplicativo com duas funções. A função principal processa eventos e, às vezes, retorna erros. A segunda função processa erros que aparecem no primeiro grupo de logs e usa o AWS SDK para chamar o X-Ray, o Amazon S3 e o Amazon CloudWatch Logs.



Para rastrear solicitações que não têm um cabeçalho de rastreamento, ative o rastreamento ativo na configuração da sua função.

Para ativar o rastreamento ativo

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Selecione **Configuração**, depois, escolha **Ferramentas de monitoramento**.
4. Selecione **Edit**.
5. Em X-Ray, habilite o **Active tracing (Rastreamento ativo)**.
6. Escolha **Save (Salvar)**.

Pricing

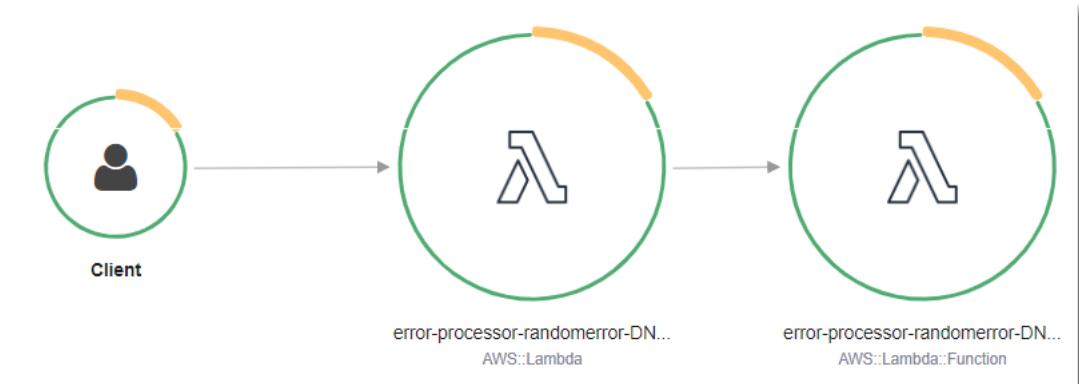
O X-Ray tem um nível gratuito vitalício. Além do limite do nível gratuito, o X-Ray cobra por armazenamento e recuperação de rastreamento. Para obter detalhes, consulte [Definição de preço do AWS X-Ray](#).

Sua função precisa de permissão para carregar dados de rastreamento no X-Ray. Quando você habilita o rastreamento ativo no console do Lambda, o Lambda adiciona as permissões necessárias à [função de](#)

execução (p. 57) da função. Caso contrário, adicione a política [AWSXRayDaemonWriteAccess](#) à função de execução.

O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento é eficiente, enquanto ainda fornece uma amostra representativa das solicitações que a sua aplicação serviu. A regra de amostragem padrão é uma solicitação por segundo e 5% de solicitações adicionais. Esta taxa de amostragem não pode ser configurada para funções do Lambda.

Quando o rastreamento ativo está habilitado, o Lambda registra um rastreamento para um subconjunto de invocações. Lambda registra doisSegmentos do, que cria dois nós no mapa de serviço. O primeiro nó representa o serviço Lambda que recebe a solicitação de invocação. O segundo nó é gravado pelo [tempo de execução \(p. 19\)](#)da função.



É possível instrumentar o código do manipulador para gravar metadados e rastrear chamadas downstream. Para registrar detalhes sobre chamadas feitas pelo manipulador para outros recursos e serviços, use o X-Ray SDK for Go. Faça download do SDK de seu [Repositório do GitHub](#) com go get:

```
go get github.com/aws/aws-xray-sdk-go
```

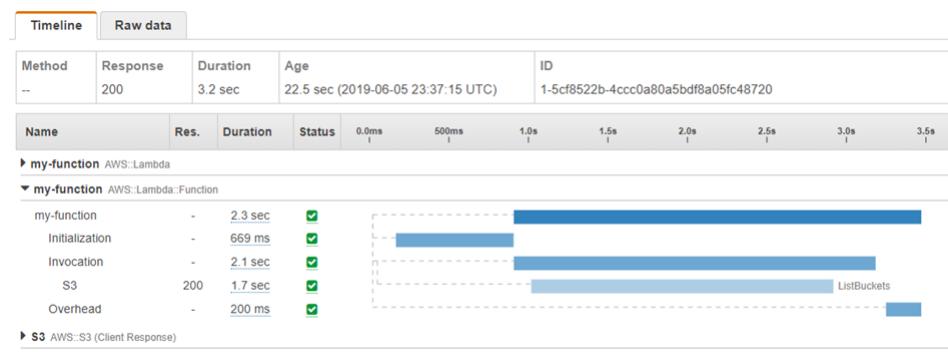
Para instrumentar clientes do AWS SDK, passe o cliente para o método `xray.AWS()`.

```
xray.AWS(s3.Client)
```

Em seguida, você pode rastrear suas chamadas usando a versão `WithContext` do método.

```
svc.ListBucketsWithContext(ctx aws.Context, input *ListBucketsInput)
```

O exemplo a seguir mostra um rastreamento com 2 segmentos. Ambos são chamados my-function, mas um é do tipo `AWS::Lambda` e o outro é `AWS::Lambda::Function`. O segmento de função é expandido para mostrar seus subsegmentos.



O primeiro segmento representa a solicitação de invocação processada pelo serviço do Lambda. O segundo segmento registra o trabalho realizado pela sua função. O segmento de função tem 3 subsegmentos.

- Inicialização: representa o tempo gasto carregando a função e executando o [código de inicialização \(p. 30\)](#). Esse subsegmento só aparece para o primeiro evento processado por cada instância da função.
- Invocação: representa o trabalho realizado pelo código do handler. Ao instrumentar o código, é possível estender esse subsegmento com subsegmentos adicionais.
- Sobrecarga: representa o trabalho realizado pelo tempo de execução do Lambda como preparação para lidar com o próximo evento.

Você também pode instrumentar clientes HTTP, registrar consultas SQL e criar subsegmentos personalizados com anotações e metadados. Para obter mais informações, consulte [The X-Ray SDK for Go](#) no Guia do desenvolvedor do AWS X-Ray.

Seções

- [Habilitar o rastreamento ativo com a API do Lambda \(p. 658\)](#)
- [Habilitar o rastreamento ativo com o AWS CloudFormation \(p. 658\)](#)

Habilitar o rastreamento ativo com a API do Lambda

Para gerenciar a configuração de rastreamento com a AWS CLI ou com o AWS SDK, use as seguintes operações de API:

- [UpdateFunctionConfiguration \(p. 974\)](#)
- [GetFunctionConfiguration \(p. 851\)](#)
- [CreateFunction \(p. 796\)](#)

O exemplo de comando da AWS CLI a seguir habilita o rastreamento ativo em uma função chamada my-function.

```
aws lambda update-function-configuration --function-name my-function \
--tracing-config Mode=Active
```

O modo de rastreamento faz parte da configuração específica da versão que é bloqueada quando você publica uma versão da função. Não é possível alterar o modo de rastreamento em uma versão publicada.

Habilitar o rastreamento ativo com o AWS CloudFormation

Para habilitar o rastreamento ativo em um recurso `AWS::Lambda::Function` em um modelo do AWS CloudFormation, use a propriedade `TracingConfig`.

Example [function-inline.yml](#): configuração de rastreamento

```
Resources:
  function:
    Type: AWS::Lambda::Function
    Properties:
      TracingConfig:
        Mode: Active
```

...

Para um recurso do AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, use a propriedade `Tracing`.

Example [template.yml](#): configuração de rastreamento

```
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      Tracing: Active  
      ...
```

Usar variáveis de ambiente do

Para acessar [variáveis de ambiente \(p. 99\)](#) em Go, use a função `Getenv`.

O seguinte explica como fazer isso. Observe que a função importa o pacote `fmt` para formatar os resultados impressos e o pacote do `os`, uma interface do sistema independente de plataforma, que permite acessar variáveis de ambiente.

```
package main

import (
    "fmt"
    "os"
    "github.com/aws/aws-lambda-go/lambda"
)

func main() {
    fmt.Printf("%s is %. years old\n", os.Getenv("NAME"), os.Getenv("AGE"))
}
```

Para obter uma lista de variáveis de ambiente definidas pelo tempo de execução do Lambda, consulte [Variáveis de ambiente com tempo de execução definido \(p. 102\)](#).

Construir funções do Lambda com C#

As seções a seguir explicam como padrões de programação comuns e conceitos fundamentais são aplicados na criação do código da função do Lambda em C#.

O AWS Lambda fornece as seguintes bibliotecas para funções em C#:

- Amazon.Lambda.Core: esta biblioteca fornece um registrador estático do Lambda, interfaces de serialização e um objeto de contexto. O objeto `Context` ([AWS LambdaObjeto de contexto do em C# \(p. 676\)](#)) fornece informações no tempo de execução sobre suas funções do Lambda.
- Amazon.Lambda.Serialization.Json : essa é uma implementação da interface de serialização em Amazon.Lambda.Core.
- Amazon.Lambda.Logging.AspNetCore : Fornece uma biblioteca para o registro a partir de ASP.NET.
- Objetos de evento (POCOs) para vários produtos da AWS, incluindo:
 - Amazon.Lambda.APIGatewayEvents
 - Amazon.Lambda.CognitoEvents
 - Amazon.Lambda.ConfigEvents
 - Amazon.Lambda.DynamoDBEvents
 - Amazon.Lambda.KinesisEvents
 - Amazon.Lambda.S3Events
 - Amazon.Lambda.SQSEvents
 - Amazon.Lambda.SNSEvents

Esses pacotes estão disponíveis em [Nuget packages](#).

Tempos de execução do .NET

Nome	Identifier	Sistema operacional	
.NET Core 3.1	dotnetcore3.1	Amazon Linux 2	
.NET Core 2.1	dotnetcore2.1	Amazon Linux	

Note

Para obter informações sobre o fim do suporte para o NET Core 2.1, consulte [the section called “Política de suporte ao tempo de execução” \(p. 217\)](#).

Para começar com o desenvolvimento de aplicativos no ambiente local, implante um dos aplicativos de exemplo disponíveis no repositório do GitHub deste guia.

Aplicações de exemplo do Lambda em C#

- `blank-csharp`: uma função do C# que mostra o uso das bibliotecas .NET do Lambda, registro em log, variáveis de ambiente, rastreamento do AWS X-Ray, testes de unidade e do AWS SDK.
- `ec2-spot`: uma função que gerencia solicitações de instâncias spot no Amazon EC2.

Tópicos

- [AWS LambdaManipulador de função do em C# \(p. 663\)](#)
- [Implantar funções do Lambda em C# com arquivos .zip \(p. 668\)](#)
- [Implantar funções do Lambda em .NET com imagens de contêiner \(p. 674\)](#)
- [AWS LambdaObjeto de contexto do em C# \(p. 676\)](#)
- [AWS LambdaRegistro em log da função do em C# \(p. 677\)](#)
- [AWS LambdaErros da função do em C# \(p. 682\)](#)
- [Instrumentar o código C # no AWS Lambda \(p. 688\)](#)

AWS LambdaManipulador de função do em C#

O manipulador da função do Lambda é o método no código da função que processa eventos. Quando sua função é invocada, o Lambda executa o método do manipulador. Quando o manipulador é encerrado ou retorna uma resposta, ele se torna disponível para tratar de outro evento.

Você define um manipulador de função do Lambda como uma instância ou método estático em uma classe. Se você deseja obter acesso ao objeto de contexto do Lambda, ele é disponibilizado através da definição de um método parâmetro do tipo `ILambdaContext`, uma interface que você pode usar para acessar informações sobre a invocação atual, como o nome da função atual, limite de memória, tempo de execução restante e registro em log.

```
returnType handler-name(inputType input, ILambdaContext context) {  
    ...  
}
```

Observe o seguinte em relação à sintaxe:

- **`inputType`** – O primeiro parâmetro é a entrada do manipulador, que pode ser os dados do evento (publicados por uma fonte de evento) ou uma entrada personalizada fornecida por você, como uma string ou um objeto de dados personalizado.
- **`returnType`**: se você planeja invocar a função do Lambda de forma síncrona (usando o tipo de invocação `RequestResponse`), pode fazer com que a saída de sua função retorne um dos tipos de dados compatíveis. Por exemplo, se você usar uma função do Lambda como um backend de aplicação móvel, estará invocando-a de forma síncrona. O tipo de dados de saída será serializado em JSON.

Se você planeja invocar a função do Lambda de forma assíncrona (usando o tipo de invocação `returnType`), o `Event` deve ser `void`. Por exemplo, se você usar o AWS Lambda com origens de eventos como Amazon S3 ou Amazon SNS, essas origens de eventos invocarão a função do Lambda usando o tipo de invocação `Event`.

- **`ILambdaContext context`**: o segundo argumento na assinatura do manipulador é opcional. Concede acesso ao [objeto de contexto \(p. 676\)](#) que possui informações sobre a função e a solicitação.

Tratamento de streams

Somente o tipo `System.IO.Stream` é suportado como um parâmetro de entrada por padrão.

Por exemplo, considere o seguinte exemplo de código em C#.

```
using System.IO;  
  
namespace Example  
{  
    public class Hello  
    {  
        public Stream MyHandler(Stream stream)  
        {  
            //function logic  
        }  
    }  
}
```

No exemplo de código em C#, o primeiro parâmetro do manipulador é a entrada do manipulador (`MyHandler`), que pode ser os dados do evento (publicados por uma origem de evento, como o Amazon

S3) ou uma entrada personalizada fornecida por você, como um `Stream` (como neste exemplo) ou um objeto de dados personalizado qualquer. A saída é do tipo `Stream`.

Tratamento de tipos de dados padrão

Todos os outros tipos, como os listados abaixo, exigem que você especifique um serializador.

- Tipos .NET primitivos (como `string` ou `int`).
- Coleções e mapas - `IList`, `IEnumerable`, `IList<T>`, `Array`, `IDictionary`, `IDictionary< TKey, TValue >`
- Tipos de POCO (objetos CLR básicos)
- Tipos de eventos da AWS predefinidos
- Para invocações assíncronas, o tipo de retorno será ignorado pelo Lambda. O tipo de retorno pode ser definido como nulo em tais casos.
- Se você estiver usando programação assíncrona .NET, o retorno pode ser do tipo `Task` e `Task<T>` e usar as palavras-chave `async` e `await`. Para obter mais informações, consulte [Usar async em funções em C# com o AWS Lambda \(p. 666\)](#).

A não ser que os parâmetros de entrada e saída de sua função sejam do tipo `System.IO.Stream`, você precisará serializá-los. O AWS Lambda fornece um serializador padrão que pode ser aplicado no nível da montagem ou método de seu aplicativo ou você pode definir o seu próprio, implementando a interface `ILambdaSerializer` fornecida pela biblioteca `Amazon.Lambda.Core`. Para obter mais informações, consulte [Implantar funções do Lambda em C# com arquivos .zip \(p. 668\)](#).

Para adicionar o atributo de serializador padrão à um método, adicione primeiro uma dependência de `Amazon.Lambda.Serialization.Json` em seu arquivo `.csproj`.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>
    <AssemblyName>AssemblyName</AssemblyName>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="1.2.0" />
    <PackageReference Include="Amazon.Lambda.APIGatewayEvents" Version="2.3.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="1.8.0" />
    <PackageReference Include="Newtonsoft.Json" Version="12.0.1" />
  </ItemGroup>
</Project>
```

O exemplo abaixo ilustra a flexibilidade que você pode aproveitar especificando o serializador padrão `Json`.NET em um método e outro de sua escolha em um método diferente:

```
public class ProductService{
    [LambdaSerializer(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]
    public Product DescribeProduct(DescribeProductRequest request)
    {
        return catalogService.DescribeProduct(request.Id);
    }

    [LambdaSerializer(typeof(MyJsonSerializer))]
    public Customer DescribeCustomer(DescribeCustomerRequest request)
    {
        return customerService.DescribeCustomer(request.Id);
    }
}
```

}

Note

Se você estiver usando o .NET Core 3.1, recomendamos usar o serializador [Amazon.Lambda.Serialization.SystemTextJson](#). Este pacote fornece uma melhoria de performance em relação ao `Amazon.Lambda.Serialization.Json`.

Assinaturas do manipulador

Ao criar funções do Lambda, você precisa fornecer uma string de manipulador que diz ao AWS Lambda onde procurar o código a invocar. Em C#, o formato é:

ASSEMBLY::TYPE::METHOD onde:

- **ASSEMBLY** é o nome do arquivo de montagem .NET para o seu aplicativo. Ao usar a CLI do .NET Core para criar sua aplicação, se você não tiver definido o nome da montagem usando a propriedade `AssemblyName` em `.csproj`, **ASSEMBLY** será o nome do arquivo `.csproj`. Para obter mais informações, consulte [CLI do .NET Core \(p. 668\)](#). Neste caso, suponhamos que o arquivo `.csproj` seja `HelloWorldApp.csproj`.
- **TYPE** é o nome completo do tipo de manipulador, que consiste em **Namespace** e o **ClassName**. Nesse caso, `Example.Hello`.
- **METHOD** é o nome do manipulador da função, neste caso `MyHandler`.

Por fim, a assinatura será neste formato: **Assembly::Namespace.ClassName::MethodName**

Mais uma vez, considere o seguinte exemplo:

```
using System.IO;

namespace Example
{
    public class Hello
    {
        public Stream MyHandler(Stream stream)
        {
            //function logic
        }
    }
}
```

A string do manipulador seria: `HelloWorldApp::Example.Hello::MyHandler`

Important

Se o método especificado em sua string do manipulador estiver sobrecarregado, você deve fornecer a assinatura exata do método que o Lambda deve chamar. O AWS Lambda rejeitará uma assinatura considerada válida se a resolução exigir a seleção entre várias assinaturas (sobrearcarragadas).

Serializar funções do Lambda

Para qualquer função do Lambda que use tipos de entrada ou de saída diferentes do objeto `Stream`, você precisará adicionar uma biblioteca de serialização à sua aplicação. Isso pode ser feito das seguintes maneiras:

- Use o pacote `Amazon.Lambda.Serialization.Json` NuGet. Essa biblioteca usa JSON.NET para lidar com a serialização.

Note

Se você estiver usando o .NET Core 3.1, recomendamos usar o serializador [Amazon.Lambda.Serialization.SystemTextJson](#). Este pacote fornece uma melhoria de performance em relação ao `Amazon.Lambda.Serialization.Json`.

- Crie sua própria biblioteca de serialização implementando a interface `ILambdaSerializer`, que está disponível como parte da biblioteca `Amazon.Lambda.Core`. A interface define dois métodos:
 - `T Deserialize<T>(Stream requestStream);`
Implemente esse método para desserializar a carga da solicitação da API `Invoke` no objeto que é passado para o manipulador da função do Lambda.
 - `T Serialize<T>(T response, Stream responseStream);`
Implemente esse método para serializar o resultado retornado pelo manipulador de função do Lambda na carga útil da resposta retornada pela API `Invoke`.

Você usa o serializador que desejar adicionando-o como uma dependência a seu arquivo `MyProject.csproj`.

```
...
<ItemGroup>
  <PackageReference Include="Amazon.Lambda.Core" Version="1.0.0" />
  <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="1.3.0" />
</ItemGroup>
```

Em seguida, adicione-o ao arquivo `AssemblyInfo.cs`. Por exemplo, se você estiver usando o serializador padrão Json.NET, isto é o que você adicionaria:

```
[assembly:LambdaSerializer(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]
```

Note

Você pode definir um atributo de serialização personalizado no nível do método, que substituirá o serializador padrão especificado no nível de montagem. Para obter mais informações, consulte [Tratamento de tipos de dados padrão \(p. 664\)](#).

Restrições do manipulador de função do Lambda

Observe que há algumas restrições na assinatura do manipulador.

- Ela não pode ser `unsafe` e usar tipos de ponteiro na assinatura do manipulador, embora o contexto `unsafe` possa ser usado dentro do método do manipulador e suas dependências. Para obter mais informações, consulte [não seguro \(Referência de C#\)](#).
- Ela não pode passar um número variável de parâmetros usando a palavra-chave `params` ou usar `ArgIterator` como um parâmetro de entrada ou retorno que é usado para dar suporte ao número variável de parâmetros.
- O manipulador não pode ser um método genérico (por exemplo, `IList<T> Sort<T>(entrada IList<T>)`).
- Manipuladores assíncronos com assinaturas `async void` não são compatíveis.

Usar `async` em funções em C# com o AWS Lambda

Se você souber que sua função do Lambda exigirá um processo de longa duração, tal como o upload de grandes arquivos no Amazon S3 ou a leitura de uma grande transmissão de registros do DynamoDB, você

pode aproveitar a vantagem do padrão `async/await`. Quando você usa essa assinatura, o Lambda invoca a função de forma síncrona e aguarda a função retornar uma resposta ou aguarda que a execução atinja o tempo limite.

```
public async Task<Response> ProcessS3ImageResizeAsync(SimpleS3Event input)
{
    var response = await client.DoAsyncWork(input);
    return response;
}
```

Se você usar esse padrão, há algumas considerações que você deve levar em conta:

- O AWS Lambda não oferece suporte a métodos `async void`.
- Se você criar uma função do Lambda assíncrona sem implementar o operador `await`, o .NET emitirá um aviso do compilador e você vai observar um comportamento inesperado. Por exemplo, algumas ações assíncronas serão executadas enquanto outras não. Ou algumas ações assíncronas não serão concluídas antes que a invocação da função esteja concluída.

```
public async Task ProcessS3ImageResizeAsync(SimpleS3Event event) // Compiler warning
{
    client.DoAsyncWork(input);
}
```

- Sua função do Lambda pode incluir várias chamadas assíncronas que podem ser invocadas em paralelo. Você pode usar os métodos `Task.WhenAll` e `Task.WhenAny` para trabalhar com múltiplas tarefas. Para usar o método `Task.WhenAll`, a lista de operações é passada ao método como uma matriz. Observe que, no exemplo abaixo, se você não incluir qualquer operação na matriz, essa chamada pode retornar antes da conclusão de sua operação.

```
public async Task DoesNotWaitForAllTasks1()
{
    // In Lambda, Console.WriteLine goes to CloudWatch Logs.
    var task1 = Task.Run(() => Console.WriteLine("Test1"));
    var task2 = Task.Run(() => Console.WriteLine("Test2"));
    var task3 = Task.Run(() => Console.WriteLine("Test3"));

    // Lambda may return before printing "Test2" since we never wait on task2.
    await Task.WhenAll(task1, task3);
}
```

Para usar o método `Task.WhenAny`, você passa novamente uma lista de operações ao método como uma matriz. A chamada retorna assim que a primeira operação é concluída, mesmo se as outras ainda estiverem em execução.

```
public async Task DoesNotWaitForAllTasks2()
{
    // In Lambda, Console.WriteLine goes to CloudWatch Logs.
    var task1 = Task.Run(() => Console.WriteLine("Test1"));
    var task2 = Task.Run(() => Console.WriteLine("Test2"));
    var task3 = Task.Run(() => Console.WriteLine("Test3"));

    // Lambda may return before printing all tests since we're only waiting for one to
    // finish.
    await Task.WhenAny(task1, task2, task3);
}
```

Implantar funções do Lambda em C# com arquivos .zip

Um pacote de implantação do .NET Core (arquivo .zip) contém o assembly compilado de sua função junto com todas as suas dependências de assembly. O pacote também contém um arquivo `proj.deps.json`. Isso sinaliza para o tempo de execução do .NET Core todas as dependências de sua função e um arquivo `proj.runtimeconfig.json`, que é usado para configurar o tempo de execução. O comando `publish` da Command Line Interface (CLI – Interface da linha de comando) do .NET pode criar uma pasta com todos esses arquivos. Por padrão, o `proj.runtimeconfig.json` não é incluído porque normalmente um projeto do Lambda é configurado para ser uma biblioteca de classes. Para forçar que o `proj.runtimeconfig.json` seja gravado como parte do processo `publish`, transmita o argumento da linha de comando `/p:GenerateRuntimeConfigurationFiles=true` para o comando `publish`.

Embora seja possível criar o pacote de implantação com o comando `dotnet publish`, recomendamos que você crie o pacote de implantação com o [CLI do .NET Core \(p. 668\)](#) ou com o [AWS Toolkit for Visual Studio \(p. 671\)](#). Essas são ferramentas otimizadas especificamente para o Lambda para garantir que o arquivo `lambda-project.runtimeconfig.json` exista e otimize o pacote, incluindo a remoção de qualquer dependência não baseada no Linux.

Tópicos

- [CLI do .NET Core \(p. 668\)](#)
- [AWS Toolkit for Visual Studio \(p. 671\)](#)

CLI do .NET Core

A CLI do .NET Core oferece uma forma de plataforma cruzada para criar aplicações Lambda com base no .NET. Esta seção pressupõe que você instalou a CLI do .NET Core. Se não tiver feito isso, consulte [Download .NET](#) no site da Microsoft.

Na CLI do .NET, você usa o comando `new` para criar projetos .NET em uma linha de comando. Isso é útil se você deseja criar um projeto fora do Visual Studio. Para visualizar uma lista de tipos de projetos disponíveis, abra uma linha de comando e navegue até onde você instalou o tempo de execução do .NET Core e execute o seguinte comando:

Templates	Short Name	Language	Tags
<hr/>			
Console Application	console	[C#], F#, VB	
Common/Console			
Class library	classlib	[C#], F#, VB	
Common/Library			
Unit Test Project	mstest	[C#], F#, VB	
Test/MSTest			
xUnit Test Project	xunit	[C#], F#, VB	
Test/xUnit			
<hr/>			
Examples:			
<code>dotnet new mvc --auth Individual</code>			
<code>dotnet new viewstart</code>			
<code>dotnet new --help</code>			

O Lambda oferece modelos adicionais por meio do pacote nuget [Amazon.Lambda.Templates](#). Para instalar esse pacote, execute o seguinte comando:

```
dotnet new -i Amazon.Lambda.Templates
```

Quando a instalação estiver concluída, os modelos do Lambda serão exibidos como parte de dotnet new. Para examinar detalhes sobre um modelo, use a opção help.

```
dotnet new lambda.EmptyFunction --help
```

O modelo lambda.EmptyFunction é compatível com as seguintes opções:

- `--name`: o nome da função
- `--profile`: o nome de um perfil no seu [arquivo de credenciais do AWS SDK for .NET](#)
- `--region`: a região da AWS na qual criar a função.

Essas opções são salvas em um arquivo chamado `aws-lambda-tools-defaults.json`.

Crie um projeto de função com o modelo do `lambda.EmptyFunction`.

```
dotnet new lambda.EmptyFunction --name MyFunction
```

No diretório `src/myfunction`, examine os seguintes arquivos:

- `aws-lambda-tools-defaults.json`: este é o local em que você especifica as opções de linha de comando ao implantar sua função do Lambda. Por exemplo:

```
"profile" : "default",
"region" : "us-east-2",
"configuration" : "Release",
"framework" : "netcoreapp2.1",
"function-runtime": "dotnetcore3.1",
"function-memory-size" : 256,
"function-timeout" : 30,
"function-handler" : "MyFunction::MyFunction.Function::FunctionHandler"
```

- `Function.cs`: o código da função do manipulador do Lambda. É um modelo C # que inclui a biblioteca `Amazon.Lambda.Core` padrão e um atributo `LambdaSerializer` padrão. Para obter mais informações sobre os requisitos e as opções de serialização, consulte [Serializar funções do Lambda \(p. 665\)](#). Isso também inclui uma função de exemplo que você pode editar para aplicar o código de sua função do Lambda.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

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.Json.JsonSerializer))]

namespace MyFunction
{
    public class Function
    {

        public string FunctionHandler1(string input, ILambdaContext context)
        {
            return input?.ToUpper();
        }
    }
}
```

```
        }
    }
```

- MyFunction.csproj: um arquivo [MSBuild](#) que lista os arquivos e assemblies que compõem seu aplicativo.

```
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>
    <TargetFramework>netcoreapp2.1</TargetFramework>
</PropertyGroup>

<ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="1.0.0 " />
    <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="1.3.0" />
</ItemGroup>

</Project>
```

- Readme: use este arquivo para documentar sua função do Lambda.

No diretório myfunction/test, examine os seguintes arquivos:

- myFunction.Tests.csproj: conforme observado anteriormente, este é um arquivo [MSBuild](#) que lista os arquivos e assemblies que compõem seu projeto de teste. Observe também que ele inclui a biblioteca Amazon.Lambda.Core, o que permite que você integre perfeitamente qualquer modelo do Lambda necessário para testar sua função.

```
<Project Sdk="Microsoft.NET.Sdk">
    ...
    <PackageReference Include="Amazon.Lambda.Core" Version="1.0.0 " />
    ...
```

- FunctionTest.cs: o mesmo arquivo de modelo de código C # que é incluído no diretório src. Edite esse arquivo para espelhar o código de produção de sua função e testá-lo antes de carregar sua função do Lambda em um ambiente de produção.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

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);
    }
}
```

Assim que sua função tiver passado nos testes, você pode compilá-la e implantá-la usando a `Amazon.Lambda.Tools .NET Core Global Tool`. Para instalar a .NET Core Global Tool, execute o seguinte comando:

```
dotnet tool install -g Amazon.Lambda.Tools
```

Se você já tiver a ferramenta instalada, certifique-se de que esteja usando a versão mais recente com o seguinte comando:

```
dotnet tool update -g Amazon.Lambda.Tools
```

Para obter mais informações sobre a `Amazon.Lambda.Tools .NET Core Global Tool`, consulte o repositório de [Extensões da AWS para CLI do .NET](#) no GitHub.

Com o `Amazon.Lambda.Tools` instalado, você pode implantar sua função usando o seguinte comando:

```
dotnet lambda deploy-function MyFunction --function-role role
```

Após a implantação, você pode testá-la novamente em um ambiente de produção com o comando a seguir e passar um valor diferente para o manipulador da função do Lambda:

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

```
dotnet lambda invoke-function MyFunction --cli-binary-format raw-in-base64-out --payload  
"Just Checking If Everything is OK"
```

Se tudo for bem-sucedido, você verá o seguinte:

```
dotnet lambda invoke-function MyFunction --cli-binary-format raw-in-base64-out --payload  
"Just Checking If Everything is OK"  
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
```

AWS Toolkit for Visual Studio

Você pode criar aplicações do Lambda baseadas em .NET usando o plugin do Lambda para o [AWS Toolkit for Visual Studio](#). O toolkit está disponível como uma [extensão do Visual Studio](#).

1. Execute o Microsoft Visual Studio e escolha New project.
 - a. No menu File, escolha New e, em seguida, Project.
 - b. Na janela New Project (Novo projeto), escolha Lambda Project (.NET Core) e, em seguida, escolha OK.

- c. Na janela Select Blueprint (Selecionar planta), será apresentada a opção de selecionar em uma lista de aplicações de exemplo que fornecerá o código de exemplo para começar com a criação de uma aplicação do Lambda com base no .NET.
 - d. Para criar uma aplicação do Lambda do zero, escolha Empty Function (Função vazia) e Finish (Concluir).
2. Examine o arquivo `aws-lambda-tools-defaults.json`, criado como parte do projeto. Você pode definir as opções nesse arquivo, que, por padrão, é lido pelas ferramentas do Lambda. Os modelos de projeto criados no Visual Studio definem muitos desses campos com valores padrão. Observe os seguintes campos:
 - profile: o nome de um perfil no seu [arquivo de credenciais do AWS SDK for .NET](#)
 - function-handler: esse é o local em que o `function handler` é especificado, e é por isso que não é necessário defini-lo no assistente. No entanto, sempre que você renomear `Assembly`, `Namespace`, `Class` ou `Function` no código da função, será necessário atualizar os campos correspondentes no arquivo `aws-lambda-tools-defaults.json`.

```
{
  "profile": "default",
  "region" : "us-east-2",
  "configuration" : "Release",
  "framework" : "netcoreapp2.1",
  "function-runtime": "dotnetcore3.1",
  "function-memory-size" : 256,
  "function-timeout" : 30,
  "function-handler" : "Assembly::Namespace.Class::Function"
}
```

3. Abra o arquivo `Function.cs`. Você recebe um modelo para implementar o código do manipulador da função de Lambda.

```

using System;
using Amazon.Lambda.Core;
using Amazon.Lambda.Serialization;

// Assembly attribute to enable the Lambda function's JSON input to be converted into a .NET class.
[assembly: LambdaSerializerAttribute(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]

namespace AWSLambda
{
    public class LambdaFunction
    {

        /// <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();
        }
    }
}

```

4. Assim que você tiver criado o código que representa sua função do Lambda, faça upload dele abrindo o menu de contexto (clique com o botão direito do mouse) no nó Project (Projeto) em sua aplicação e, em seguida, escolhendo Publish to AWS Lambda (Publicar no AWS Lambda).
5. Na janela Upload Lambda Function (Fazer upload da função do Lambda), insira um nome para a função ou selecione uma função publicada anteriormente para ser republicada. Em seguida, escolha Avançar.
6. Na janela Advanced Function Details (Detalhes da função avançada), configure as opções a seguir:
 - Função(obrigatório) — O [AWS Identity and Access Management](#)Função do (IAM) para (p. 57)O Lambda assume quando executa a função.

- Environment (Ambiente): pares de chave-valor que o Lambda define no ambiente de execução. [Use variáveis de ambiente \(p. 99\)](#) para estender a configuração da função fora do código.
 - Memory (Memória): a quantidade de memória disponível para a função no tempo de execução. Escolha um valor [entre 128 MB e 10.240 MB \(p. 53\)](#) em incrementos de 1 MB.
 - Timeout (Tempo limite): a quantidade de tempo durante a qual o Lambda permite que uma função seja executada antes de interrompê-la. O padrão é três segundos. O valor máximo permitido é de 900 segundos.
 - VPC: se a função precisa de acesso à rede para recursos que não estão disponíveis pela Internet, [configure-a para se conectar a uma Virtual Private Cloud \(VPC\) \(p. 124\)](#).
 - DLQ (Fila de mensagens mortas): se a função for invocada de forma assíncrona, [escolha uma fila de mensagens mortas \(p. 167\)](#) para receber invocações com falha.
 - Enable active tracing (Habilitar o rastreamento ativo): crie uma amostra de solicitações recebidas e [rastreie solicitações de amostra com o AWS X-Ray \(p. 477\)](#).
7. Escolha Next (Avançar) e, em seguida, escolha Upload para implantar a aplicação.

Para obter mais informações, consulte [Implantar um projeto do AWS Lambda com a CLI do .NET Core](#).

Implantar funções do Lambda em .NET com imagens de contêiner

Você pode implantar seu código de função do Lambda como uma [imagem de contêiner \(p. 267\)](#). A AWS oferece os seguintes recursos para ajudar você a criar uma imagem de contêiner para sua função .NET:

- [AWSImagens base para o Lambda](#)

Essas imagens base são pré-carregadas com um tempo de execução de linguagem e outros componentes necessários para executar a imagem no Lambda. AWS fornece um Dockerfile para cada uma das imagens de base para ajudar a criar sua imagem de contêiner.

- Clientes de interface de tempo de execução de

Se você usar uma imagem de base de comunidade ou de empresa privada, adicione um cliente de interface de tempo de execução à imagem base para torná-lo compatível com o Lambda.

O fluxo de trabalho de uma função definida como uma imagem de contêiner inclui as seguintes etapas:

1. Crie sua imagem de contêiner usando os recursos listados neste tópico.
2. Faça upload da imagem em seu registro do contêiner do Amazon ECR. Consulte os passos 7 a 9 no [Criar imagem \(p. 268\)](#).
3. [Criar \(p. 91\)](#) a função do Lambda ou [atualizar o código da função \(p. 92\)](#) para implantar a imagem em uma função existente.

Tópicos

- [AWSImagens de base da para .NET \(p. 674\)](#)
- [Usar uma imagem base .NET \(p. 675\)](#)
- [Clientes de interface de tempo de execução .NET \(p. 675\)](#)
- [Implantar a imagem do contêiner \(p. 675\)](#)

AWSImagens de base da para .NET

AWSA oferece as seguintes imagens de base para .NET:

Tags	Tempo de execução	Sistema operacional	Dockerfile
5,0	.NET 5.0	Amazon Linux 2	Dockerfile para .NET 5.0 no GitHub
core3.1	.NET Core 3.1	Amazon Linux 2	Dockerfile para .NET 3.1 no GitHub
core2.1	.NET Core 2.1	Amazon Linux 2018.03	Dockerfile para .NET 2.1 no GitHub

Repositório do Docker Hub: [amazon/aws-lambda-dotnet](#)

Repositório do Amazon ECR: [gallery.ecr.aws/lambda/dotnet](#)

Usar uma imagem base .NET

Para obter instruções sobre como usar uma imagem base .NET, escolha a guia usage (uso) em [Imagens base do AWS Lambda para .NET](#) no repositório do Amazon ECR.

As instruções também estão disponíveis em [Imagens base do Lambda para .NET](#) no repositório do Docker Hub.

Cientes de interface de tempo de execução .NET

Faça download do cliente de interface de tempo de execução .NET no repositório do [AWS Lambda para .NET Core](#) no GitHub.

Implantar a imagem do contêiner

Para uma nova função, você implanta a imagem do contêiner quando [cria a função \(p. 91\)](#). Para uma função existente, se você reconstruir a imagem do contêiner, precisará reimplantar a imagem ao [atualizar o código da função \(p. 92\)](#).

AWS LambdaObjeto de contexto do em C#

Quando o Lambda executa a função, ele transmite um objeto de contexto para o [handler \(p. 663\)](#). Esse objeto fornece propriedades com informações sobre a invocação, a função e o ambiente de execução.

Propriedades de contexto

- `FunctionName`: o nome da função do Lambda.
- `FunctionVersion`: a [versão \(p. 106\)](#) da função.
- `InvokedFunctionArn`: o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um alias ou número de versão.
- `MemoryLimitInMB`: a quantidade de memória alocada para a função.
- `AwsRequestId`: o identificador da solicitação de invocação.
- `LogGroupName`: o grupo de logs da função.
- `LogStreamName`: a transmissão de log para a instância da função.
- `RemainingTime (TimeSpan)`: o número de milissegundos restantes antes do tempo limite da execução.
- `Identity`: (aplicativos móveis) informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
- `ClientContext`: (aplicativos móveis) contexto do cliente fornecido ao Lambda pela aplicação cliente.
- `Logger` O [objeto logger \(p. 677\)](#) para a função.

O trecho do código C# a seguir mostra uma função de manipulador simples que imprime algumas das informações de contexto.

```
public async Task Handler(ILambdaContext context)
{
    Console.WriteLine("Function name: " + context.FunctionName);
    Console.WriteLine("RemainingTime: " + context.RemainingTime);
    await Task.Delay(TimeSpan.FromSeconds(0.42));
    Console.WriteLine("RemainingTime after sleep: " + context.RemainingTime);
}
```

AWS Lambda Registro em log da função do em C#

O AWS Lambda monitora automaticamente as funções do Lambda em seu nome e envia métricas da função para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente do tempo de execução do Lambda envia detalhes sobre cada invocação à transmissão de logs e transmite os logs e outras saídas do código de sua função.

Esta página descreve como produzir a saída de logs usando o código de sua função do Lambda ou acessar os logs usando a AWS Command Line Interface, o console do Lambda ou o console do CloudWatch.

Seções

- [Criar uma função que retorna logs \(p. 677\)](#)
- [Usar o console do Lambda \(p. 678\)](#)
- [Usando o console do CloudWatch \(p. 678\)](#)
- [Usar a AWS Command Line Interface \(AWS CLI\) \(p. 679\)](#)
- [Excluir logs \(p. 681\)](#)

Criar uma função que retorna logs

Para gerar os logs do código de função, você pode usar métodos na [classe Console](#) ou qualquer biblioteca de logs que grave no `stdout` ou no `stderr`. O exemplo a seguir usa a classe `LambdaLogger` da biblioteca [Amazon.Lambda.Core \(p. 661\)](#).

Example [src/blank-csharp/Function.cs](#): registro em log

```
public async Task<AccountUsage> FunctionHandler(SQSEvent invocationEvent, ILambdaContext context)
{
    GetAccountSettingsResponse accountSettings;
    try
    {
        accountSettings = await callLambda();
    }
    catch (AmazonLambdaException ex)
    {
        throw ex;
    }
    AccountUsage accountUsage = accountSettings.AccountUsage;
    LambdaLogger.Log("ENVIRONMENT VARIABLES: " +
JsonConvert.SerializeObject(System.Environment.GetEnvironmentVariables()));
    LambdaLogger.Log("CONTEXT: " + JsonConvert.SerializeObject(context));
    LambdaLogger.Log("EVENT: " + JsonConvert.SerializeObject(invocationEvent));
    return accountUsage;
}
```

Example Formato do log

```
START RequestId: d1cf0ccb-xmpl-46e6-950d-04c96c9b1c5d Version: $LATEST
ENVIRONMENT VARIABLES:
{
    "AWS_EXECUTION_ENV": "AWS_Lambda_dotnetcore2.1",
    "AWS_LAMBDA_FUNCTION_MEMORY_SIZE": "256",
    "AWS_LAMBDA_LOG_GROUP_NAME": "/aws/lambda/blank-csharp-function-WU56XMPV2XA",
    "AWS_LAMBDA_FUNCTION_VERSION": "$LATEST",
```

```
"AWS_LAMBDA_LOG_STREAM_NAME": "2020/03/27[$LATEST]5296xmpl084f411d9fb73b258393f30c",
"AWS_LAMBDA_FUNCTION_NAME": "blank-csharp-function-WU56XMPLV2XA",
...
EVENT:
{
  "Records": [
    {
      "MessageId": "19dd0b57-b21e-4ac1-bd88-01bbb068cb78",
      "ReceiptHandle": "MessageReceiptHandle",
      "Body": "Hello from SQS!",
      "Md5OfBody": "7b270e59b47ff90a553787216d55d91d",
      "Md5OfMessageAttributes": null,
      "EventSourceArn": "arn:aws:sqs:us-west-2:123456789012:MyQueue",
      "EventSource": "aws:sqs",
      "AwsRegion": "us-west-2",
      "Attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1523232000000",
        "SenderId": "123456789012",
        "ApproximateFirstReceiveTimestamp": "1523232000001"
      },
      ...
    }
  ]
}
END RequestId: d1cf0ccb-xmpl-46e6-950d-04c96c9b1c5d
REPORT RequestId: d1cf0ccb-xmpl-46e6-950d-04c96c9b1c5d Duration: 4157.16 ms Billed Duration: 4200 ms Memory Size: 256 MB Max Memory Used: 99 MB Init Duration: 841.60 ms XRAY TraceId: 1-5e7e8131-7ff0xmpl32bfb31045d0a3bb SegmentId: 0152xmpl6016310f Sampled: true
```

O tempo de execução do .NET registra em log as linhas START, END e REPORT para cada invocação. A linha do relatório fornece os seguintes detalhes.

Registro em log de relatório

- RequestId: o ID de solicitação exclusivo para a invocação.
- Duração: a quantidade de tempo que o método de manipulador da função gastou processando o evento.
- Duração faturada: a quantia de tempo faturada para a invocação.
- Tamanho da memória: a quantidade de memória alocada para a função.
- Memória máxima utilizada: a quantidade de memória utilizada pela função.
- Duração inicial: para a primeira solicitação atendida, a quantidade de tempo que o tempo de execução levou para carregar a função e executar o código fora do método do manipulador.
- XRAY Traceld: para solicitações rastreadas, o [ID de rastreamento do AWS X-Ray \(p. 477\)](#).
- SegmentId: para solicitações rastreadas, o ID do segmento do X-Ray.
- Amostragem: para solicitações rastreadas, o resultado da amostragem.

Usar o console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda. Para obter mais informações, consulte [Acessar o Amazon CloudWatch Logs para o AWS Lambda \(p. 720\)](#).

Usando o console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a [página Log groups](#) (Grupos de log).

2. Escolha o grupo de logs de sua função (/aws/lambda/**nome-de-sua-função**).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma [instância da sua função \(p. 219\)](#). Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos instrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

Para usar uma aplicação de exemplo que correlaciona os logs e os rastreamentos com o X-Ray, consulte [Aplicativo de exemplo de processador de erros para o AWS Lambda \(p. 500\)](#).

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Você pode usar a [AWS CLI](#) para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Você deve ver a saída a seguir:

```
{  
    "StatusCode": 200,  
    "LogResult":  
        "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",  
    "ExecutedVersion": "$LATEST"  
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \  
--query 'LogResult' --output text | base64 -d
```

Você deve ver a saída a seguir:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e [Ubuntu no Windows](#). Os usuários do macOS precisam usar `base64 -D`.

Example get-logs.sh script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa sed para remover as aspas do arquivo de saída e fica inativo por 15 segundos para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando `get-log-events`.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como `get-logs.sh`.

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

```
#!/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 $(cat
out) --limit 5
```

Example macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod +x get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

Você deve ver a saída a seguir:

```
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\+INFO\+ENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$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"
}
```

Excluir logs

Os grupos de logs não são excluídos automaticamente quando você exclui uma função. Para evitar armazenar logs indefinidamente, exclua o grupo de logs ou[Configurar um período de retenção](#)após o qual os logs são excluídos automaticamente.

AWS Lambda Erros da função do em C#

Quando o código gera um erro, o Lambda gera uma representação JSON do erro. Esse documento de erro aparece no log de invocação e, para invocações síncronas, na saída.

Esta página descreve como exibir erros de invocação de função do Lambda para o tempo de execução do C# usando o console do Lambda e a AWS CLI.

Seções

- [Syntax \(p. 682\)](#)
- [Como funcionam \(p. 684\)](#)
- [Usar o console do Lambda \(p. 685\)](#)
- [Usar a AWS Command Line Interface \(AWS CLI\) \(p. 685\)](#)
- [Tratamento de erros em outros serviços da AWS \(p. 686\)](#)
- [Próximas etapas \(p. 686\)](#)

Syntax

Na fase de inicialização, exceções podem ser geradas para strings de manipulador inválidas, um tipo ou método de violação de regra (consulte [Restrições do manipulador de função do Lambda \(p. 666\)](#)) ou qualquer outro método de validação (como ignorar o atributo do serializador e ter um POCO como seu tipo de entrada ou saída). Essas exceções são do tipo `LambdaException`. Por exemplo:

```
{  
  "errorType": "LambdaException",  
  "errorMessage": "Invalid lambda function handler: 'http://this.is.not.a.valid.handler/'.  
  The valid format is 'ASSEMBLY::TYPE::METHOD'."  
}
```

Se o seu construtor gerar uma exceção, o tipo de erro também será do tipo `LambdaException`, mas a exceção gerada durante a construção é fornecida na propriedade `cause`, que é em si um objeto de exceção modelado:

```
{  
  "errorType": "LambdaException",  
  "errorMessage": "An exception was thrown when the constructor for type  
'LambdaExceptionTestFunction.ThrowExceptionInConstructor'  
  was invoked. Check inner exception for more details.",  
  "cause": {  
    "errorType": "TargetInvocationException",  
    "errorMessage": "Exception has been thrown by the target of an invocation.",  
    "stackTrace": [  
      "at System.RuntimeTypeHandle.CreateInstance(RuntimeType type, Boolean publicOnly,  
Boolean noCheck, Boolean&canBeCached,  
      RuntimeMethodHandleInternal&ctor, Boolean&bNeedSecurityCheck)",  
      "at System.RuntimeType.CreateInstanceSlow(Boolean publicOnly, Boolean skipCheckThis,  
Boolean fillCache, StackCrawlMark& stackMark)",  
      "at System.Activator.CreateInstance(Type type, Boolean nonPublic)",  
      "at System.Activator.CreateInstance(Type type)"  
    ],  
    "cause": {  
      "errorType": "ArithmetException",  
      "errorMessage": "Sorry, 2 + 2 = 5",  
      "stackTrace": [  
        "at LambdaExceptionTestFunction.ThrowExceptionInConstructor..ctor()"  
      ]  
    }  
  }  
}
```

```
}
```

Como mostra o exemplo, as exceções internas são sempre preservadas (como a propriedade `cause`) e podem ser profundamente aninhadas.

Exceções também podem ocorrer durante a invocação. Neste caso, o tipo de exceção é preservado e a exceção é retornada diretamente como a carga útil e no CloudWatch Logs. Por exemplo:

```
{
  "errorType": "AggregateException",
  "errorMessage": "One or more errors occurred. (An unknown web exception occurred!)",
  "stackTrace": [
    "at System.Threading.Tasks.Task.ThrowIfExceptional(Boolean
includeTaskCanceledExceptions)",
    "at System.Threading.Tasks.Task`1.GetResultCore(Boolean waitCompletionNotification)",
    "at lambda_method(Closure , Stream , Stream , ContextInfo )"
  ],
  "cause": {
    "errorType": "UnknownWebException",
    "errorMessage": "An unknown web exception occurred!",
    "stackTrace": [
      "at LambdaDemo107.LambdaEntryPoint.<GetUriResponse>d__1.MoveNext()",
      "--- End of stack trace from previous location where exception was thrown ---",
      "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
      "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
task)",
      "at System.Runtime.CompilerServices.TaskAwaiter`1.GetResult()",  

      "at LambdaDemo107.LambdaEntryPoint.<CheckWebsiteStatus>d__0.MoveNext()"
    ],
    "cause": {
      "errorType": "WebException",
      "errorMessage": "An error occurred while sending the request. SSL peer certificate or
SSH remote key was not OK",
      "stackTrace": [
        "at System.Net.HttpWebRequest.EndGetResponse(IAsyncResult asyncResult)",
        "at System.Threading.Tasks.TaskFactory`1.FromAsyncCoreLogic(IAsyncResult iar,
Func`2 endFunction, Action`1 endAction, Task`1 promise, Boolean requiresSynchronization)",
        "--- End of stack trace from previous location where exception was thrown ---",
        "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
        "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
task)",
        "at System.Runtime.CompilerServices.TaskAwaiter`1.GetResult()",  

        "at LambdaDemo107.LambdaEntryPoint.<GetUriResponse>d__1.MoveNext()"
      ],
      "cause": {
        "errorType": "HttpRequestException",
        "errorMessage": "An error occurred while sending the request.",
        "stackTrace": [
          "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
          "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
task)",
          "at System.Net.Http.HttpClient.<FinishSendAsync>d__58.MoveNext()",
          "--- End of stack trace from previous location where exception was thrown ---",
          "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
          "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
task)",
          "at System.Net.HttpWebRequest.<SendRequest>d__63.MoveNext()",
          "--- End of stack trace from previous location where exception was thrown ---",
          "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",  

          "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
task)",
          "at System.Net.HttpWebRequest.<SendRequest>d__63.MoveNext()",
          "--- End of stack trace from previous location where exception was thrown ---",
          "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
          "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
task)"
        ]
      }
    }
  }
}
```

```
        "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
task)",
        "at System.Net.HttpWebRequest.EndGetResponse(IAsyncResult asyncResult)"
],
"cause": {
    "errorType": "CurlException",
    "errorMessage": "SSL peer certificate or SSH remote key was not OK",
    "stackTrace": [
        "at System.Net.Http.CurlHandler.ThrowIfCURLError(CURLcode error)",
        "at
System.Net.Http.CurlHandler.MultiAgent.FinishRequest(StrongToWeakReference`1 easyWrapper,
CURLcode messageResult)"
    ]
}
}
}
```

O método no qual as informações de erro são transmitidas depende do tipo de invocação:

- **RequestResponse** Tipo de invocação (ou seja, execução síncrona): nesse caso, você recebe uma mensagem de erro de volta.

Por exemplo, se você invocar uma função do Lambda usando o console do Lambda, **RequestResponse** será sempre o tipo de invocação e o console exibirá as informações de erro retornadas pelo AWS Lambda na seção Execution result (Resultado da execução) do console.

- Tipo de invocação **Event** (ou seja, execução assíncrona): nesse caso, o AWS Lambda não retorna nada. Em vez disso, ele registra as informações de erro em métricas do CloudWatch Logs e do CloudWatch.

Como funcionam

Ao invocar uma função do Lambda, o Lambda recebe a solicitação de invocação e valida as permissões de sua função de execução, verifica se o documento do evento é um documento JSON válido e verifica valores de parâmetros.

Se a solicitação for validada, o Lambda a envia para uma instância da função. O ambiente de [tempo de execução do Lambda \(p. 214\)](#) converte o documento do evento em um objeto e o transmite ao handler da função.

Se o Lambda encontra um erro, ele retorna um tipo de exceção, uma mensagem e o código HTTP do status que indica a causa do erro. O cliente ou o serviço que invocou a função do Lambda pode processar o erro de maneira programática ou transmiti-lo a um usuário final. O comportamento correto de tratamento de erros depende do tipo de aplicativo, do público e da origem do erro.

A lista a seguir descreve o intervalo de códigos de status que você pode receber do Lambda.

2xx

Um erro da série 2xx com um cabeçalho **X-Amz-Function-Error** na resposta indica um erro de tempo de execução ou de função do Lambda. Um código de status da série 2xx indica que o Lambda aceitou a solicitação, mas em vez de um código de erro, o Lambda indica o erro incluindo o cabeçalho **X-Amz-Function-Error** na resposta.

4xx

Um erro da série 4xx indica um erro que o cliente ou serviço que fez a invocação pode corrigir modificando a solicitação, solicitando permissão ou tentando a solicitação novamente. Os erros da série 4xx diferentes de 429 geralmente indicam um erro na solicitação.

5xx

Um erro da série 5xx indica um problema com o Lambda ou com a configuração/recursos da função. Os erros da série 5xx podem indicar uma condição temporária que pode ser resolvida sem nenhuma ação do usuário. O cliente ou serviço que fez a invocação não podem solucionar esses problemas, mas o proprietário de uma função do Lambda pode ser capaz de corrigi-los.

Para obter uma lista completa de erros de invocação, consulte [Erros de InvokeFunction \(p. 877\)](#).

Usar o console do Lambda

Você pode invocar sua função no console do Lambda configurando um evento de teste e visualizando a saída. A saída é capturada pelos logs de execução da função e, quando o [rastreamento ativo \(p. 477\)](#) está habilitado, pelo AWS X-Ray.

Para invocar uma função no console do Lambda

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Test (Testar).
4. Selecione New event (Novo evento) e escolha um Event template (Modelo de evento) na lista suspensa.
5. Insira um nome para o evento de teste.
6. Digite o JSON para o evento de teste.
7. Escolha Create event (Criar evento).
8. Escolha Invoke (Invocar).

O console do Lambda invoca sua função [de forma síncrona \(p. 158\)](#) e exibe o resultado. Para ver a resposta, os logs e outras informações, expanda a seção Details (Detalhes).

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Quando você invoca uma função do Lambda na AWS CLI, a AWS CLI divide a resposta em dois documentos. A resposta da AWS CLI é exibida no prompt de comando. Se ocorreu um erro, a resposta contém um campo `FunctionError`. A resposta ou o erro de invocação retornado pela função é gravado no arquivo de saída. Por exemplo, o `output.json` ou o `output.txt`.

O exemplo de comando `invoke` a seguir demonstra como invocar uma função e escrever a resposta de invocação em um arquivo `output.txt`.

```
aws lambda invoke \
--function-name my-function \

```

```
--cli-binary-format raw-in-base64-out \
--payload '{"key1": "value1", "key2": "value2", "key3": "value3"}' output.txt
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

Você deve ver a resposta da AWS CLI em seu prompt de comando:

```
{
  "StatusCode": 200,
  "FunctionError": "Unhandled",
  "ExecutedVersion": "$LATEST"
}
```

Você deve ver a resposta de invocação de função no arquivo `output.txt`. Também é possível visualizar a saída no mesmo prompt de comando usando:

```
cat output.txt
```

Você deve ver a resposta de invocação no prompt de comando.

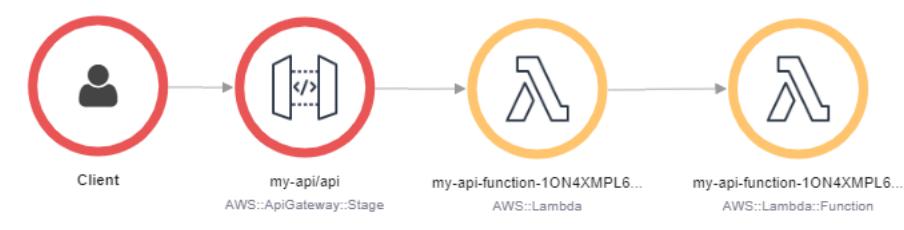
O Lambda também grava até 256 KB do objeto de erro nos logs de função. Para obter mais informações, consulte [AWS Lambda Registro em log da função do em C# \(p. 677\)](#).

Tratamento de erros em outros serviços da AWS

Quando outro serviço da AWS invoca sua função, o serviço escolhe o tipo de invocação e o comportamento de repetição. Os serviços da AWS podem invocar sua função em um agendamento, em resposta a um evento de ciclo de vida em um recurso ou para atender a uma solicitação de um usuário. Alguns serviços invocam funções de forma assíncrona e permitem que o Lambda trate erros, enquanto outros fazem novas tentativas ou transmitem os erros de volta ao usuário.

Por exemplo, o API Gateway trata todos os erros de invocação e função como erros internos. Se a API do Lambda rejeitar a solicitação de invocação, o API Gateway retornará um código de erro 500. Se a função é executada, mas retorna um erro ou retorna uma resposta no formato errado, o API Gateway retorna um código de erro 502. Para personalizar a resposta de erro, é necessário detectar erros no código e formatar uma resposta no formato necessário.

Recomendamos usar AWS X-Ray para determinar a fonte de um erro e a respectiva causa. O X-Ray permite localizar qual componente encontrou um erro e ver detalhes sobre os erros. O exemplo a seguir mostra um erro de função que resultou em uma resposta 502 do API Gateway.



Para obter mais informações, consulte [Instrumentar o código C # no AWS Lambda \(p. 688\)](#).

Próximas etapas

- Saiba como mostrar eventos de registro em log para sua função do Lambda na página [the section called “Registro em log” \(p. 677\)](#).

Instrumentar o código C # no AWS Lambda

O Lambda se integra ao AWS X-Ray para permitir que você rastreie, depure e otimize aplicativos do Lambda. É possível usar o X-Ray para rastrear uma solicitação à medida que ela atravessa recursos na aplicação, da API de frontend ao armazenamento e aos banco de dados no backend. Ao simplesmente adicionar a biblioteca do X-Ray SDK à configuração de compilação, é possível registrar erros e latência para qualquer chamada que a função faça para um serviço da AWS.

O X-Ray Mapa de serviço mostra o fluxo de solicitações através de seu aplicativo. O exemplo a seguir do aplicativo de exemplo [processador de erros \(p. 500\)](#) mostra um aplicativo com duas funções. A função principal processa eventos e, às vezes, retorna erros. A segunda função processa erros que aparecem no primeiro grupo de logs e usa o AWS SDK para chamar o X-Ray, o Amazon S3 e o Amazon CloudWatch Logs.



Para rastrear solicitações que não têm um cabeçalho de rastreamento, ative o rastreamento ativo na configuração da sua função.

Para ativar o rastreamento ativo

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Selecione **Configuração**, depois, escolha **Ferramentas de monitoramento**.
4. Selecione **Edit**.
5. Em X-Ray, habilite o **Active tracing (Rastreamento ativo)**.
6. Escolha **Save (Salvar)**.

Pricing

O X-Ray tem um nível gratuito vitalício. Além do limite do nível gratuito, o X-Ray cobra por armazenamento e recuperação de rastreamento. Para obter detalhes, consulte [Definição de preço do AWS X-Ray](#).

Sua função precisa de permissão para carregar dados de rastreamento no X-Ray. Quando você habilita o rastreamento ativo no console do Lambda, o Lambda adiciona as permissões necessárias à [função de](#)

execução (p. 57) da função. Caso contrário, adicione a política [AWSXRayDaemonWriteAccess](#) à função de execução.

O X-Ray aplica um algoritmo de amostragem para garantir que o rastreamento é eficiente, enquanto ainda fornece uma amostra representativa das solicitações que a sua aplicação serviu. A regra de amostragem padrão é uma solicitação por segundo e 5% de solicitações adicionais. Esta taxa de amostragem não pode ser configurada para funções do Lambda.

Quando o rastreamento ativo está habilitado, o Lambda registra um rastreamento para um subconjunto de invocações. Lambda registra doisSegmentos do, que cria dois nós no mapa de serviço. O primeiro nó representa o serviço Lambda que recebe a solicitação de invocação. O segundo nó é gravado pelo tempo de execução (p. 19) da função.



É possível instrumentar seu código de função para gravar metadados e rastrear chamadas downstream. Para registrar detalhes sobre chamadas que sua função faz para outros recursos e serviços, use o X-Ray SDK for .NET. Para obter o SDK, adicione os pacotes [AWSXRayRecorder](#) ao arquivo do projeto.

Example [src/blank-csharp/blank-csharp.csproj](#)

```

<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <GenerateRuntimeConfigurationFiles>true</GenerateRuntimeConfigurationFiles>
    <AWSProjectType>Lambda</AWSProjectType>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Newtonsoft.Json" Version="12.0.3" />
    <PackageReference Include="Amazon.Lambda.Core" Version="1.1.0" />
    <PackageReference Include="Amazon.Lambda.SQSEvents" Version="1.1.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="1.7.0" />
    <PackageReference Include="AWSSDK.Core" Version="3.3.104.38" />
    <PackageReference Include="AWSSDK.Lambda" Version="3.3.108.11" />
    <PackageReference Include="AWSXRayRecorder.Core" Version="2.6.2" />
    <PackageReference Include="AWSXRayRecorder.Handlers.AwsSdk" Version="2.7.2" />
  </ItemGroup>
</Project>

```

Para instrumentar clientes do AWS SDK, chame o método `RegisterXRayForAllServices` no código de inicialização.

Example [src/blank-csharp/Function.cs](#): inicializar o X-Ray

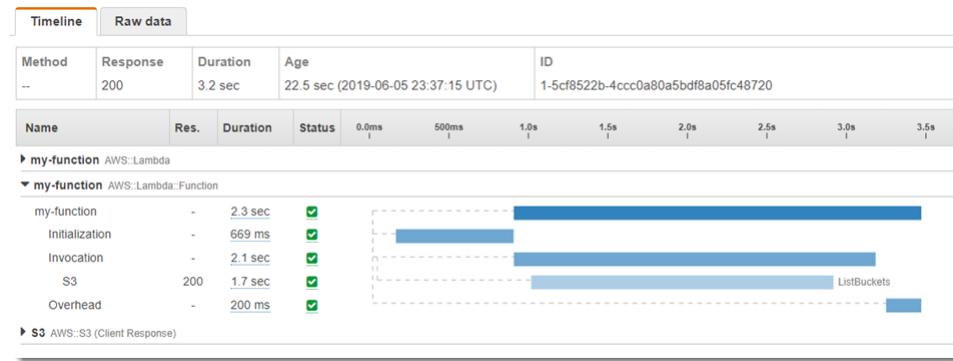
```

static async void initialize() {
    AWSSDKHandler.RegisterXRayForAllServices();
    lambdaClient = new AmazonLambdaClient();
}

```

```
    await callLambda();  
}
```

O exemplo a seguir mostra um rastreamento com 2 segmentos. Ambos são chamados my-function, mas um é do tipo AWS::Lambda e o outro é AWS::Function. O segmento de função é expandido para mostrar seus subsegmentos.



O primeiro segmento representa a solicitação de invocação processada pelo serviço do Lambda. O segundo segmento registra o trabalho realizado pela sua função. O segmento de função tem 3 subsegmentos.

- Inicialização: representa o tempo gasto carregando a função e executando o [código de inicialização \(p. 30\)](#). Esse subsegmento só aparece para o primeiro evento processado por cada instância da função.
- Invocação: representa o trabalho realizado pelo código do handler. Ao instrumentar o código, é possível estender esse subsegmento com subsegmentos adicionais.
- Sobrecarga: representa o trabalho realizado pelo tempo de execução do Lambda como preparação para lidar com o próximo evento.

Você também pode instrumentar clientes HTTP, registrar consultas SQL e criar subsegmentos personalizados com anotações e metadados. Para obter mais informações, consulte [The X-Ray SDK for .NET](#) no Guia do desenvolvedor do AWS X-Ray.

Seções

- [Habilitar o rastreamento ativo com a API do Lambda \(p. 690\)](#)
- [Habilitar o rastreamento ativo com o AWS CloudFormation \(p. 691\)](#)

Habilitar o rastreamento ativo com a API do Lambda

Para gerenciar a configuração de rastreamento com a AWS CLI ou com o AWS SDK, use as seguintes operações de API:

- [UpdateFunctionConfiguration \(p. 974\)](#)
- [GetFunctionConfiguration \(p. 851\)](#)
- [CreateFunction \(p. 796\)](#)

O exemplo de comando da AWS CLI a seguir habilita o rastreamento ativo em uma função chamada my-function.

```
aws lambda update-function-configuration --function-name my-function \
```

```
--tracing-config Mode=Active
```

O modo de rastreamento faz parte da configuração específica da versão que é bloqueada quando você publica uma versão da função. Não é possível alterar o modo de rastreamento em uma versão publicada.

Habilitar o rastreamento ativo com o AWS CloudFormation

Para habilitar o rastreamento ativo em um recurso `AWS::Lambda::Function` em um modelo do AWS CloudFormation, use a propriedade `TracingConfig`.

Example [function-inline.yml](#): configuração de rastreamento

```
Resources:  
  function:  
    Type: AWS::Lambda::Function  
    Properties:  
      TracingConfig:  
        Mode: Active  
      ...
```

Para um recurso do AWS Serverless Application Model (AWS SAM) `AWS::Serverless::Function`, use a propriedade `Tracing`.

Example [template.yml](#): configuração de rastreamento

```
Resources:  
  function:  
    Type: AWS::Serverless::Function  
    Properties:  
      Tracing: Active  
      ...
```

Construir funções do Lambda com o PowerShell

As seções a seguir explicam como padrões comuns de programação e conceitos fundamentais são aplicados ao criar o código da função do Lambda no PowerShell.

Tempos de execução do .NET

Nome	Identifier	Sistema operacional	
.NET Core 3.1	dotnetcore3.1	Amazon Linux 2	
.NET Core 2.1	dotnetcore2.1	Amazon Linux	

O Lambda fornece as seguintes aplicações de exemplo para o tempo de execução do PowerShell:

- [blank-powershell](#): uma função do PowerShell que mostra o uso do registro em log, as variáveis de ambiente e o AWS SDK.

Antes de começar, você deve primeiro configurar um ambiente de desenvolvimento do PowerShell. Para obter instruções sobre como fazer isso, consulte [Definindo um ambiente de desenvolvimento do PowerShell \(p. 693\)](#).

Para saber mais sobre como usar o módulo AWSLambdaPSCore para fazer download de projetos de amostra do PowerShell usando modelos, criar pacotes de implantação do PowerShell e implantar funções do PowerShell na Nuvem AWS, consulte [Implantar funções do Lambda para PowerShell com arquivos .zip \(p. 694\)](#).

Tópicos

- [Definindo um ambiente de desenvolvimento do PowerShell \(p. 693\)](#)
- [Implantar funções do Lambda para PowerShell com arquivos .zip \(p. 694\)](#)
- [AWS LambdaManipulador de função do no PowerShell \(p. 696\)](#)
- [AWS LambdaObjeto de contexto do no PowerShell \(p. 697\)](#)
- [AWS LambdaRegistro em log da função do no PowerShell \(p. 698\)](#)
- [AWS LambdaErros de função do no PowerShell \(p. 703\)](#)

Definindo um ambiente de desenvolvimento do PowerShell

O Lambda fornece um conjunto de ferramentas e bibliotecas para o tempo de execução do PowerShell. Para obter instruções de instalação, consulte [Lambda tools for Powershell](#) no GitHub.

O módulo AWSLambdaPSCore inclui os cmdlets a seguir para ajudar a criar e publicar as funções Lambda para PowerShell:

- Get-AWSPowerShellLambdaTemplate: retorna uma lista de modelos de conceitos básicos.
- New-AWSPowerShellLambda: cria um script do PowerShell inicial com base em um modelo.
- Publish-AWSPowerShellLambda: publica um determinado script do PowerShell para o Lambda.
- New-AWSPowerShellLambdaPackage: cria um pacote de implantação do Lambda que pode ser usado em um sistema CI/CD para implantação.

Implantar funções do Lambda para PowerShell com arquivos .zip

Um pacote de implantação do runtime para PowerShell conterá seu script do PowerShell, os módulos do PowerShell exigidos por esse script e os assemblies requeridos para hospedar o PowerShell Core.

Criar a função do Lambda

Para começar a gravar e invocar um script do PowerShell com o Lambda, você pode usar o cmdlet `New-AWSPowerShellLambda` para criar um script inicial com base em um modelo. Você pode usar o cmdlet `Publish-AWSPowerShellLambda` para implantar seu script no Lambda. Em seguida, você pode testar seu script na linha de comando ou no console do Lambda.

Para criar um novo script do PowerShell, fazer upload dele e testá-lo, siga este procedimento:

1. Execute o seguinte comando para exibir a lista de modelos disponíveis:

```
PS C:\> Get-AWSPowerShellLambdaTemplate

Template          Description
-----
Basic            Bare bones script
CodeCommitTrigger Script to process AWS CodeCommit Triggers
...
```

2. Para criar um script de amostra com base no modelo `Basic`, execute o seguinte comando:

```
New-AWSPowerShellLambda -ScriptName MyFirstPSScript -Template Basic
```

Um novo arquivo chamado `MyFirstPSScript.ps1` é criado em um novo subdiretório do diretório atual. O nome do diretório é baseado no parâmetro `-ScriptName`. Você pode usar o parâmetro `-Directory` para escolher um diretório alternativo.

É possível ver que o novo arquivo tem o seguinte conteúdo:

```
# 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. Para ver como as mensagens de log do script do PowerShell são enviadas ao Amazon CloudWatch Logs, exclua as barras de comentário da linha `Write-Host` do script de amostra.

Para demonstrar como você pode retornar dados das suas funções do Lambda, adicione uma nova linha no final do script com `$PSVersionTable`. Isso adiciona o `$PSVersionTable` ao pipeline do PowerShell. Depois que o script PowerShell estiver concluído, o último objeto no pipeline do PowerShell será os dados de retorno da função do Lambda. `$PSVersionTable` é uma variável global do PowerShell que também fornece informações sobre o ambiente de execução.

Depois de fazer essas alterações, as duas últimas linhas do script de amostra terão esta aparência:

```
Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)  
$PSVersionTable
```

4. Depois de editar o arquivo `MyFirstPSScript.ps1`, altere o diretório para o local do script. Em seguida, execute o seguinte comando para publicar o script no Lambda:

```
Publish-AWSPowerShellLambda -ScriptPath .\MyFirstPSScript.ps1 -Name MyFirstPSScript -  
Region us-east-2
```

Observe que o parâmetro `-Name` especifica o nome da função do Lambda, que aparece no console do Lambda. Você pode usar essa função para chamar seu script manualmente.

5. Invoca sua função usando o comando AWS Command Line Interface da AWS CLI (`invoke`).

```
> aws lambda invoke --function-name MyFirstPSScript out
```

AWS LambdaManipulador de função do no PowerShell

Quando uma função do Lambda é invocada, o manipulador do Lambda chama o script do PowerShell.

Quando o script do PowerShell é invocado, as seguintes variáveis são predefinidas:

- **\$LambdaInput** : um PSObject que contém a entrada para o manipulador. Essa entrada pode ser os dados do evento (publicados por uma origem de evento) ou uma entrada personalizada fornecida por você, tal como uma string ou qualquer objeto de dados personalizado.
- **\$LambdaContext**: um objeto Amazon.Lambda.Core.ILambdaContext que você pode usar para acessar informações sobre a invocação atual, como o nome da função atual, limite de memória, tempo de execução restante e registro em log.

Por exemplo, considere o código de exemplo em PowerShell a seguir.

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}
Write-Host 'Function Name:' $LambdaContext.FunctionName
```

Esse script retorna a propriedade FunctionName que é obtida a partir da variável \$LambdaContext.

Note

É necessário usar a instrução `#Requires` nos seus scripts do PowerShell para indicar os módulos dos quais os seus scripts dependem. Essa declaração executa duas tarefas importantes. 1) Ele comunica a outros desenvolvedores quais módulos o script usa e 2) identifica os módulos dependentes que as ferramentas do AWS PowerShell precisam empacotar com o script como parte da implantação. Para obter mais informações sobre a instrução `#Requires` no PowerShell, consulte [About requires](#). Para obter mais informações sobre os pacotes de implantação do PowerShell, consulte [Implantar funções do Lambda para PowerShell com arquivos .zip \(p. 694\)](#). Quando a sua função do Lambda para PowerShell usa cmdlets do AWS PowerShell, certifique-se de definir uma instrução `#Requires` que faça referência ao módulo `AWSPowerShell.NetCore`, que oferece suporte ao PowerShell Core e não ao módulo `AWSPowerShell`, que apenas oferece suporte para o Windows PowerShell. Além disso, certifique-se de usar a versão 3.3.270.0 ou mais recente do `AWSPowerShell.NetCore`, que otimiza o processo de importação de cmdlets. Se você usar uma versão mais antiga, haverá mais partidas a frio. Para obter mais informações, consulte [AWSTools for PowerShell](#).

Retorno de dados

Algumas invocações do Lambda são destinadas a retornar dados ao chamador. Por exemplo, se uma invocação tiver ocorrido em resposta a uma solicitação do API Gateway, nossa função do Lambda precisará retornar essa resposta. Para o PowerShell Lambda, o último objeto adicionado ao pipeline do PowerShell são os dados de retorno da invocação do Lambda. Se o objeto for uma string, os dados serão retornados no estado em que se encontram. Caso contrário, o objeto será convertido em JSON usando o cmdlet `ConvertTo-Json`.

Por exemplo, considere a seguinte instrução PowerShell, que adiciona `$PSVersionTable` ao pipeline do PowerShell:

```
$PSVersionTable
```

Depois que o script PowerShell estiver finalizado, o último objeto no pipeline do PowerShell será os dados de retorno da função do Lambda. `$PSVersionTable` é uma variável global do PowerShell que também fornece informações sobre o ambiente de execução.

AWS LambdaObjeto de contexto do no PowerShell

Quando o Lambda executa a função, ele transmite informações de contexto, disponibilizando uma variável `$LambdaContext` para o [handler \(p. 696\)](#). Essa variável fornece métodos e propriedades com informações sobre a invocação, a função e o ambiente de execução.

Propriedades de contexto

- `FunctionName`: o nome da função do Lambda.
- `FunctionVersion`: a [versão \(p. 106\)](#) da função.
- `InvokedFunctionArn`: o nome do recurso da Amazon (ARN) usado para invocar a função. Indica se o invocador especificou um alias ou número de versão.
- `MemoryLimitInMB`: a quantidade de memória alocada para a função.
- `AwsRequestId`: o identificador da solicitação de invocação.
- `LogGroupName`: o grupo de logs da função.
- `LogStreamName`: a transmissão de log para a instância da função.
- `RemainingTime`: o número de milissegundos restantes antes do tempo limite da execução.
- `Identity`: (aplicativos móveis) informações sobre a identidade do Amazon Cognito que autorizou a solicitação.
- `ClientContext`: (aplicativos móveis) contexto do cliente fornecido ao Lambda pela aplicação cliente.
- `Logger`: o [objeto do logger \(p. 698\)](#) da função.

O trecho de código PowerShell a seguir mostra uma função de manipulador simples que imprime algumas das informações de contexto.

```
#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
```

AWS Lambda Registro em log da função do no PowerShell

O AWS Lambda monitora automaticamente as funções do Lambda em seu nome e envia métricas da função para o Amazon CloudWatch. Sua função do Lambda vem com um grupo de logs do CloudWatch Logs e uma transmissão de logs para cada instância de sua função. O ambiente do tempo de execução do Lambda envia detalhes sobre cada invocação à transmissão de logs e transmite os logs e outras saídas do código de sua função.

Esta página descreve como produzir a saída de logs usando o código de sua função do Lambda ou acessar os logs usando a AWS Command Line Interface, o console do Lambda ou o console do CloudWatch.

Seções

- [Criar uma função que retorna logs \(p. 698\)](#)
- [Usar o console do Lambda \(p. 699\)](#)
- [Usando o console do CloudWatch \(p. 699\)](#)
- [Usar a AWS Command Line Interface \(AWS CLI\) \(p. 700\)](#)
- [Excluir logs \(p. 702\)](#)

Criar uma função que retorna logs

Para os logs de saída do código da função, você pode usar cmdlets no [utilitário Microsoft.PowerShell](#), ou em qualquer módulo de registro em log que grave no `stdout` ou no `stderr`. O exemplo a seguir usa `Write-Host`.

Example `function/Handler.ps1`: registro em log

```
#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 do 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_dotnetcore2.1_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: 4000 ms Memory Size: 512 MB Max Memory Used: 367 MB Init Duration: 5960.19 ms  
XRAY TraceId: 1-5e843da6-733cxmpl7d0c3c020510040 SegmentId: 3913xmpl20999446 Sampled: true
```

O tempo de execução do .NET registra em log as linhas START, END e REPORT para cada invocação. A linha do relatório fornece os seguintes detalhes.

Registro em log de relatório

- RequestId: o ID de solicitação exclusivo para a invocação.
- Duração: a quantidade de tempo que o método de manipulador da função gastou processando o evento.
- Duração faturada: a quantia de tempo faturada para a invocação.
- Tamanho da memória: a quantidade de memória alocada para a função.
- Memória máxima utilizada: a quantidade de memória utilizada pela função.
- Duração inicial: para a primeira solicitação atendida, a quantidade de tempo que o tempo de execução levou para carregar a função e executar o código fora do método do manipulador.
- XRAY Traceld: para solicitações rastreadas, o [ID de rastreamento do AWS X-Ray \(p. 477\)](#).
- SegmentId: para solicitações rastreadas, o ID do segmento do X-Ray.
- Amostragem: para solicitações rastreadas, o resultado da amostragem.

Usar o console do Lambda

Você pode usar o console do Lambda para exibir a saída do log depois de invocar uma função do Lambda. Para obter mais informações, consulte [Acessar o Amazon CloudWatch Logs para o AWS Lambda \(p. 720\)](#).

Usando o console do CloudWatch

Você pode usar o console do Amazon CloudWatch para exibir registros de todas as invocações da função do Lambda.

Para visualizar logs no console do CloudWatch

1. No console do Amazon CloudWatch, abra a [página Log groups](#) (Grupos de log).
2. Escolha o grupo de logs de sua função (/aws/lambda/**nome-de-sua-função**).
3. Escolha um stream de logs.

Cada fluxo de log corresponde a uma [instância da sua função \(p. 219\)](#). Uma transmissão de logs é exibida quando você atualiza sua função do Lambda e quando mais instâncias são criadas para lidar

com várias invocações simultâneas. Para localizar logs de uma invocação específica, recomendamos instrumentar sua função com AWS X-Ray. O X-Ray registra detalhes sobre a solicitação e o stream de logs no rastreamento.

Para usar uma aplicação de exemplo que correlaciona os logs e os rastreamentos com o X-Ray, consulte [Aplicativo de exemplo de processador de erros para o AWS Lambda \(p. 500\)](#).

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Você pode usar a [AWS CLI](#) para recuperar logs de uma invocação usando a opção de comando `--log-type`. A resposta contém um campo `LogResult` com até 4 KB de logs codificados em base64 obtidos da invocação.

Example recuperar um ID de log

O exemplo a seguir mostra como recuperar um ID de log do campo `LogResult` para uma função chamada `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail
```

Você deve ver a saída a seguir:

```
{  
    "StatusCode": 200,  
    "LogResult":  
        "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...".  
    "ExecutedVersion": "$LATEST"  
}
```

Example decodificar os logs

No mesmo prompt de comando, use o utilitário `base64` para decodificar os logs. O exemplo a seguir mostra como recuperar logs codificados em base64 de `my-function`.

```
aws lambda invoke --function-name my-function out --log-type Tail \  
--query 'LogResult' --output text | base64 -d
```

Você deve ver a saída a seguir:

```
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
```

O utilitário `base64` está disponível no Linux, macOS e [Ubuntu no Windows](#). Os usuários do macOS precisam usar `base64 -D`.

Example get-logs.sh script

No mesmo prompt de comando, use o script a seguir para fazer download dos últimos cinco eventos de log. O script usa sed para remover as aspas do arquivo de saída e fica inativo por 15 segundos para que os logs tenham tempo de ficar disponíveis. A saída inclui a resposta do Lambda, e a saída do comando get-log-events.

Copie o conteúdo do exemplo de código a seguir e salve no diretório de seu projeto do Lambda como get-logs.sh.

O comando cli-binary-format é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

```
#!/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 $(cat
out) --limit 5
```

Example macOS e Linux (somente)

No mesmo prompt de comando, os usuários do macOS e do Linux podem precisar executar o comando a seguir para garantir que o script seja executável.

```
chmod +x get-logs.sh
```

Example recuperar os últimos cinco eventos de log

No mesmo prompt de comando, execute o script a seguir para obter os últimos cinco eventos de log.

```
./get-logs.sh
```

Você deve ver a saída a seguir:

```
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
{
    "events": [
        {
            "timestamp": 1559763003171,
            "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version: $LATEST\n",
            "ingestionTime": 1559763003309
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf \tINFO\tENVIRONMENT VARIABLES\r{\r\t\t\"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\", \r\t\t...",
            "ingestionTime": 1559763018353
        },
        {
            "timestamp": 1559763003173,
            "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf \tINFO\tEVENT\r{\r\t\t\"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"  
}
```

Excluir logs

Os grupos de logs não são excluídos automaticamente excluídos quando você exclui uma função. Para evitar armazenar logs indefinidamente, exclua o grupo de logs ou[Configurar um período de retenção](#)após o qual os logs são excluídos automaticamente.

AWS Lambda Erros de função do no PowerShell

Quando o código gera um erro, o Lambda gera uma representação JSON do erro. Esse documento de erro aparece no log de invocação e, para invocações síncronas, na saída.

Esta página descreve como exibir erros de invocação da função do Lambda para o tempo de execução do Powershell usando o console do Lambda e a AWS CLI.

Seções

- [Syntax \(p. 703\)](#)
- [Como funcionam \(p. 704\)](#)
- [Usar o console do Lambda \(p. 704\)](#)
- [Usar a AWS Command Line Interface \(AWS CLI\) \(p. 705\)](#)
- [Tratamento de erros em outros serviços da AWS \(p. 705\)](#)
- [Próximas etapas \(p. 706\)](#)

Syntax

Considere a seguinte instrução de exemplo de script do PowerShell:

```
throw 'The Account is not found'
```

Quando você invoca essa função do Lambda, ela lança um erro de finalização, e o AWS Lambda retorna a seguinte mensagem de erro:

```
{  
  "errorMessage": "The Account is not found",  
  "errorType": "RuntimeException"  
}
```

Observe que `errorType` é `RuntimeException`, que é a exceção padrão lançada pelo PowerShell. Você pode usar tipos de erros personalizados lançando o erro da seguinte maneira:

```
throw @{ 'Exception'='AccountNotFound'; 'Message'='The Account is not found' }
```

A mensagem de erro é serializada com `errorType` definido como `AccountNotFound`:

```
{  
  "errorMessage": "The Account is not found",  
  "errorType": "AccountNotFound"  
}
```

Se você não precisar de uma mensagem de erro, poderá lançar uma string no formato de um código de erro. O formato do código de erro requer que a string comece com um caractere e apenas letras e dígitos depois, sem espaços ou símbolos.

Por exemplo, se sua função do Lambda contém o seguinte:

```
throw 'AccountNotFound'
```

O erro é serializado desta maneira:

```
{  
  "errorMessage": "AccountNotFound",  
  "errorType": "AccountNotFound"  
}
```

Como funcionam

Ao invocar uma função do Lambda, o Lambda recebe a solicitação de invocação e valida as permissões de sua função de execução, verifica se o documento do evento é um documento JSON válido e verifica valores de parâmetros.

Se a solicitação for validada, o Lambda a envia para uma instância da função. O ambiente de [tempo de execução do Lambda \(p. 214\)](#) converte o documento do evento em um objeto e o transmite ao handler da função.

Se o Lambda encontra um erro, ele retorna um tipo de exceção, uma mensagem e o código HTTP do status que indica a causa do erro. O cliente ou o serviço que invocou a função do Lambda pode processar o erro de maneira programática ou transmiti-lo a um usuário final. O comportamento correto de tratamento de erros depende do tipo de aplicativo, do público e da origem do erro.

A lista a seguir descreve o intervalo de códigos de status que você pode receber do Lambda.

2xx

Um erro da série 2xx com um cabeçalho `X-Amz-Function-Error` na resposta indica um erro de tempo de execução ou de função do Lambda. Um código de status da série 2xx indica que o Lambda aceitou a solicitação, mas em vez de um código de erro, o Lambda indica o erro incluindo o cabeçalho `X-Amz-Function-Error` na resposta.

4xx

Um erro da série 4xx indica um erro que o cliente ou serviço que fez a invocação pode corrigir modificando a solicitação, solicitando permissão ou tentando a solicitação novamente. Os erros da série 4xx diferentes de 429 geralmente indicam um erro na solicitação.

5xx

Um erro da série 5xx indica um problema com o Lambda ou com a configuração/recursos da função. Os erros da série 5xx podem indicar uma condição temporária que pode ser resolvida sem nenhuma ação do usuário. O cliente ou serviço que fez a invocação não podem solucionar esses problemas, mas o proprietário de uma função do Lambda pode ser capaz de corrigi-los.

Para obter uma lista completa de erros de invocação, consulte [Erros de InvokeFunction \(p. 877\)](#).

Usar o console do Lambda

Você pode invocar sua função no console do Lambda configurando um evento de teste e visualizando a saída. A saída é capturada pelos logs de execução da função e, quando o [rastreamento ativo \(p. 477\)](#) está habilitado, pelo AWS X-Ray.

Para invocar uma função no console do Lambda

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Test (Testar).
4. Selecione New event (Novo evento) e escolha um Event template (Modelo de evento) na lista suspensa.

5. Insira um nome para o evento de teste.
6. Digite o JSON para o evento de teste.
7. Escolha Create event (Criar evento).
8. Escolha Invoke (Invocar).

O console do Lambda invoca sua função [de forma síncrona \(p. 158\)](#) e exibe o resultado. Para ver a resposta, os logs e outras informações, expanda a seção Details (Detalhes).

Usar a AWS Command Line Interface (AWS CLI)

O AWS CLI é uma ferramenta de código aberto que permite interagir com os serviços do AWS usando comandos no shell da linha de comando. Para executar as etapas desta seção, você deve ter o seguinte:

- [AWS CLI – Instalar a versão 2](#)
- [AWS CLI – Configuração rápida com aws configure](#)

Quando você invoca uma função do Lambda na AWS CLI, a AWS CLI divide a resposta em dois documentos. A resposta da AWS CLI é exibida no prompt de comando. Se ocorreu um erro, a resposta contém um campo `FunctionError`. A resposta ou o erro de invocação retornado pela função é gravado no arquivo de saída. Por exemplo, o `output.json` ou o `output.txt`.

O exemplo de comando `invoke` a seguir demonstra como invocar uma função e escrever a resposta de invocação em um arquivo `output.txt`.

```
aws lambda invoke \
    --function-name my-function \
    --cli-binary-format raw-in-base64-out \
    --payload '{"key1": "value1", "key2": "value2", "key3": "value3"}' output.txt
```

O comando `cli-binary-format` é necessário se você estiver usando a AWS CLI versão 2. Também é possível configurar essa opção no [arquivo de configuração da AWS CLI](#).

Você deve ver a resposta da AWS CLI em seu prompt de comando:

```
{
  "StatusCode": 200,
  "FunctionError": "Unhandled",
  "ExecutedVersion": "$LATEST"
}
```

Você deve ver a resposta de invocação de função no arquivo `output.txt`. Também é possível visualizar a saída no mesmo prompt de comando usando:

```
cat output.txt
```

Você deve ver a resposta de invocação no prompt de comando.

O Lambda também grava até 256 KB do objeto de erro nos logs de função. Para obter mais informações, consulte [AWS Lambda Registro em log da função do no PowerShell \(p. 698\)](#).

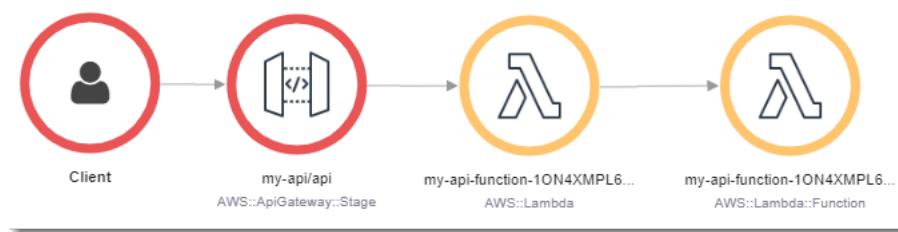
Tratamento de erros em outros serviços da AWS

Quando outro serviço da AWS invoca sua função, o serviço escolhe o tipo de invocação e o comportamento de repetição. Os serviços da AWS podem invocar sua função em um agendamento, em

resposta a um evento de ciclo de vida em um recurso ou para atender a uma solicitação de um usuário. Alguns serviços invocam funções de forma assíncrona e permitem que o Lambda trate erros, enquanto outros fazem novas tentativas ou transmitem os erros de volta ao usuário.

Por exemplo, o API Gateway trata todos os erros de invocação e função como erros internos. Se a API do Lambda rejeitar a solicitação de invocação, o API Gateway retornará um código de erro 500. Se a função é executada, mas retorna um erro ou retorna uma resposta no formato errado, o API Gateway retorna um código de erro 502. Para personalizar a resposta de erro, é necessário detectar erros no código e formatar uma resposta no formato necessário.

Recomendamos usar AWS X-Ray para determinar a fonte de um erro e a respectiva causa. O X-Ray permite localizar qual componente encontrou um erro e ver detalhes sobre os erros. O exemplo a seguir mostra um erro de função que resultou em uma resposta 502 do API Gateway.



Para obter mais informações, consulte [Usar o AWS Lambda com o AWS X-Ray \(p. 477\)](#).

Próximas etapas

- Saiba como mostrar eventos de registro em log para sua função do Lambda na página [the section called “Registro em log” \(p. 698\)](#).

Monitoramento e solução de problemas de aplicações do Lambda

O AWS Lambda pode ser integrado a outros serviços da AWS para ajudar você a monitorar e solucionar problemas de suas funções do Lambda. O Lambda monitora automaticamente as funções do Lambda em seu nome e gera relatórios sobre métricas por meio do Amazon CloudWatch. Para ajudar você a monitorar seu código quando ele estiver sendo executado, o Lambda rastreia automaticamente o número de solicitações, a duração de chamadas por solicitação e o número de solicitações que resultam em erro.

Você pode usar outros serviços do AWS para solucionar problemas de suas funções do Lambda. Esta seção descreve como usar esses serviços da AWS para monitorar, rastrear, depurar e resolver problemas de suas funções e aplicações do Lambda.

Para obter mais informações sobre monitoramento de aplicações do Lambda, consulte [Monitoramento e observabilidade](#) no Guia do operador do Lambda.

Seções

- [Funções de monitoramento no console do AWS Lambda \(p. 708\)](#)
- [Usar o Lambda Insights no Amazon CloudWatch \(p. 710\)](#)
- [Trabalhar com métricas de função do AWS Lambda \(p. 717\)](#)
- [Acessar o Amazon CloudWatch Logs para o AWS Lambda \(p. 720\)](#)
- [Usando o CodeGuru Profiler com sua função do Lambda \(p. 721\)](#)
- [Exemplo de fluxos de trabalho usando outros serviços do AWS \(p. 723\)](#)

Funções de monitoramento no console do AWS Lambda

O AWS Lambda monitora funções em seu nome e envia as métricas para o Amazon CloudWatch. O console do Lambda cria gráficos para monitoramento dessas métricas e os mostra na página Monitoring (Monitoramento) para cada função do Lambda.

Esta página descreve os conceitos básicos do uso do console do Lambda para exibir métricas de função, incluindo o total de solicitações, a duração e taxas de erro.

Pricing

O CloudWatch tem um nível gratuito vitalício. Além do limite do nível gratuito, o CloudWatch cobra por métricas, painéis, alarmes, logs e insights. Para obter mais informações, consulte [Definição de preços do Amazon CloudWatch](#).

Usar o console do Lambda

Você pode usar o painel de monitoramento do console do Lambda para monitorar suas funções e aplicações do Lambda.

Como monitorar uma função

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Monitoring.

Tipos de gráficos de monitoramento

A seção a seguir descreve os gráficos de monitoramento do console do Lambda.

Gráficos de monitoramento do Lambda

- Invocations (Invocações): o número de vezes que a função foi invocada.
- Duration (Duração): tempos de execução médio, mínimo e máximo.
- Error count and success rate (%) (Contagem de erros e taxa de êxito (%)): o número de erros e a porcentagem de execuções que foram concluídas sem erros.
- Throttles (Controladores de utilização): o número de vezes que a execução falhou devido a limites de simultaneidade.
- IteratorAge: para fontes de eventos de fluxo, a idade do último item no lote quando o Lambda o recebeu e invocou a função.
- Async delivery failures (Falhas de entrega assíncrona): o número de erros que ocorreram quando o Lambda tentou gravar em um destino ou em uma fila de mensagens mortas.
- Concurrent executions (Execuções simultâneas): o número de instâncias da função que estão processando eventos.

Visualizando gráficos no console do Lambda

A seção a seguir descreve como exibir gráficos de monitoramento do CloudWatch no console do Lambda e abrir o painel de métricas do CloudWatch.

Para visualizar gráficos de monitoramento de uma função

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Monitoring.
4. Escolha entre os intervalos de tempo predefinidos ou escolha um intervalo de tempo personalizado.
5. Para ver a definição de um gráfico no CloudWatch, selecione os três pontos verticais e, em seguida, selecione View in metrics (Exibir em métricas) para abrir o painel Metrics (Métricas) no CloudWatch.

Exibindo consultas no console do CloudWatch Logs

A seção a seguir descreve como exibir e adicionar relatórios de insights do CloudWatch Logs em um painel personalizado no console do CloudWatch Logs.

Para exibir relatórios de uma função

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Monitoring.
4. Escolha View logs in CloudWatch (Exibir logs no CloudWatch).
5. Selecione Exibir no Logs Insights.
6. Escolha entre os intervalos de tempo predefinidos ou escolha um intervalo de tempo personalizado.
7. Selecione Run query (Executar consulta).
8. (Opcional) Selecione Salvar.

Próximas etapas

- Saiba mais sobre as métricas que o Lambda registra e envia para o CloudWatch no [Trabalhar com métricas de função do AWS Lambda \(p. 717\)](#).
- Saiba como usar o os insights do Lambda para coletar e agregar métricas e registros de performance de tempo de execução da função do Lambda no [Usar o Lambda Insights no Amazon CloudWatch \(p. 710\)](#).

Usar o Lambda Insights no Amazon CloudWatch

O Lambda Insights do Amazon CloudWatch coleta e agrupa métricas e logs de performance do tempo de execução da função do Lambda para as aplicações sem servidor. Esta página descreve como ativar e usar o Lambda Insights para diagnosticar problemas em suas funções do Lambda.

Seções

- [Como o Lambda Insights monitora aplicações sem servidor \(p. 710\)](#)
- [Pricing \(p. 710\)](#)
- [Tempos de execução compatíveis \(p. 710\)](#)
- [Ativar o Lambda Insights no console do Lambda \(p. 710\)](#)
- [Ativação do Lambda Insights por programação \(p. 711\)](#)
- [Usando o painel do Lambda Insights \(p. 711\)](#)
- [Exemplo de fluxo de trabalho para detectar anomalias de função \(p. 713\)](#)
- [Exemplo de fluxo de trabalho usando consultas para solucionar problemas de uma função \(p. 714\)](#)
- [Próximas etapas \(p. 709\)](#)

Como o Lambda Insights monitora aplicações sem servidor

O Lambda Insights do CloudWatch Lambda é uma solução de monitoramento e solução de problemas para aplicações sem servidor em execução no AWS Lambda. A solução coleta, agrupa e resume métricas no nível do sistema, incluindo tempo da CPU, memória, disco e uso da rede. Ele também coleta, agrupa e resume informações de diagnóstico, como inicializações a frio e desligamentos do operador do Lambda para ajudar a isolar problemas com as funções do Lambda e resolvê-los rapidamente.

Lambda Insights usa um novo CloudWatch Lambda Insights[extensão](#), que é fornecido como [umLambda \(p. 85\)](#). Quando você habilita essa extensão em uma função do Lambda para um formato compatível, ela coleta métricas no nível do sistema e emite um único evento de log de performance para cada invocação dessa função do Lambda. O CloudWatch usa formatação métrica incorporada para extrair métricas dos eventos de log. Para obter mais informações, consulte [Usar extensões do AWS Lambda](#).

A camada do Lambda Insights estende `CreateLogStream` e `PutLogEvents` para o grupo de logs `/aws/lambda-insights/`.

Pricing

Para cada função do Lambda habilitada para o Lambda Insights, você paga apenas pelo que usar com relação a métricas e logs. Para obter um exemplo de preço, consulte [Preço do Amazon CloudWatch](#).

Tempos de execução compatíveis

Você pode usar o Lambda Insights com qualquer um dos tempos de execução que oferecem suporte para [extensões do Lambda \(p. 228\)](#).

Ativar o Lambda Insights no console do Lambda

É possível habilitar o monitoramento aprimorado do Lambda Insights em funções do Lambda novas e existentes. Quando você ativa o Lambda Insights em uma função no console do Lambda para um tempo de execução compatível, o Lambda adiciona a [extensão](#) do Lambda Insights à sua função como uma

camada e verifica ou tenta associar a política [CloudWatchLambdaInsightsExecutionRolePolicy](#) à função de execução da função.

Para habilitar o Lambda Insights no console do Lambda

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha a função.
3. Escolha a guia Configuration (Configuração).
4. No painel Monitoring tools (Ferramentas de monitoramento), selecione Edit (Editar).
5. Sob o Lambda Insights, ative o Enhanced monitoring.
6. Escolha Save (Salvar).

Ativação do Lambda Insights por programação

Também é possível habilitar o Lambda Insights usando a AWS Command Line Interface (AWS CLI), a CLI do AWS Serverless Application Model (SAM), o AWS CloudFormation ou o AWS Cloud Development Kit (CDK). Quando você habilita o Lambda Insights programaticamente em uma função para um tempo de execução compatível, o CloudWatch associa a política [CloudWatchLambdaInsightsExecutionRolePolicy](#) à função de execução da sua função.

Para obter mais informações, consulte [Conceitos básicos do Lambda Insights](#) no Manual do usuário do Amazon CloudWatch.

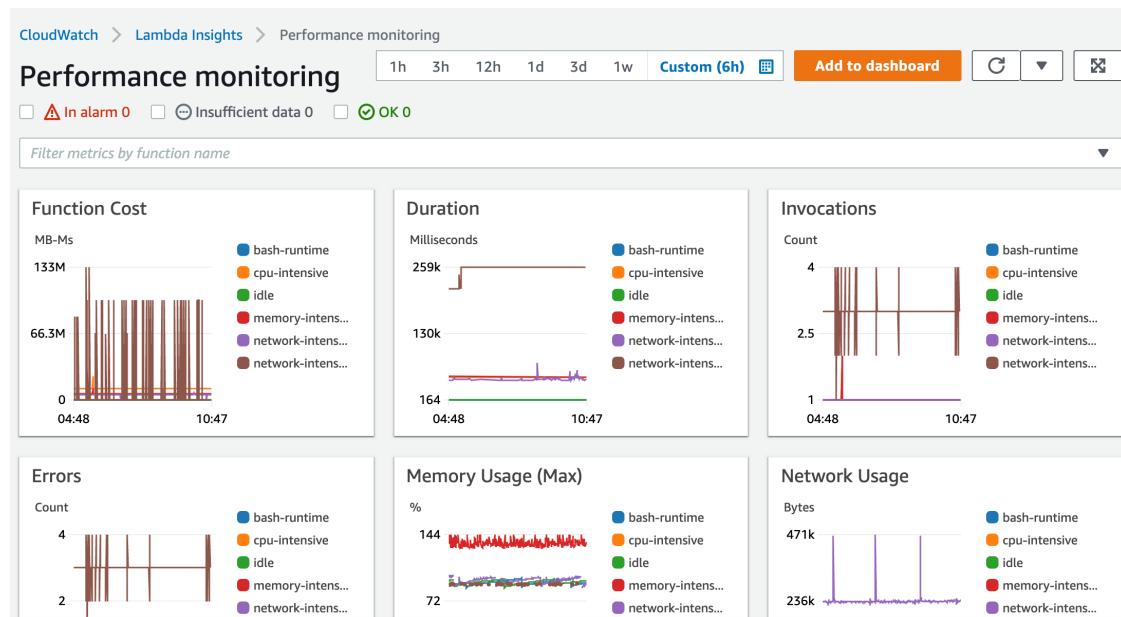
Usando o painel do Lambda Insights

O painel do Lambda Insights tem duas exibições no console do CloudWatch: a visão geral de várias funções e a exibição de função única. A visão geral de várias funções agrupa as métricas de tempo de execução para as funções do Lambda na conta e na região atual da AWS. A visualização de função única mostra as métricas de tempo de execução disponíveis para uma única função do Lambda.

É possível usar a visão geral multifuncional do painel do Lambda Insights no console do CloudWatch para identificar funções do Lambda usadas em excesso e subutilizadas. É possível usar a visualização de função única do painel do Lambda Insights no console do CloudWatch para solucionar problemas de solicitações individuais.

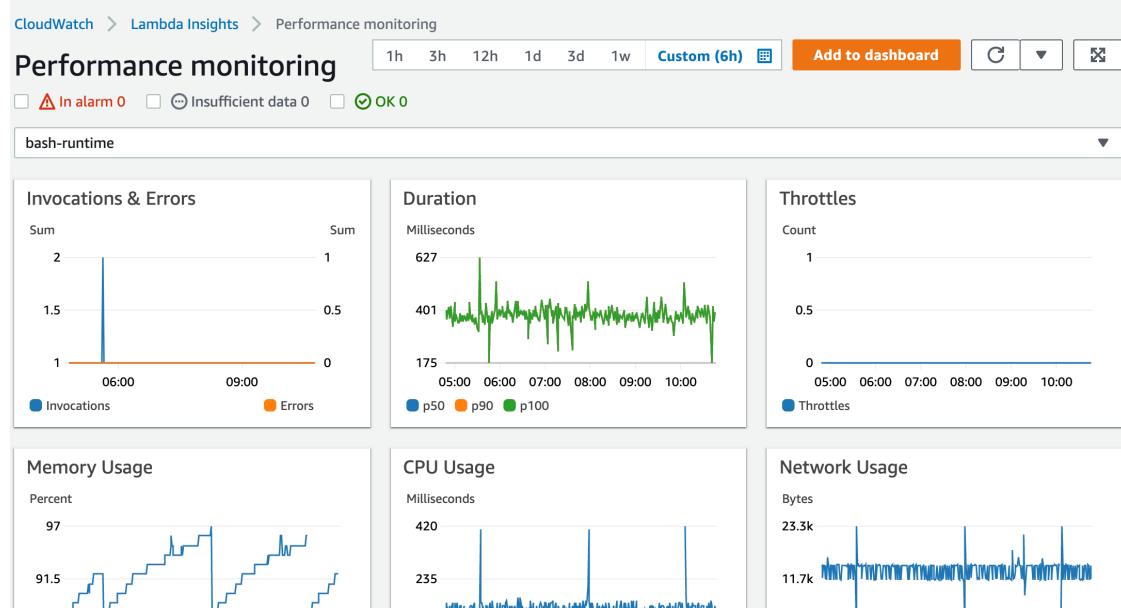
Como visualizar as métricas de tempo de execução de todas as funções

1. Abra a página [Multi-funcion](#) (Várias funções) no console do CloudWatch.
2. Escolha entre os intervalos de tempo predefinidos ou escolha um intervalo de tempo personalizado.
3. (Opcional) Selecione Add to dashboard (Adicionar ao painel) para adicionar os widgets ao painel do CloudWatch.



Como visualizar as métricas de tempo de execução de uma única função

1. Abra a página [Single-function](#) (Função única) no console do CloudWatch.
2. Escolha entre os intervalos de tempo predefinidos ou escolha um intervalo de tempo personalizado.
3. (Opcional) Selecione **Add to dashboard** (Adicionar ao painel) para adicionar os widgets ao painel do CloudWatch.



Para obter mais informações, consulte [Criar e trabalhar com widgets em painéis do CloudWatch](#).

Exemplo de fluxo de trabalho para detectar anomalias de função

É possível usar a visão geral de várias funções no painel Lambda Insights para identificar e detectar anomalias de memória computacional com a função. Por exemplo, se a visão geral de várias funções indicar que uma função está usando uma grande quantidade de memória, você poderá visualizar métricas detalhadas de utilização da memória no painel Memory Usage (Uso de memória). Depois, você pode acessar o painel de métricas para habilitar a detecção de anomalias ou criar um alarme.

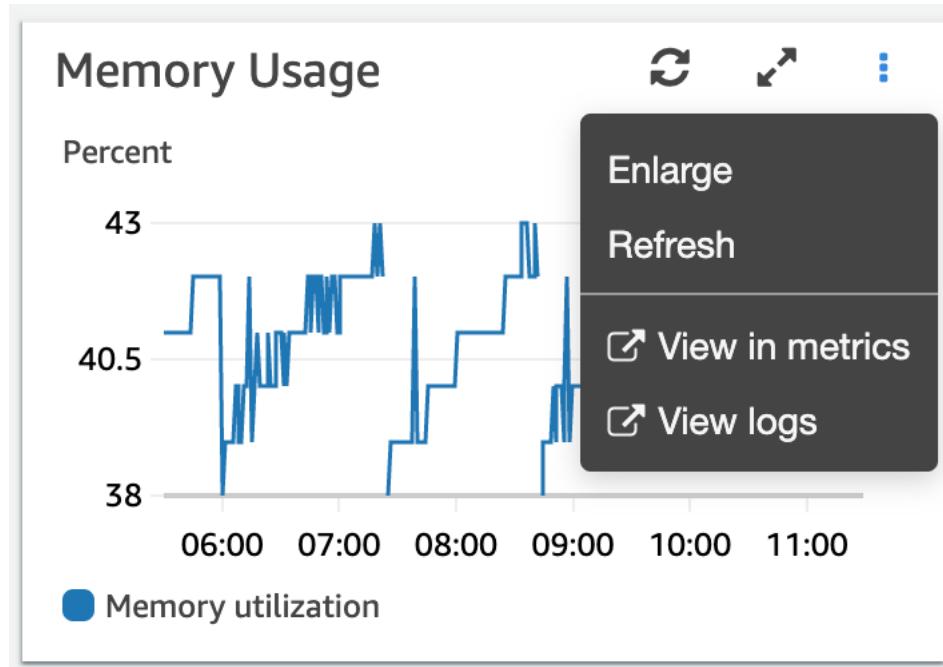
Como habilitar a detecção de anomalias para uma função

1. Abra a página [Multi-funcion](#) (Várias funções) no console do CloudWatch.
2. Em Function summary (Resumo da função), escolha o nome da função.

A visualização de função única é aberta com as métricas de tempo de execução da função.

Function summary (6)							Actions	
	Function name	Invocations	CPU time	Network IO	Max. memory	Cold starts		
	bash-runtime	360	132.9167ms	4770 kB	97%	3		
	cpu-intensive	359	6714.2897ms	4780 kB	43%	4		
	idle	359	120.2507ms	4746 kB	96%	3		
	memory-intensive	358	2385.9497ms	4794 kB	44%	4		
	network-intensive	359	781.0585ms	82008 kB	99%	3		
	network-intensive-vpc	43	2730.6977ms	95 kB	91%	43		

3. No painel Memory Usage (Uso de memória), selecione os três pontos na vertical e selecione View in metrics (Visualizar nas métricas) para abrir o painel Metrics (Métricas).



- Na guia Graphed metrics (Métricas em gráficos), na coluna Actions (Ações), selecione o primeiro ícone para habilitar a detecção de anomalias da função.

All metrics	Graphed metrics (6)	Graph options	Source		
Math expression		Dynamic labels	Statistic: Maximum Period: 1 Minute Remove all		
Label	Details	Statistic	Period	Y Axis	Actions
bash-runtime	LambdaInsights • memory_utilization • function_name: bash-runtime	Maximum	1 Minute	< >	alarm bell refresh
cpu-intensive	LambdaInsights • memory_utilization • function_name: cpu-intensive	Maximum	1 Minute	< >	alarm bell refresh
idle	LambdaInsights • memory_utilization • function_name: idle	Maximum	1 Minute	< >	alarm bell refresh
memory-intensive	LambdaInsights • memory_utilization • function_name: memory-intensive	Maximum	1 Minute	< >	alarm bell refresh

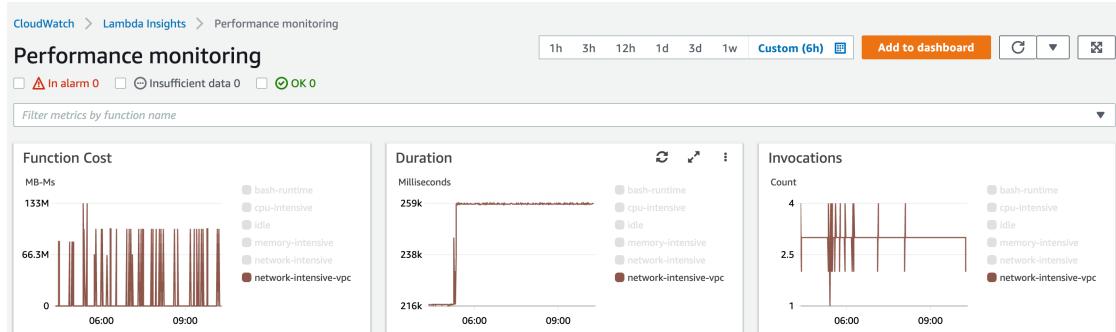
Para obter mais informações, consulte [Usar a detecção de anomalias do CloudWatch](#).

Exemplo de fluxo de trabalho usando consultas para solucionar problemas de uma função

É possível usar a visualização de função única no painel Lambda Insights para identificar a causa raiz de um pico na duração da função. Por exemplo, se a visão geral de várias funções indicar um grande aumento na duração da função, você poderá pausar ou escolher cada função no painel Duration (Duração) para determinar qual função está causando o aumento. Depois, você pode acessar a visualização de função única e revisar os Application logs (Logs de aplicações) para determinar a causa raiz.

Como executar consultas em uma função

- Abra a página [Multi-funcion](#) (Várias funções) no console do CloudWatch.
- No painel Duration (Duração), selecione a função para filtrar as métricas de duração.



- Abra a página [Função única](#).
- Selecione a lista suspensa Filter metrics by function name (Filtrar métricas por nome da função) e selecione a função.
- Para visualizar os Most recent 1000 application logs (1000 logs de aplicações mais recentes), selecione a guia Application logs (Logs de aplicações).
- Revise o Timestamp (Time stamp) e a Message (Mensagem) para identificar a solicitação de invocação que você deseja solucionar.

Most recent 1000 application logs (1000)	
Timestamp	Message
2020-09-30T16:24:36.121-06	0 0 0 0 0 0 0 --:--:-- 0:03:06 --:--:-- 0
2020-09-30T16:24:34.917-06	0 0 0 0 0 0 0 --:--:-- 0:04:15 --:--:-- 0
2020-09-30T16:24:34.120-06	0 0 0 0 0 0 0 --:--:-- 0:03:04 --:--:-- 0
2020-09-30T16:24:33.033-06	0 0 0 0 0 0 0 --:--:-- 0:01:26 --:--:-- 0

7. Para mostrar as Most recent 1000 invocations (1000 invocações mais recentes), selecione a guia Invocations (Invocações).
8. Selecione o Timestamp (Time stamp) ou a Message (Mensagem) para a solicitação de invocação que você deseja solucionar.

Most recent 1000 invocations (1/45)						
Timestamp	Request ID	Trace	Memory %	Network IO	CPU time	Cold start
2020-09-30 16:22:34 (UTC-06:00)	247e6369-3a2b...	-	91%	2 kB	2550ms	Yes
2020-09-30 16:13:39 (UTC-06:00)	311fb438-fa9d-4...	-	90%	2 kB	2340ms	Yes

9. Selecione a lista suspensa View logs (Visualizar logs) e, depois, selecione View performance logs (Visualizar logs de performance).

Uma consulta gerada automaticamente para a função é aberta no painel do Logs Insights.

10. Selecione Run query (Executar consulta) para gerar uma mensagem Logs para a solicitação de invocação.

The screenshot shows the AWS CloudWatch Logs Insights interface. At the top, there is a dropdown menu for 'Select log group(s)' with '/aws/lambda-insights' selected. Below it is a search bar with the same path. A code editor displays a query:

```
1 fields @timestamp, @message
2 | filter function_name = "network-intensive-vpc"
3 | filter request_id = "247e6369-3a2b-4ccf-9e95-fb80c6ba711f"
4 | sort @timestamp desc
```

Below the code editor are three buttons: 'Run query' (highlighted in orange), 'Save', and 'History'.

The main area shows the results of the query. It includes a histogram at the top right with the text 'Showing 1 of 1 records matched (1)' and '1,856 records (2.0 MB) scanned in 4.0s @ 467 records/s (521.7 kB/s)'. The histogram has a single bar reaching the 4 PM mark. Below the histogram is a table with columns '#', '@timestamp', and '@message'. One row is shown, corresponding to the query result:

#	@timestamp	@message
► 1	2020-09-30T16:22:34...	{"cpu_system_time":1520,"shutdown":1,"cpu_user_time":1030,"agent_memory_avg":7487349,"used_memory":...

Próximas etapas

- Saiba como criar um painel do CloudWatch Logs no [Criar um painel](#) no Guia do usuário do Amazon CloudWatch.

- Saiba como adicionar consultas a um painel do CloudWatch Logs em [Adicionar consulta ao painel ou exportar resultados de consulta](#) no Manual do usuário do Amazon CloudWatch.

Trabalhar com métricas de função do AWS Lambda

Quando a função termina o processamento de um evento, o Lambda envia métricas sobre a invocação para o Amazon CloudWatch. É possível criar gráficos e painéis com essas métricas no console do CloudWatch e definir alarmes para responder a alterações na utilização, na performance ou nas taxas de erro. O Lambda envia dados de métricas ao CloudWatch em intervalos de um minuto.

Esta página descreve as métricas de invocação, performance e simultaneidade da função do Lambda disponíveis no console do CloudWatch.

Seções

- [Visualizar métricas no console do CloudWatch \(p. 717\)](#)
- [Tipos de métricas \(p. 717\)](#)

Visualizar métricas no console do CloudWatch

Você pode usar o console do CloudWatch para filtrar e classificar métricas de funções por nome, alias ou versão da função.

Para exibir métricas no console do CloudWatch

1. Abra a [página Metrics](#) (Métricas) (namespace AWS/Lambda) no console do CloudWatch.
2. Escolha uma dimensão.
 - Por nome da função (`FunctionName`): visualize métricas agregadas para todas as versões e aliases de uma função.
 - Por recurso (`Resource`): visualize métricas para uma versão ou um alias de uma função.
 - Por versão executada (`ExecutedVersion`): visualize métricas para uma combinação de alias e versão. Use a dimensão `ExecutedVersion` para comparar taxas de erro para duas versões de uma função que são ambas destinos de um [alias ponderado \(p. 108\)](#).
 - Em todas as funções (nenhum): visualize métricas agregadas para todas as funções na região atual da AWS.
3. Escolha métricas para adicioná-las ao gráfico.

Por padrão, os gráficos usam a estatística `Sum` para todas as métricas. Para escolher uma estatística diferente e personalizar o gráfico, use as opções na guia Graphed métricas (Métricas no gráfico).

Note

O timestamp em uma métrica reflete quando a função foi invocada. Dependendo da duração da execução, isso pode representar vários minutos antes de quando a métrica é emitida. Se, por exemplo, a função tiver um tempo-limite de 10 minutos, para obter métricas precisas, procure mais de 10 minutos no passado.

Para obter mais informações sobre o CloudWatch, consulte o [Guia do usuário do Amazon CloudWatch](#).

Tipos de métricas

A seção a seguir descreve o tipo de métricas disponíveis no console do CloudWatch.

Usar métricas de invocação

As métricas de invocação são indicadores binários do resultado de uma invocação. Por exemplo, se a função retornar um erro, o Lambda enviará a métrica `Errors` com um valor de 1. Para obter uma

contagem do número de erros da função que ocorrem a cada minuto, visualize `Sum` da métrica `Errors` com um período de um minuto.

Você deve visualizar as métricas a seguir com a estatística `Sum`.

Métricas de invocação

- `Invocations`: o número de vezes que o código da função foi executado, incluindo execuções bem-sucedidas e execuções que resultam em um erro de função. As invocações não serão registradas se a solicitação de invocação for limitada ou se resultar em um erro de invocação. Isso é igual ao número de solicitações faturadas.
- `Errors`: o número de invocações que resultam em um erro de função. Os erros de função incluem exceções lançadas pelo código e exceções lançadas pelo tempo de execução do Lambda. O tempo de execução retorna um erro para problemas como tempos limite e erros de configuração. Para calcular a taxa de erro, divida o valor de `Errors` pelo valor de `Invocations`. Observe que o carimbo de data/hora em uma métrica de erro reflete quando a função foi chamada, não quando o erro ocorreu.
- `DeadLetterErrors`: para [invocação assíncrona \(p. 161\)](#), o número de vezes que o Lambda tenta enviar um evento para uma fila de mensagens mortas, mas falha. Os erros de mensagens mortas podem ocorrer devido a erros de permissões, recursos configurados incorretamente ou limites de tamanho.
- `DestinationDeliveryFailures`: para invocação assíncrona, o número de vezes que o Lambda tenta enviar um evento para um [destino \(p. 27\)](#), mas falha. Os erros de entrega podem ocorrer devido a erros de permissão, recursos configurados incorretamente ou limites de tamanho.
- `Throttles`: o número de solicitações de invocação que são limitadas. Quando todas as instâncias da função estão processando solicitações e nenhuma simultaneidade está disponível para aumento da escala na vertical, o Lambda rejeita solicitações adicionais com `TooManyRequestsException`. As solicitações limitadas e outros erros de invocação não contam como `Invocations` ou `Errors`.
- `ProvisionedConcurrencyInvocations`: o número de vezes que o código da função é executado em [simultaneidade provisionada \(p. 113\)](#).
- `ProvisionedConcurrencySpilloverInvocations`: o número de vezes que o código da função é executado em simultaneidade padrão quando toda a simultaneidade provisionada está em uso.

Usar métricas de performance

As métricas de performance fornecem detalhes de performance sobre uma única invocação. Por exemplo, a métrica `Duration` indica a quantidade de tempo, em milissegundos, que a função gasta processando um evento. Para ter uma ideia da velocidade de processamento de eventos da função, visualize essas métricas com a estatística `Average` ou `Max`.

Métricas de performance

- `Duration`: quantidade de tempo que o código da função gasta processando um evento. A duração faturada de uma invocação é o valor de `Duration` arredondado para o milissegundo mais próximo.
- `PostRuntimeExtensionsDuration`: a quantidade cumulativa de tempo que o tempo de execução gasta executando o código para extensões após a conclusão do código de função.
- `IteratorAge`: para os [mapeamentos de fontes de eventos \(p. 170\)](#) que faz a leitura de transmissões, a idade do último registro do evento. A idade é a quantidade de tempo entre quando o fluxo recebe o registro e quando o mapeamento da origem do evento envia o evento à função.

`Duration` também oferece suporte a [estatísticas de percentil](#). Use os percentis para excluir valores de ponto fora da curva que distorcem as estatísticas média e máxima. Por exemplo, a estatística P95 mostra a duração máxima de 95% das execuções, excluindo os 5% mais lentos.

Usar métricas de simultaneidade

O Lambda relata métricas de simultaneidade como uma contagem agregada do número de instâncias que estão processando eventos em uma função, uma versão, um alias ou uma região da AWS. Para ver se você está próximo de atingir os limites de simultaneidade, visualize essas métricas com a estatística Max.

Métricas de simultaneidade

- **ConcurrentExecutions:** o número de instâncias da função que estão processando eventos. Se esse número atingir a [cota de execuções simultâneas \(p. 53\)](#) para a região ou o [limite de simultaneidade reservada \(p. 113\)](#) que você configurou na função, as solicitações de invocação adicionais serão limitadas.
- **ProvisionedConcurrentExecutions:** o número de instâncias da função que estão processando eventos na [simultaneidade provisionada \(p. 113\)](#). Para cada invocação de um alias ou versão com simultaneidade provisionada, o Lambda emite a contagem atual.
- **ProvisionedConcurrencyUtilization:** para uma versão ou um alias, o valor de ProvisionedConcurrentExecutions dividido pela quantidade total de simultaneidade provisionada alocada. Por exemplo, .5 indica que 50% da simultaneidade provisionada alocada estão em uso.
- **UnreservedConcurrentExecutions:** para uma região da AWS, o número de eventos que estão sendo processados por funções que não têm simultaneidade reservada.

Acessar o Amazon CloudWatch Logs para o AWS Lambda

O AWS Lambda monitora automaticamente as funções do Lambda em seu nome e gera relatórios sobre métricas por meio do Amazon CloudWatch. Para ajudar a solucionar falhas em uma função, o Lambda depois da configuração de permissões, o Lambda registra em logs todas as solicitações tratadas por sua função e armazena automaticamente logs gerados pelo código pelo Amazon CloudWatch Logs.

Você pode inserir instruções de registro em log no seu código para ajudá-lo a validar se o seu código está funcionando conforme o esperado. O Lambda se integra automaticamente com o CloudWatch Logs e envia todos os logs do seu código a um grupo do CloudWatch Logs associado a uma função do Lambda, de nome `/aws/lambda/<nome da função>`.

Você pode exibir registros de funções do Lambda usando o console do Lambda, o console do CloudWatch, o AWS Command Line Interface(AWS CLI) ou a API do CloudWatch. Esta página descreve como exibir registros usando o console do Lambda.

Note

Pode levar de 5 a 10 minutos para que os logs apareçam após uma invocação de função.

Seção

- [Prerequisites \(p. 720\)](#)
- [Pricing \(p. 720\)](#)
- [Usar o console do Lambda \(p. 720\)](#)
- [Usar a AWS CLI \(p. 721\)](#)
- [Próximas etapas \(p. 721\)](#)

Prerequisites

Sua [função de execução \(p. 57\)](#) precisa de permissão para fazer o upload de registros no CloudWatch Logs. Você pode adicionar permissões do CloudWatch Logs usando uma política gerenciada da `AWSLambdaBasicExecutionRole` AWS fornecida pelo Lambda. Para adicionar essa política à sua função, execute o seguinte comando:

```
aws iam attach-role-policy --role-name your-role --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

Para obter mais informações, consulte [Políticas gerenciadas da AWS para recursos do Lambda \(p. 57\)](#).

Pricing

Não há nenhuma cobrança adicional para usar os logs do Lambda, porém, a cobrança padrão do CloudWatch Logs é aplicável. Para obter mais informações, consulte [Definição de preço do CloudWatch](#).

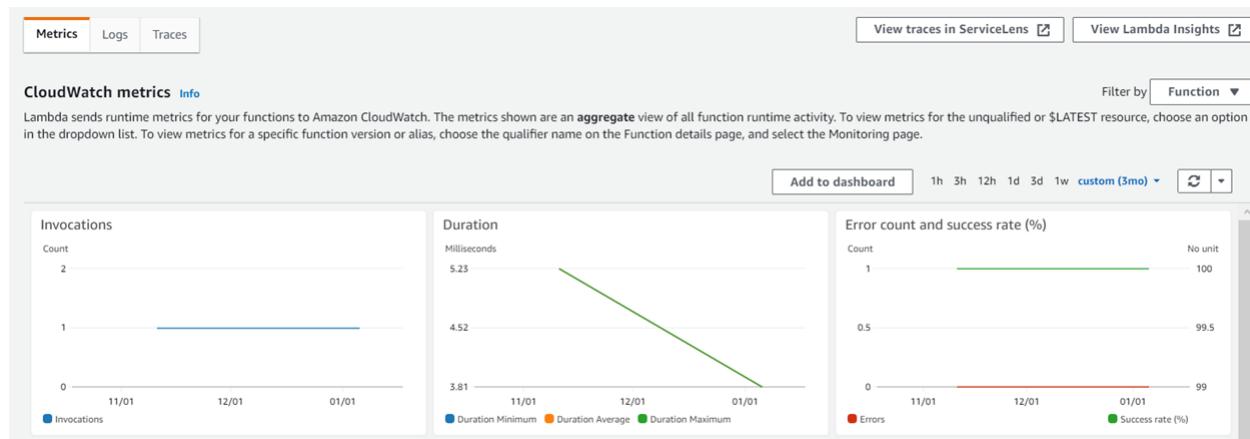
Usar o console do Lambda

A seção a seguir descreve como exibir registros de sua função no console do Lambda.

Para visualizar os logs usando o console do Lambda

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.

3. Escolha Monitor.



É exibida uma representação gráfica das métricas da função Lambda.

4. Escolha View logs in CloudWatch (Exibir logs no CloudWatch).

Usar a AWS CLI

Para depurar e validar que seu código está funcionando conforme o esperado, é possível gerar logs com a funcionalidade de registro em log padrão para sua linguagem de programação. O tempo de execução do Lambda faz upload da saída de log da sua função para o CloudWatch Logs. Para obter instruções específicas de linguagem, consulte os tópicos a seguir:

- [AWS Lambda Registro em log da função do em Node.js \(p. 524\)](#)
- [AWS Lambda Registro em log da função do em Python \(p. 553\)](#)
- [AWS Lambda Registro em log da função do em Ruby \(p. 578\)](#)
- [AWS Lambda Registro em log da função do em Java \(p. 611\)](#)
- [AWS Lambda Registro em log da função do em Go \(p. 647\)](#)
- [AWS Lambda Registro em log da função do em C# \(p. 677\)](#)
- [AWS Lambda Registro em log da função do no PowerShell \(p. 698\)](#)

Próximas etapas

- Saiba mais sobre os grupos de logs e como acessá-los no console do CloudWatch em [Monitoring system, application, and custom log files](#) no Manual do usuário do Amazon CloudWatch.

Usando o CodeGuru Profiler com sua função do Lambda

Você pode usar o Amazon CodeGuru Profiler para obter insights sobre a performance de tempo de execução de suas funções do Lambda. Esta página descreve como ativar o CodeGuru Profiler pelo console do Lambda.

Seções

- [Tempos de execução compatíveis \(p. 722\)](#)

- Ativando o CodeGuru Profiler do console do Lambda (p. 722)
- O que acontece quando você ativa o CodeGuru Profiler no console do Lambda? (p. 722)
- Próximas etapas (p. 723)

Tempos de execução compatíveis

Você pode ativar o CodeGuru Profiler no console do Lambda se o tempo de execução da sua função for Python 3.8, Java 8 com Amazon Linux 2 ou Java 11. Para versões adicionais de tempo de execução, você pode ativar o CodeGuru Profiler manualmente.

- Para tempos de execução Java, consulte [Criando o perfil de seus aplicativos Java que são executados em AWS Lambda](#).
- Para tempos de execução do Python, consulte [Perfilando seus aplicativos Python que são executados em AWS Lambda](#).

Ativando o CodeGuru Profiler do console do Lambda

Esta seção descreve como ativar o CodeGuru Profiler no no console do Lambda.

Para ativar o CodeGuru Profiler a partir do console do Lambda

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha a função.
3. Escolha a guia Configuration (Configuração).
4. No painel Monitoring and operations tools (Ferramentas de monitoramento e operações), selecione Edit (Editar).
5. UnderAmazon CodeGuru Profiler, ative o Código perfilado.
6. Escolha Save (Salvar).

Após a ativação, o CodeGuru cria automaticamente um grupo de profiler com o nome `aws-lambda-<your-function-name>`. Você pode alterar o nome do console CodeGuru.

O que acontece quando você ativa o CodeGuru Profiler no console do Lambda?

Quando você ativa o CodeGuru Profiler do console, o Lambda faz automaticamente o seguinte em seu nome:

- Lambda adiciona uma camada CodeGuru Profiler à função. Para obter mais detalhes, consulte [Usar o AWS LambdaCamadas do no Amazon CodeGuru Profiler Guia do usuário](#).
- O Lambda também adiciona variáveis de ambiente à sua função. O valor exato varia de acordo com o tempo de execução.

Variáveis de ambiente

Tempos de execução	Chave	Valor
java8.al2, java11	JAVA_TOOL_OPTIONS	<code>-javaagent:/opt/codeguru-profiler-java-agent-standalone.jar</code>

Tempos de execução	Chave	Valor
python3.8	AWS_LAMBDA_EXEC_WRAPPER	/opt/ codeguru_profiler_lambda_exec

- O Lambda adiciona o `AmazonCodeGuruProfilerAgentAccess` à função de execução da sua função.

Note

Quando você desativa o CodeGuru Profiler do console, o Lambda remove automaticamente a camada do CodeGuru Profiler e as variáveis de ambiente da sua função. No entanto, o Lambda não remove o `AmazonCodeGuruProfilerAgentAccess` A partir da função de execução.

Próximas etapas

- Saiba mais sobre os dados coletados pelo seu grupo de profiler em [Trabalhar com visuais](#) no Amazon CodeGuru Profiler Guia do usuário.

Exemplo de fluxos de trabalho usando outros serviços do AWS

O AWS Lambda pode ser integrado a outros serviços da AWS para ajudá-lo a monitorar, rastrear, depurar e solucionar problemas de suas funções do Lambda. Esta página mostra os fluxos de trabalho que você pode usar com o AWS X-Ray, o AWS Trusted Advisor e o CloudWatch ServiceLens para rastrear e solucionar problemas de suas funções do Lambda.

Seções

- [Prerequisites](#) (p. 723)
- [Pricing](#) (p. 724)
- [Exemplo de fluxo de trabalho do AWS X-Ray para visualização de um mapa de serviço](#) (p. 724)
- [Exemplo de fluxo de trabalho do AWS X-Ray para a exibição de detalhes de rastreamento](#) (p. 725)
- [Exemplo de fluxo de trabalho do AWS Trusted Advisor para exibir recomendações](#) (p. 725)
- [Próximas etapas](#) (p. 726)

Prerequisites

A seção a seguir descreve as etapas para usar o AWS X-Ray e o Trusted Advisor para solucionar problemas em suas funções do Lambda.

Usar o AWS X-Ray

O AWS X-Ray precisa ser ativado no console do Lambda para a conclusão dos fluxos de trabalho do AWS X-Ray nessa página. Se sua função de execução não tiver as permissões necessárias, o console do Lambda tentará adicioná-las à função.

Para ativar o AWS X-Ray no console do Lambda

1. Abra a [página Functions \(Funções\)](#) no console do Lambda.
2. Escolha a função.
3. Escolha a guia Configuration (Configuração).

4. No painel Monitoring tools (Ferramentas de monitoramento), selecione Edit (Editar).
5. Em AWS X-Ray, ative o Rastreamento ativo.
6. Escolha Save (Salvar).

Usar o AWS Trusted Advisor

O AWS Trusted Advisor inspeciona o seu ambiente da AWS e faz recomendações sobre como economizar dinheiro, melhorar a performance e a disponibilidade do sistema ou ajuda a corrigir falhas de segurança. Você pode usar as verificações do Trusted Advisor para avaliar as funções e aplicações do Lambda em sua conta da AWS. As verificações fornecem etapas recomendadas a tomar e recursos para obter mais informações.

- Para obter mais informações sobre os planos de suporte da AWS para verificações do Trusted Advisor, consulte [Planos de suporte](#).
- Para obter mais informações sobre as verificações para o Lambda, consulte a [Lista de verificação de práticas recomendadas do AWS Trusted Advisor](#).
- Para obter mais informações sobre como usar o console do Trusted Advisor, consulte [Comece a usar o AWS Trusted Advisor](#).
- Para obter instruções sobre como permitir e negar acesso ao console do Trusted Advisor, consulte os [Exemplos de políticas do IAM](#).

Pricing

- Com o AWS X-Ray, você só paga pelo que usa, de acordo com o número de rastreamentos gravados, recuperados e verificados. Para obter mais informações, consulte [Definição de preço do AWS X-Ray](#).
- As verificações de otimização de custos do Trusted Advisor estão incluídas nas assinaturas de suporte Business e Enterprise da AWS. Para obter mais informações, consulte [Definição de preço do AWS Trusted Advisor](#).

Exemplo de fluxo de trabalho do AWS X-Ray para visualização de um mapa de serviço

Se você tiver ativado o AWS X-Ray, poderá visualizar um mapa de serviço do ServiceLens no console do CloudWatch. Um mapa de serviço exibe os recursos e os endpoints de serviço como “nós” e destaca o tráfego, a latência e os erros de cada nó e de suas conexões.

É possível escolher um nó para ver insights detalhados sobre as métricas, os logs e os rastreamentos correlacionados associados a essa parte do serviço. Isso permite investigar os problemas e seus efeitos no aplicativo.

Para visualizar o mapa de serviço e os rastreamentos usando o console do CloudWatch

1. Abra a página [Functions \(Funções\)](#) no console do Lambda.
2. Escolha uma função.
3. Escolha Monitoring.
4. Selecione Exibir traços no X-Ray.
5. Selecione Mapa de serviço.
6. Escolha entre os intervalos de tempo predefinidos ou escolha um intervalo de tempo personalizado.
7. Para solucionar problemas de solicitações, selecione um filtro.

Exemplo de fluxo de trabalho do AWS X-Ray para a exibição de detalhes de rastreamento

Se você tiver ativado o AWS X-Ray, poderá usar a exibição de função única no painel do CloudWatch Lambda Insights para mostrar os dados de rastreamento distribuídos de um erro de invocação de função. Por exemplo, se a mensagem de log do aplicativo mostrar um erro, você poderá abrir a exibição de rastreamentos do ServiceLens para ver os dados de rastreamento distribuídos e os outros serviços que processam a transação.

Para exibir detalhes de rastreamento de uma função

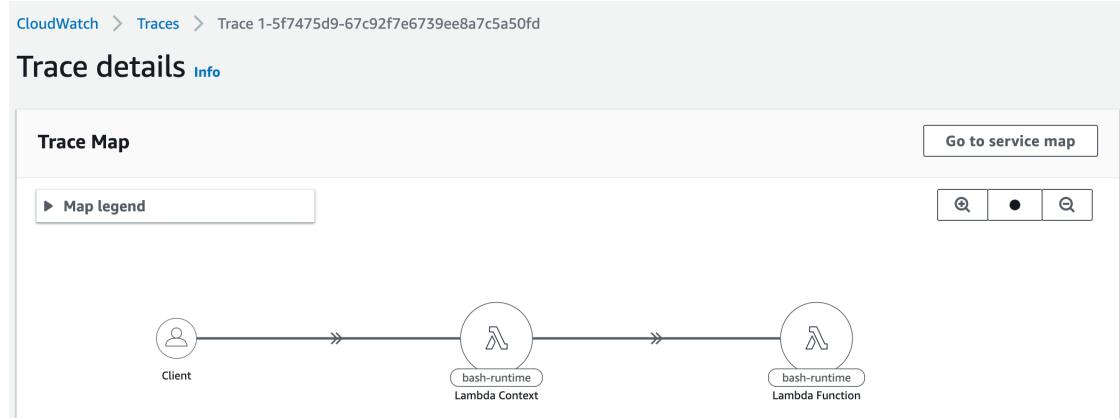
1. Abra a [visualização de função única](#) no console do CloudWatch.
2. Selecione a aba Logs de aplicativos.
3. Use o Timestamp ou a Mensagem para identificar a solicitação de invocação que deseja solucionar.
4. Para mostrar as Most recent 1000 invocations (1000 invocações mais recentes), selecione a guia Invocations (Invocações).

The screenshot shows the 'Invocations' tab selected in the CloudWatch Lambda Insights interface. The main area displays a table titled 'Most recent 1000 invocations (359)'. The table has columns for 'Timestamp', 'Request ID', 'Trace' (with a 'View' link), 'Memory %', and 'Network IO'. Three rows are visible, each showing a timestamp, request ID, trace link, memory usage (91% or 92%), and network IO (14 kB or 11 kB). The table includes sorting and filtering options at the top.

Timestamp	Request ID	Trace	Memory %	Network IO
2020-09-30 12:12:05 (UTC-06:00)	00c99bab-92f7-46cc-af28-ca71ad43f894	View	91%	14 kB
2020-09-30 14:35:05 (UTC-06:00)	01fd5427-f3cd-4689-a39e-19f59c3eb7a2	View	91%	11 kB
2020-09-30 14:45:05 (UTC-06:00)	02be2a9a-88ef-4b08-ba94-02a1a0c7893d	View	92%	14 kB

5. Selecione a coluna ID da solicitação para classificar as entradas em ordem alfabética crescente.
6. Na coluna Rastreamento, selecione Exibir.

A página Detalhes do rastreamento será aberta na exibição de rastreamentos do ServiceLens.



Exemplo de fluxo de trabalho do AWS Trusted Advisor para exibir recomendações

O Trusted Advisor verifica as funções do Lambda em todas as regiões da AWS para identificar as funções com o maior potencial de economia de custos, além de fornecer recomendações acionáveis para

otimização. Ele analisa os seus dados de uso do Lambda, como tempo de execução da função, duração faturada, memória utilizada, memória configurada, configuração de tempo limite e erros.

Por exemplo, a verificação Funções do Lambda com alta taxa de erro recomenda que você use o AWS X-Ray ou o CloudWatch para detectar erros em funções do Lambda.

Para verificar se há funções com altas taxas de erro

1. Abra o [console do Trusted Advisor](#).
2. Escolha a categoria Cost Optimization (Otimização de custos).
3. Role para baixo até Funções do AWS Lambda com altas taxas de erro. Expanda a seção para ver os resultados e as ações recomendadas.

Próximas etapas

- Saiba mais sobre como integrar rastreamentos, métricas, logs e alarmes em [Usando o ServiceLens para monitorar a integridade de seus aplicativos](#).
- Saiba mais sobre como obter uma lista de verificações do Trusted Advisor em [Como usar o Trusted Advisor como um serviço Web](#).

Segurança em AWS Lambda

A segurança da nuvem na AWS é a nossa maior prioridade. Como cliente da AWS, você se beneficiará de um datacenter e de uma arquitetura de rede criados para atender aos requisitos das empresas com as maiores exigências de segurança.

A segurança é uma responsabilidade compartilhada entre a AWS e você. O [modelo de responsabilidade compartilhada](#) descreve isto como segurança da nuvem e segurança na nuvem:

- Segurança da nuvem: a AWS é responsável pela proteção da infraestrutura que executa produtos da AWS na Nuvem AWS. A AWS também fornece serviços que podem ser usados com segurança. Auditores de terceiros testam e verificam regularmente a eficácia da nossa segurança como parte dos [programas de conformidade da AWS](#). Para saber mais sobre os programas de conformidade que se aplicam ao AWS Lambda, consulte [Serviços da AWS no escopo pelo programa de conformidade](#).
- Segurança da nuvem: sua responsabilidade é determinada pelo serviço da AWS que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade de seus dados, os requisitos da sua empresa e as leis e regulamentos aplicáveis.

Esta documentação ajuda a entender como aplicar o modelo de responsabilidade compartilhada ao usar o Lambda. Os tópicos a seguir mostram como configurar o Lambda para atender aos seus objetivos de segurança e conformidade. Saiba também como usar outros serviços da AWS que ajudam a monitorar e proteger os recursos do Lambda.

Para obter mais informações sobre como colocar princípios de segurança em prática em aplicações do Lambda, consulte [Segurança](#) no Guia do operador do Lambda.

Tópicos

- [Proteção de dados no AWS Lambda \(p. 727\)](#)
- [Gerenciamento de identidade e acesso para o Lambda \(p. 729\)](#)
- [Validação de conformidade do AWS Lambda \(p. 738\)](#)
- [Resiliência no AWS Lambda \(p. 738\)](#)
- [Segurança da infraestrutura no AWS Lambda \(p. 739\)](#)
- [Análise de vulnerabilidade e configuração no AWS Lambda \(p. 740\)](#)

Proteção de dados no AWS Lambda

O [modelo de responsabilidade compartilhada](#) da AWS se aplica à proteção de dados no AWS Lambda. Conforme descrito nesse modelo, a AWS é responsável por proteger a infraestrutura global que executa toda a Nuvem AWS. Você é responsável por manter o controle sobre seu conteúdo hospedado nessa infraestrutura. Esse conteúdo inclui as tarefas de configuração e gerenciamento de segurança dos serviços da AWS que você usa. Para obter mais informações sobre a privacidade de dados, consulte as [Perguntas frequentes sobre privacidade de dados](#). Para obter mais informações sobre a proteção de dados na Europa, consulte a postagem do blog [AWS Shared Responsibility Model and GDPR](#) no Blog de segurança da AWS.

Para fins de proteção de dados, recomendamos que você proteja as credenciais da conta da Conta da AWS e configure as contas de usuário individuais com o AWS Identity and Access Management (IAM).

Dessa maneira, cada usuário receberá apenas as permissões necessárias para cumprir suas obrigações de trabalho. Recomendamos também que você proteja seus dados das seguintes formas:

- Use uma autenticação multifator (MFA) com cada conta.
- Use SSL/TLS para se comunicar com os recursos da AWS. Recomendamos TLS 1.2 ou posterior.
- Configure o registro em log das atividades da API e do usuário com o AWS CloudTrail.
- Use as soluções de criptografia da AWS, juntamente com todos os controles de segurança padrão nos serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados pessoais armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-2 ao acessar a AWS por meio de uma interface de linha de comando ou uma API, use um endpoint do FIPS. Para obter mais informações sobre endpoints do FIPS, consulte [Federal Information Processing Standard \(FIPS\) 140-2](#).

É altamente recomendável que você nunca coloque informações de identificação confidenciais, como endereços de e-mail dos seus clientes, em marcações ou campos de formato livre, como um campo Name (Nome). Isso inclui usar o console, a API, a AWS CLI ou os AWS SDKs ao trabalhar com o Lambda ou outros serviços da AWS. Quaisquer dados inseridos em marcações ou campos de formato livre usados para nomes podem ser usados para logs de cobrança ou diagnóstico. Se você fornecer um URL para um servidor externo, recomendamos fortemente que não sejam incluídas informações de credenciais no URL para validar a solicitação a esse servidor.

Seções

- [Criptografia em trânsito \(p. 728\)](#)
- [Criptografia em repouso \(p. 728\)](#)

Criptografia em trânsito

Os endpoints da API do Lambda oferecem suporte a conexões seguras somente em HTTPS. Ao gerenciar recursos do Lambda com o AWS Management Console, o AWS SDK ou a API do Lambda, toda a comunicação é criptografada com Transport Layer Security (TLS).

Quando você [conecta a função a um sistema de arquivos \(p. 139\)](#), o Lambda usa a [Criptografia em trânsito](#) para todas as conexões.

Para obter uma lista completa de todos os endpoints da API, consulte [Regiões e endpoints da AWS](#) na Referência geral da AWS.

Criptografia em repouso

Você pode usar variáveis de ambiente para armazenar segredos de forma segura para uso com funções do Lambda. O Lambda sempre criptografa variáveis de ambiente em repouso.

Além disso, você pode usar os seguintes recursos para personalizar o modo como as variáveis de ambiente serão criptografadas.

- **Configuração de chave:** de acordo com cada função, você pode configurar o Lambda para usar a chave de criptografia que você criar e gerenciar no AWS Key Management Service. Essas políticas são referenciadas como CMKs gerenciadas pelo cliente. Para funções que não têm uma CMK gerenciada pelo cliente, o Lambda usa uma CMK gerenciada pela AWS chamada `aws/lambda`, que o Lambda cria na sua conta.
- **Auxiliares de criptografia:** o console do Lambda permite criptografar os valores das variáveis do ambiente do lado do cliente, antes de enviá-los para o Lambda. Isso aumenta a segurança, impedindo

que segredos sejam exibidos sem criptografia no console do Lambda ou na configuração da função que é retornada pela API do Lambda. O console também oferece um código de exemplo que você pode adaptar para descriptografar os valores no seu manipulador de funções.

Para obter mais informações, consulte [Usar variáveis de ambiente do AWS Lambda \(p. 99\)](#).

O Lambda sempre criptografa os arquivos dos quais você faz upload no Lambda, incluindo [pacotes de implantação \(p. 12\)](#) e [arquivos de camada \(p. 85\)](#).

O Amazon CloudWatch Logs e o AWS X-Ray também criptografam dados por padrão, e podem ser configurados para usarem uma chave gerenciada pelo cliente. Para obter mais detalhes, consulte [Criptografar dados de log no CloudWatch Logs](#) e [Proteção de dados no AWS X-Ray](#).

Gerenciamento de identidade e acesso para o Lambda

O AWS Identity and Access Management (IAM) é um serviço da AWS que ajuda a controlar o acesso aos recursos da AWS de forma segura. Os administradores do IAM controlam quem pode ser autenticado (conectado) e autorizado (ter permissões) para usar os recursos do Lambda. O IAM é um serviço da AWS que pode ser usado sem custo adicional.

Tópicos

- [Audience \(p. 729\)](#)
- [Autenticar com identidades \(p. 730\)](#)
- [Gerenciamento do acesso usando políticas \(p. 732\)](#)
- [Como o AWS Lambda funciona com o IAM \(p. 733\)](#)
- [AWS Lambda Exemplos de políticas baseadas em identidade do \(p. 733\)](#)
- [Solução de problemas de identidade e acesso do AWS Lambda \(p. 735\)](#)

Audience

O uso do AWS Identity and Access Management (IAM) varia dependendo do trabalho que é realizado no Lambda.

Usuário do serviço: se você usar o serviço Lambda para fazer o trabalho, o administrador fornecerá as credenciais e as permissões necessárias. À medida que usar mais recursos do Lambda para fazer seu trabalho, você poderá precisar de permissões adicionais. Entender como o acesso é gerenciado pode ajudar você a solicitar as permissões corretas ao seu administrador. Se não for possível acessar um recurso no Lambda, consulte [Solução de problemas de identidade e acesso do AWS Lambda \(p. 735\)](#).

Administrador do serviço: se você for o responsável pelos recursos do Lambda na empresa, provavelmente terá acesso total ao Lambda. Seu trabalho é determinar quais recursos do Lambda seus funcionários devem acessar. Assim, é necessário enviar solicitações ao administrador do IAM para alterar as permissões dos usuários de seu serviço. Revise as informações nesta página para entender os conceitos básicos do IAM. Para saber mais sobre como a empresa pode usar o IAM com o Lambda, consulte [Como o AWS Lambda funciona com o IAM \(p. 733\)](#).

Administrador do IAM: se você é um administrador do IAM, talvez queira saber detalhes sobre como pode escrever políticas para gerenciar o acesso ao Lambda. Para visualizar exemplos de políticas baseadas

em identidade do Lambda que podem ser usadas no IAM, consulte [AWS Lambda Exemplos de políticas baseadas em identidade do IAM](#) (p. 733).

Autenticar com identidades

A autenticação é a forma como você faz login na AWS usando suas credenciais de identidade. Para obter mais informações sobre como fazer login usando o AWS Management Console, consulte [Login no AWS Management Console como usuário do IAM ou usuário root](#) no Manual do usuário do IAM.

É necessário estar autenticado (conectado à AWS) como o usuário root da Conta da AWS ou um usuário do IAM, ou ainda assumindo uma função do IAM. Também é possível usar a autenticação de logon único da sua empresa ou até mesmo fazer login usando o Google ou o Facebook. Nesses casos, o administrador configurou anteriormente federação de identidades usando funções do IAM. Ao acessar a AWS usando credenciais de outra empresa, você estará assumindo uma função indiretamente.

Para fazer login diretamente no [AWS Management Console](#), use sua senha com o e-mail do usuário root ou seu nome de usuário do IAM. É possível acessar a AWS de maneira programática usando chaves de acesso do AWS seu usuário root ou dos usuários do IAM. Se você não utilizar as ferramentas da AWS, você deverá assinar a solicitação por conta própria. Faça isso usando o Signature Version 4, um protocolo para autenticação de solicitações de API de entrada. Para obter mais informações sobre solicitações de autenticação, consulte [Processo de assinatura do Signature Version 4](#) na Referência geral da AWS.

Independentemente do método de autenticação usado, também pode ser exigido que você forneça informações adicionais de segurança. Por exemplo, a AWS recomenda o uso da autenticação multifator (MFA) para aumentar a segurança de sua conta. Para saber mais, consulte [Uso da autenticação multifator \(MFA\) na AWS](#) no Manual do usuário do IAM.

Usuário root da Conta da AWS

Ao criar uma Conta da AWS, você começa com uma única identidade de login que tenha acesso total a todos os recursos e serviços da AWS na conta. Essa identidade é denominada usuário root da Conta da AWS e é acessada pelo login com o endereço de e-mail e a senha que você usou para criar a conta. Recomendamos que não use o usuário raiz para suas tarefas do dia a dia, nem mesmo as administrativas. Em vez disso, siga as [práticas recomendadas para o uso do usuário raiz somente a fim de criar seu primeiro usuário do IAM](#). Depois, armazene as credenciais do usuário raiz com segurança e use-as para executar somente algumas tarefas de gerenciamento de contas e de serviços.

Grupos e usuários do IAM

Um [usuário do IAM](#) é uma identidade dentro da Conta da AWS que tem permissões específicas para uma única pessoa ou aplicação. Um usuário do IAM pode ter credenciais de longo prazo, como um nome de usuário e uma senha ou um conjunto de chaves de acesso. Para saber como gerar chaves de acesso, consulte [Gerenciar chaves de acesso para usuários do IAM](#) no Guia do usuário do IAM. Ao gerar chaves de acesso para um usuário do IAM, visualize e salve o par de chaves de maneira segura. Não será possível recuperar a chave de acesso secreta futuramente. Em vez disso, você deverá gerar outro par de chaves de acesso.

Um [grupo do IAM](#) é uma identidade que especifica uma coleção de usuários do IAM. Não é possível fazer login como um grupo. É possível usar grupos para especificar permissões para vários usuários de uma vez. Os grupos facilitam o gerenciamento de permissões para grandes conjuntos de usuários. Por exemplo, você pode ter um grupo chamado IAMAdmins e atribuir a esse grupo permissões para administrar recursos do IAM.

Usuários são diferentes de funções. Um usuário é exclusivamente associado a uma pessoa ou a um aplicativo, mas uma função pode ser assumida por qualquer pessoa que precisar dela. Os usuários têm credenciais permanentes de longo prazo, mas as funções fornecem credenciais temporárias. Para saber mais, consulte [Quando criar um usuário do IAM \(em vez de uma função\)](#) no Manual do usuário do IAM.

Funções do IAM

Uma [função do IAM](#) é uma identidade dentro da Conta da AWS que tem permissões específicas. Ela é semelhante a um usuário do IAM, mas não está associada a uma pessoa específica. É possível assumir temporariamente uma função do IAM no AWS Management Console [alternando funções](#). É possível assumir uma função chamando uma operação de API da AWS CLI ou da AWS, ou usando um URL personalizado. Para mais informações sobre métodos para o uso de funções, consulte [Usar funções do IAM](#) no Guia do usuário do IAM.

As funções do IAM com credenciais temporárias são úteis nas seguintes situações:

- Permissões temporárias para usuários do IAM: um usuário do IAM pode assumir uma função do IAM para obter temporariamente permissões diferentes para uma tarefa específica.
- Acesso de usuário federado: em vez de criar um usuário do IAM, você poderá usar identidades de usuários existentes no AWS Directory Service, em seu diretório de usuários corporativos ou em um provedor de identidades da Web. Estes são conhecidos como usuários federados. A AWS atribui uma função a um usuário federado quando o acesso é solicitado por meio de um [provedor de identidades](#). Para obter mais informações sobre usuários federados, consulte [Usuários federados e funções](#) no Guia do usuário do IAM.
- Acesso entre contas: é possível usar uma função do IAM para permitir que alguém (um principal confiável) em outra conta acesse recursos em sua conta. As funções são a principal forma de conceder acesso entre contas. No entanto, alguns serviços da AWS permitem que você anexe uma política diretamente a um recurso (em vez de usar uma função como proxy). Para saber a diferença entre funções e políticas baseadas em recurso para acesso entre contas, consulte [Como as funções do IAM diferem das políticas baseadas em recurso](#) no Manual do usuário do IAM.
- Acesso entre serviços: alguns serviços da AWS usam recursos em outros serviços da AWS. Por exemplo, quando você faz uma chamada em um serviço, é comum que esse serviço execute aplicações no Amazon EC2 ou armazene objetos no Amazon S3. Um serviço pode fazer isso usando as permissões do principal de chamada, usando uma função de serviço ou uma função vinculada ao serviço.
 - Permissões de principal: ao usar um usuário ou uma função do IAM para executar ações na AWS, você é considerado um principal. As políticas concedem permissões a uma entidade principal. Quando você usa alguns serviços, pode executar uma ação que, em seguida, aciona outra ação em outro serviço. Nesse caso, você deve ter permissões para executar ambas as ações. Para ver se uma ação exige ações dependentes adicionais em uma política, consulte [Ações, recursos e chaves de condição para o AWS Lambda](#) na Referência de autorização do serviço.
 - Função de serviço: uma função de serviço é uma [função do IAM](#) que um serviço assume para realizar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir uma função de serviço do IAM. Para obter mais informações, consulte [Criação de uma função para delegar permissões a um serviço da AWS](#) no Manual do usuário do IAM.
 - Função vinculada a serviço: uma função vinculada a serviço é um tipo de função de serviço vinculada a um serviço da AWS. O serviço pode assumir a função de executar uma ação em seu nome. As funções vinculadas ao serviço aparecem em sua conta do IAM e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não pode editar as permissões para funções vinculadas ao serviço.
- Aplicações em execução no Amazon EC2: é possível usar uma função do IAM para gerenciar credenciais temporárias para aplicações em execução em uma instância do EC2 e fazer solicitações da AWS CLI ou da AWS API. É preferível fazer isso do que armazenar chaves de acesso na instância do EC2. Para atribuir uma função da AWS a uma instância do EC2 e disponibilizá-la para todos os seus aplicativos, crie um perfil de instância que esteja anexado à instância. Um perfil de instância contém a função e permite que programas que estão em execução na instância do EC2 obtenham credenciais temporárias. Para mais informações, consulte [Usar uma função do IAM para conceder permissões a aplicações em execução nas instâncias do Amazon EC2](#) no Manual do usuário do IAM.

Para saber se deseja usar as funções do IAM, consulte [Quando criar uma função do IAM \(em vez de um usuário\)](#) no Guia do usuário do IAM.

Gerenciamento do acesso usando políticas

Você controla o acesso na AWS criando e anexando políticas às identidades do IAM ou aos recursos da AWS. Uma política é um objeto na AWS que, quando associado a uma identidade ou recurso, define suas permissões. Você pode fazer login como o usuário raiz ou um usuário do IAM ou assumir uma função do IAM. Quando você faz uma solicitação, a AWS avalia as políticas relacionadas baseadas em identidade ou baseadas em recursos. As permissões nas políticas determinam se a solicitação será permitida ou negada. A maioria das políticas são armazenadas na AWS como documentos JSON. Para obter mais informações sobre a estrutura e o conteúdo de documentos de políticas JSON, consulte [Visão geral das políticas JSON](#) no Manual do usuário do IAM.

Os administradores podem usar as políticas JSON da AWS para especificar quem tem acesso a quê. Ou seja, qual principal pode executar ações em quais recursos, e em que condições.

Cada entidade do IAM (usuário ou função) começa sem permissões. Em outras palavras, por padrão, os usuários não podem fazer nada, nem mesmo alterar sua própria senha. Para dar permissão a um usuário para fazer algo, um administrador deve anexar uma política de permissões ao usuário. Ou o administrador pode adicionar o usuário a um grupo que tenha as permissões pretendidas. Quando um administrador concede permissões a um grupo, todos os usuários desse grupo recebem essas permissões.

As políticas do IAM definem permissões para uma ação, independentemente do método usado para executar a operação. Por exemplo, suponha que você tenha uma política que permite a ação `iam:GetRole`. Um usuário com essa política pode obter informações de funções do AWS Management Console, da AWS CLI ou da API da AWS.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissões JSON que você pode anexar a uma identidade, como usuário, grupo de usuários ou função do IAM. Essas políticas controlam quais ações os usuários e funções podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte [Criar políticas do IAM](#) no Manual do usuário do IAM.

As políticas baseadas em identidade podem ser categorizadas ainda mais como políticas em linha ou políticas gerenciadas. As políticas em linha são anexadas diretamente a um único usuário, grupo ou função. As políticas gerenciadas são políticas independentes que podem ser anexadas a vários usuários, grupos e funções na Conta da AWS. As políticas gerenciadas incluem políticas gerenciadas pela AWS e políticas gerenciadas pelo cliente. Para saber como escolher entre uma política gerenciada ou uma política em linha, consulte [Escolher entre políticas gerenciadas e políticas em linha](#) no Manual do usuário do IAM.

Políticas baseadas em recursos

Políticas baseadas em recurso são documentos de políticas JSON que você anexa a um recurso. São exemplos de políticas baseadas em recursos as políticas de confiança de função do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o recurso ao qual a política está anexada, a política define quais ações um principal especificado pode executar nesse recurso e em que condições. Você deve [especificar um principal](#) em uma política baseada em recursos. Os principais podem incluir contas, usuários, funções, usuários federados ou serviços da AWS.

Políticas baseadas em recursos são políticas em linha que estão localizadas nesse serviço. Não é possível usar as políticas gerenciadas da AWS do IAM em uma política baseada em recursos.

Listas de controle de acesso (ACLs)

As listas de controle de acesso (ACLs) controlam quais principais (membros, usuários ou funções da conta) têm permissões para acessar um recurso. As ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

Amazon S3, AWS WAF e Amazon VPC são exemplos de serviços que oferecem suporte a ACLs. Para saber mais sobre ACLs, consulte [Visão geral da lista de controle de acesso \(ACL\)](#) no Guia do desenvolvedor do Amazon Simple Storage Service.

Outros tipos de política

AWSA oferece suporte a tipos de política menos comuns. Esses tipos de política podem definir o máximo de permissões concedidas a você pelos tipos de política mais comuns.

- **Limites de permissões:** um limite de permissões é um recurso avançado no qual você define o máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM (usuário ou função do IAM). É possível definir um limite de permissões para uma entidade. As permissões resultantes são a interseção das políticas baseadas em identidade da entidade e seus limites de permissões. As políticas baseadas em recurso que especificam o usuário ou a função no campo Principal não são limitadas pelo limite de permissões. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações sobre limites de permissões, consulte [Limites de permissões para identidades do IAM](#) no Manual do usuário do IAM.
- **Políticas de controle de serviço (SCPs):** SCPs são políticas JSON que especificam as permissões máximas para uma organização ou unidade organizacional (UO) no AWS Organizations. O AWS Organizations é um serviço para agrupar e gerenciar centralmente várias Contas da AWS pertencentes à sua empresa. Se você habilitar todos os recursos em uma organização, poderá aplicar políticas de controle de serviço (SCPs) a qualquer uma ou a todas as contas. Uma SCP limita as permissões para entidades em contas-membro, incluindo cada usuário root da Conta da AWS . Para obter mais informações sobre o Organizations e SCPs, consulte [Como os SCPs funcionam](#) no Manual do usuário do AWS Organizations.
- **Políticas de sessão:** são políticas avançadas que você transmite como um parâmetro quando cria de forma programática uma sessão temporária para uma função ou um usuário federado. As permissões da sessão resultante são a interseção das políticas baseadas em identidade do usuário ou da função e das políticas de sessão. As permissões também podem ser provenientes de uma política baseada em recurso. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações, consulte [Políticas de sessão](#) no Guia do usuário do IAM.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como a AWS determina se deve permitir uma solicitação quando há vários tipos de política envolvidos, consulte [Lógica da avaliação de políticas](#) no Guia do Usuário do IAM.

Como o AWS Lambda funciona com o IAM

Antes de usar o IAM para gerenciar o acesso ao Lambda, você deve entender quais recursos do IAM estão disponíveis para uso com o Lambda. Para obter uma visualização de alto nível de como o Lambda e outros serviços da AWS funcionam com o IAM, consulte [Serviços da AWS compatíveis com o IAM](#) no Manual do usuário do IAM.

Para obter uma visão geral de permissões, políticas e funções como são usadas pelo Lambda, consulte [AWS LambdaPermissões \(p. 56\)](#).

AWS LambdaExemplos de políticas baseadas em identidade do

Por padrão, os usuários e as funções do IAM não têm permissão para criar ou modificar recursos do Lambda. Eles também não podem executar tarefas usando o AWS Management Console, a AWS CLI

ou uma API da AWS. Um administrador do IAM deve criar políticas do IAM que concedam aos usuários e funções permissão para executarem operações de API específicas nos recursos especificados de que precisam. O administrador deve anexar essas políticas aos usuários ou grupos do IAM que exigem essas permissões.

Para saber como criar uma política baseada em identidade do IAM usando esses exemplos de documentos de política JSON, consulte [Criar políticas na guia JSON](#) no Guia do usuário do IAM.

Tópicos

- [Melhores práticas de políticas \(p. 734\)](#)
- [Usar o console do Lambda \(p. 734\)](#)
- [Permitir que os usuários visualizem suas próprias permissões \(p. 735\)](#)

Melhores práticas de políticas

As políticas baseadas em identidade são muito eficientes. Elas determinam se alguém pode criar, acessar ou excluir recursos do Lambda em sua conta. Essas ações podem incorrer em custos para a Conta da AWS . Ao criar ou editar políticas baseadas em identidade, siga estas diretrizes e recomendações:

- Começar a usar políticas gerenciadas pela AWS: para começar a usar o Lambda rapidamente, use as políticas gerenciadas pela AWS para conceder a seus funcionários as permissões necessárias. Essas políticas já estão disponíveis em sua conta e são mantidas e atualizadas pela AWS. Para obter mais informações, consulte [Começar a usar permissões com políticas gerenciadas da AWS](#) no Guia do usuário do IAM.
- Conceder privilégio mínimo: ao criar políticas personalizadas, conceda apenas as permissões necessárias para executar uma tarefa. Comece com um conjunto mínimo de permissões e conceda permissões adicionais conforme necessário. Fazer isso é mais seguro do que começar com permissões que são muito lenientes e tentar restringi-las posteriormente. Para obter mais informações, consulte [Conceder privilégio mínimo](#) no Guia do usuário do IAM.
- Habilitar MFA para operações confidenciais: para aumentar a segurança, exija que os usuários do IAM usem Multi-Factor Authentication (MFA) para acessar recursos ou operações de API confidenciais. Para obter mais informações, consulte [Usar autenticação multifator \(MFA\) na AWS](#) no Guia do Usuário do IAM.
- Usar condições de política para segurança adicional: na medida do possível, defina as condições sob as quais suas políticas baseadas em identidade permitem o acesso a um recurso. Por exemplo, você pode gravar condições para especificar um intervalo de endereços IP permitidos do qual a solicitação deve partir. Você também pode escrever condições para permitir somente solicitações em uma data especificada ou período ou para exigir o uso de SSL ou MFA. Para obter mais informações, consulte [Elementos de política JSON do IAM: condição](#) no Guia do usuário do IAM.

Usar o console do Lambda

Para acessar o console do AWS Lambda, você precisa ter um conjunto mínimo de permissões. Essas permissões dão autorização para que você liste e visualize detalhes sobre os recursos do Lambda na sua conta da AWS. Se você criar uma política baseada em identidade que seja mais restritiva que as permissões mínimas necessárias, o console não funcionará como pretendido para entidades (usuários ou funções do IAM) com essa política.

Para obter um exemplo de política que concede acesso mínimo ao desenvolvimento de funções, consulte [Desenvolvimento da função \(p. 67\)](#). Além das APIs do Lambda, o console do Lambda usa outros serviços para exibir a configuração do acionador e permite que você adicione novos acionadores. Se os usuários utilizam o Lambda com outros serviços, eles também precisam de acesso a esses serviços. Para obter mais detalhes sobre como configurar outros serviços do Lambda, consulte [Usar o AWS Lambda com outros serviços \(p. 277\)](#).

Permitir que os usuários visualizem suas próprias permissões

Este exemplo mostra como você pode criar uma política que permite que os usuários do IAM visualizem as políticas gerenciadas e em linha anexadas a sua identidade de usuário. Essa política inclui permissões para concluir essa ação no console ou de forma programática usando a AWS CLI ou a API da 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": "*"  
        }  
    ]  
}
```

Solução de problemas de identidade e acesso do AWS Lambda

Use as informações a seguir para ajudar a diagnosticar e corrigir problemas comuns que você possa encontrar ao trabalhar com o Lambda e o IAM.

Tópicos

- [Não tenho autorização para executar uma ação no Lambda \(p. 736\)](#)
- [Não estou autorizado a executar iam:PassRole \(p. 736\)](#)
- [Quero visualizar minhas chaves de acesso \(p. 736\)](#)
- [Sou administrador e desejo permitir que outras pessoas tenham acesso ao Lambda \(p. 737\)](#)
- [Sou administrador e quero migrar de políticas gerenciadas da AWS para o Lambda que ficarão defasadas \(p. 737\)](#)
- [Desejo permitir que pessoas fora da minha conta da AWS acessem meus recursos do Lambda \(p. 737\)](#)

Não tenho autorização para executar uma ação no Lambda

Se o AWS Management Console informar que você não está autorizado a executar uma ação, você deverá entrar em contato com o administrador para obter assistência. O administrador é a pessoa que forneceu a você o seu nome de usuário e senha.

O erro de exemplo a seguir ocorre quando o usuário `mateojackson` do IAM tenta usar o console para visualizar detalhes sobre uma função, mas não tem as permissões `lambda:GetFunction`.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:  
lambda:GetFunction on resource: my-function
```

Neste caso, Mateo pede ao administrador para atualizar suas políticas para permitir a ele o acesso ao recurso `my-function` usando a ação `lambda:GetFunction`.

Não estou autorizado a executar iam:PassRole

Se você receber uma mensagem de erro informando que você não está autorizado a executar a ação `iam:PassRole`, entre em contato com o administrador para obter assistência. O administrador é a pessoa que forneceu a você o seu nome de usuário e senha. Peça a essa pessoa para atualizar suas políticas para permitir que você passe uma função para o Lambda.

Alguns serviços da AWS permitem que você passe uma função existente para o serviço, em vez de criar uma nova função de serviço ou função vinculada ao serviço. Para fazer isso, um usuário deve ter permissões para passar a função para o serviço.

O erro de exemplo a seguir ocorre quando uma usuária do IAM chamada `marymajor` tenta usar o console para executar uma ação no Lambda. No entanto, a ação exige que o serviço tenha permissões concedidas por uma função de serviço. Mary não tem permissões para passar a função para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

Neste caso, Mary pede ao administrador para atualizar suas políticas para permitir que ela execute a ação `iam:PassRole`.

Quero visualizar minhas chaves de acesso

Depois de criar suas chaves de acesso do IAM, é possível visualizar seu ID da chave de acesso a qualquer momento. No entanto, você não pode visualizar sua chave de acesso secreta novamente. Se você perder sua chave secreta, crie um novo par de chaves de acesso.

As chaves de acesso consistem em duas partes: um ID de chave de acesso (por exemplo, `AKIAIOSFODNN7EXAMPLE`) e uma chave de acesso secreta (por exemplo, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Como um nome de usuário e uma senha, você deve usar o ID da chave de acesso e a chave de acesso secreta em conjunto para autenticar suas solicitações. Gerencie suas chaves de acesso de forma tão segura quanto você gerencia seu nome de usuário e sua senha.

Important

Não forneça as chaves de acesso a terceiros, mesmo que seja para ajudar a [encontrar o ID de usuário canônico](#). Ao fazer isso, você pode dar a alguém acesso permanente à sua conta.

Ao criar um par de chaves de acesso, você é solicitado a guardar o ID da chave de acesso e a chave de acesso secreta em um local seguro. A chave de acesso secreta só está disponível no momento em que é criada. Se você perder sua chave de acesso secreta, será necessário adicionar novas chaves de acesso para seu usuário do IAM. Você pode ter no máximo duas chaves de acesso. Se você já tiver duas, você deverá excluir um par de chaves para poder criar um novo. Para visualizar as instruções, consulte [Gerenciar chaves de acesso](#) no Guia do usuário do IAM.

Sou administrador e desejo permitir que outras pessoas tenham acesso ao Lambda

Para permitir que outros usuários acessem o Lambda, é necessário criar uma entidade do IAM (usuário ou função) para a pessoa ou a aplicação que precisa do acesso. Eles usarão as credenciais dessa entidade para acessar a AWS. Você deve anexar uma política à entidade que concede a eles as permissões corretas no Lambda.

Para começar a usar imediatamente, consulte [Criar os primeiros usuário e grupo delegados do IAM](#) no Guia do usuário do IAM.

Sou administrador e quero migrar de políticas gerenciadas da AWS para o Lambda que ficarão defasadas

Depois de 1º de março de 2021, as AWS políticas gerenciadas da AWSLambdaReadOnlyAccess e AWSLambdaFullAccess ficarão defasadas e não poderão mais ser associadas a novos usuários do IAM. Para obter informações sobre políticas defasadas, consulte [Políticas gerenciadas defasadas da AWS](#) no Manual do usuário do IAM.

O Lambda introduziu duas novas políticas gerenciadas da AWS:

- A política AWSLambda_ReadOnlyAccess concede acesso somente leitura ao Lambda, aos recursos de console do Lambda e a outros serviços da AWS relacionados. Esta política foi criada com a redução do escopo da política anterior AWSLambdaReadOnlyAccess.
- A política AWSLambda_FullAccess concede acesso total ao Lambda, aos recursos de console do Lambda e a outros serviços da AWS relacionados. Esta política foi criada com a redução do escopo da política anterior AWSLambdaFullAccess.

Usar as políticas gerenciadas da AWS

Recomendamos usar as políticas gerenciadas recém-lançadas para conceder acesso ao Lambda a usuários, grupos e funções. No entanto, revise as permissões concedidas nas políticas para garantir que elas atendam aos seus requisitos.

- Para revisar as permissões da política AWSLambda_ReadOnlyAccess, consulte a página dela [AWSLambda_ReadOnlyAccess](#) no console do IAM.
- Para revisar as permissões da política AWSLambda_FullAccess, consulte a página da política [AWSLambda_FullAccess](#) no console do IAM.

Depois de analisar as permissões, você pode associar as políticas a uma identidade do IAM (grupos, usuários ou funções). Para obter instruções sobre como associar uma política gerenciada da AWS, consulte [Adicionar e remover permissões de identidade do IAM](#) no Manual do usuário do IAM.

Usar políticas gerenciadas pelo cliente

Se você precisar de um controle de acesso mais refinado ou quiser adicionar permissões, poderá criar suas próprias [políticas gerenciadas pelo cliente](#). Para obter mais informações [Criar políticas na guia JSON](#) no Manual do usuário do IAM.

Desejo permitir que pessoas fora da minha conta da AWS acessem meus recursos do Lambda

Você pode criar uma função que os usuários de outras contas ou pessoas fora da sua organização podem usar para acessar seus recursos. Você pode especificar quem é confiável para assumir a função. Para

serviços que oferecem suporte a políticas baseadas em recursos ou listas de controle de acesso (ACLs), você pode usar essas políticas para conceder às pessoas acesso aos seus recursos.

Para saber mais, consulte o seguinte:

- Para saber se o Lambda oferece suporte a esses recursos, consulte [Como o AWS Lambda funciona com o IAM \(p. 733\)](#).
- Para saber como conceder acesso a seus recursos em todas as Contas da AWS pertencentes a você, consulte [Fornecimento de acesso a um usuário do IAM em outra Conta da AWS pertencente a você](#) no Guia de usuário do IAM.
- Para saber como conceder acesso a seus recursos para Contas da AWS de terceiros, consulte [Fornecimento de acesso a Contas da AWS pertencentes a terceiros](#) no Manual do usuário do IAM.
- Para saber como conceder acesso por meio da federação de identidades, consulte [Conceder acesso a usuários autenticados externamente \(federação de identidades\)](#) no Manual do usuário do IAM.
- Para saber a diferença entre usar funções e políticas baseadas em recursos para acesso entre contas, consulte [Como as funções do IAM diferem de políticas baseadas em recursos](#) no Guia do usuário do IAM.

Validação de conformidade do AWS Lambda

Auditores de terceiros avaliam a segurança e a conformidade do AWS Lambda como parte de vários programas de conformidade da AWS. Isso inclui SOC, PCI, FedRAMP, HIPAA e outros.

Para obter uma lista dos serviços da AWS no escopo de programas de conformidade específicos, consulte [Serviços da AWS no escopo por programa de conformidade](#). Para obter informações gerais, consulte [Programas de conformidade da AWS](#).

Você pode fazer download de relatórios de auditoria de terceiros usando o AWS Artifact. Para obter mais informações, consulte [Downloading reports in AWS Artifact](#).

Sua responsabilidade de conformidade ao usar o Lambda é determinada pela confidencialidade dos seus dados, pelos objetivos de compatibilidade da sua empresa e pelos regulamentos e leis aplicáveis. A AWS fornece os seguintes recursos para ajudar com a compatibilidade:

- [Guias de início rápido de segurança e conformidade](#): esses guias de implantação abordam as considerações de arquitetura e fornecem etapas para a implantação de ambientes de linha de base concentrados em compatibilidade e segurança na AWS.
- [Whitepaper Arquitetura para segurança e compatibilidade com a HIPAA](#): esse whitepaper descreve como as empresas podem usar a AWS para criar aplicações compatíveis com a HIPAA.
- [Recursos de compatibilidade da AWS](#): essa coleção de manuais e guias pode ser aplicável a seu setor e local.
- [AWS Config](#): esse serviço da AWS avalia até que ponto suas configurações de recursos atendem adequadamente às práticas internas e às diretrizes e regulamentações do setor.
- [AWS Security Hub](#): esse serviço da AWS fornece uma visão abrangente do estado de sua segurança na AWS que ajuda você a verificar sua conformidade com padrões e práticas recomendadas de segurança do setor.

Resiliência no AWS Lambda

A infraestrutura global da AWS é criada com base em regiões e zonas de disponibilidade da AWS. As regiões fornecem várias zonas de disponibilidade separadas e isoladas fisicamente, que são

conectadas com baixa latência, altas taxas de transferência e redes altamente redundantes. Com as zonas de disponibilidade, você pode projetar e operar aplicativos e bancos de dados que executam o failover automaticamente entre as zonas de disponibilidade sem interrupção. As zonas de disponibilidade são mais altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de data center tradicionais.

Para obter mais informações sobre regiões e zonas de disponibilidade da AWS, consulte [Infraestrutura global da AWS](#).

Além da infraestrutura global da AWS, o Lambda oferece vários recursos para ajudar a oferecer suporte às suas necessidades de resiliência de dados e backup.

- Versionamento: você pode usar o versionamento no Lambda para salvar a configuração e o código da sua função à medida que os desenvolve. Junto com aliases, você pode usar o versionamento para executar implantações azul/verde e contínuas. Para obter mais detalhes, consulte [Versões da função do Lambda \(p. 106\)](#).
- Escalabilidade— quando a função recebe uma solicitação enquanto está processando uma solicitação anterior, o Lambda executa outra instância da função para lidar com o aumento de carga. O Lambda é dimensionado automaticamente para lidar com 1.000 execuções simultâneas por região, um [quota \(p. 53\)](#) que pode ser aumentada, se necessário. Para obter mais detalhes, consulte [Escalabilidade da função do Lambda \(p. 32\)](#).
- Alta disponibilidade: o Lambda executa sua função em várias zonas de disponibilidade para garantir que ela esteja disponível para processar eventos no caso de uma interrupção do serviço em uma única zona. Se você configurar a função para se conectar a uma nuvem privada virtual (VPC) na sua conta, especifique sub-redes em várias zonas de disponibilidade para garantir uma alta disponibilidade. Para obter mais detalhes, consulte [Configurar uma função do Lambda para acessar recursos em uma VPC \(p. 124\)](#).
- Simultaneidade reservada: para garantir que sua função sempre pode se dimensionar para manipular solicitações adicionais, você pode reservar simultaneidade para ela. Configurar a simultaneidade reservada para uma função garante que ela pode se dimensionar, mas não exceder, um número especificado de invocações simultâneas. Isso garante que você não perderá solicitações por causa de outras funções que estejam consumindo toda a simultaneidade disponível. Para obter mais detalhes, consulte [Gerenciar simultaneidade para uma função do Lambda \(p. 113\)](#).
- Tentativas: para invocações assíncronas e um subconjunto de invocações acionadas por outros serviços, o Lambda faz novas tentativas automaticamente sobre um erro com um atraso entre as tentativas. Outros clientes e serviços da AWS que invocam funções de forma síncrona são responsáveis por realizar novas tentativas. Para obter mais detalhes, consulte [Lidar com erros e novas tentativas automáticas no AWS Lambda \(p. 176\)](#).
- Fila de mensagens mortas: para invocações assíncronas, você pode configurar o Lambda para enviar solicitações para uma fila de mensagens mortas caso todas as tentativas falhem. Uma fila de mensagens mortas é um tópico do Amazon SNS ou uma fila do Amazon SQS que recebe eventos para solução de problemas ou reprocessamento. Para obter mais detalhes, consulte [Filas de mensagens mortas \(p. 167\)](#).

Segurança da infraestrutura no AWS Lambda

Como um serviço gerenciado, o AWS Lambda é protegido pelos procedimentos de segurança de rede global da AWS descritos nas [práticas recomendadas de segurança, identidade e compatibilidade](#).

Você usa chamadas de API publicadas pela AWS para acessar o Lambda por meio da rede. Os clientes devem oferecer suporte a Transport Layer Security (TLS) 1.0 ou posterior. Recomendamos TLS 1.2 ou posterior. Os clientes também devem ter suporte a conjuntos de criptografia com perfect forward secrecy (PFS) como Ephemeral Diffie-Hellman (DHE) ou Ephemeral Elliptic Curve Diffie-Hellman (ECDHE). A maioria dos sistemas modernos como Java 7 e versões posteriores oferece suporte a esses modos.

Além disso, as solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou você pode usar o [AWS Security Token Service](#) (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Análise de vulnerabilidade e configuração no AWS Lambda

O AWS Lambda fornece [tempos de execução \(p. 214\)](#) que executam o código de sua função em um ambiente de execução baseado no Amazon Linux. O Lambda é responsável por manter o software no tempo de execução e no ambiente de execução atualizado, lançando novos tempos de execução para novas linguagens e frameworks, e por descontinuar tempos de execução quando o software subjacente não é mais compatível.

Se você usar bibliotecas adicionais com sua função, será responsável por atualizá-las. É possível incluir bibliotecas adicionais no [pacote de implantação \(p. 12\)](#) ou em [camadas \(p. 85\)](#) que você associa à função. Você também pode criar [tempos de execução personalizados \(p. 255\)](#) e usar camadas para compartilhá-los com outras contas.

O Lambda substitui os tempos de execução quando o software no tempo de execução ou o ambiente de execução dele atinge o fim da vida. Quando o Lambda rejeita uma execução, você é responsável por migrar suas funções a um tempo de execução compatível na mesma linguagem ou framework. Para obter mais detalhes, consulte [Política de suporte ao tempo de execução \(p. 217\)](#).

Solução de problemas no Lambda

Os tópicos a seguir fornecem orientações para a solução de erros e problemas que você pode encontrar ao usar a API, o console ou as ferramentas do Lambda. Se encontrar um problema que não esteja listado aqui, você poderá usar o botão Feedback desta página para relatá-lo.

Para obter mais orientações sobre solução de problemas e respostas a perguntas comuns de suporte, acesse a [Central de Conhecimento da AWS](#).

Para obter mais informações sobre depuração e solução de problemas de aplicações do Lambda, consulte [Debugging](#) no Guia do operador do Lambda.

Tópicos

- [Solucionar problemas de implantação no Lambda \(p. 741\)](#)
- [Solucionar problemas de invocação no Lambda \(p. 744\)](#)
- [Solucionar problemas de execução no Lambda \(p. 748\)](#)
- [Solucionar problemas de redes no Lambda \(p. 749\)](#)
- [Solucionar problemas de imagem de contêiner no Lambda \(p. 750\)](#)

Solucionar problemas de implantação no Lambda

Quando você atualiza a função, o Lambda implanta a alteração executando novas instâncias da função com o código ou as configurações atualizados. Erros de implantação impedem que a nova versão seja usada e podem surgir de problemas com o pacote de implantação, o código, as permissões ou as ferramentas.

Ao implantar atualizações na função diretamente com a API do Lambda ou com um cliente, como a AWS CLI, é possível visualizar os erros do Lambda diretamente na saída. Se você usar serviços como o AWS CloudFormation, o AWS CodeDeploy ou o AWS CodePipeline, procure a resposta do Lambda nos logs ou no fluxo de eventos desses serviços.

Geral: A permissão foi negada/Não é possível carregar esse arquivo

Erro: EACCES: permissão negada, abra '/var/task/index.js'

Erro: não é possível carregar esse arquivo -- função

Erro: [Errno 13] Permissão negada: '/var/task/function.py'

O tempo de execução do Lambda precisa de permissão para ler os arquivos no pacote de implantação. É possível usar o comando `chmod` para alterar o modo de arquivo. Os exemplos de comandos a seguir tornam todos os arquivos e pastas no diretório atual legíveis por qualquer usuário.

```
chmod 644 $(find . -type f)
chmod 755 $(find . -type d)
```

Geral: Ocorre um erro ao acionar o updateFunctionCode

Erro: ocorreu um erro (RequestEntityTooLargeException) ao chamar a operação UpdateFunctionCode

Quando você faz upload de um pacote de implantação ou de um arquivamento de camada diretamente no Lambda, o tamanho do arquivo ZIP é limitado a 50 MB. Para fazer upload de um arquivo maior, armazene-o no Amazon S3 e use os parâmetros [S3Bucket e S3Key \(p. 966\)](#).

Note

Quando você faz upload de um arquivo diretamente com a AWS CLI, o AWS SDK ou de outra forma, o arquivo ZIP binário é convertido em base64, o que aumenta o tamanho dele em cerca de 30%. Para permitir isso e o tamanho de outros parâmetros na solicitação, o limite do tamanho real da solicitação que o Lambda aplica é maior. Por isso, o limite de 50 MB é aproximado.

Amazon S3: Código de erro PermanentRedirect.

Erro: Ocorreu um erro ao GetObject. Código de erro S3: PermanentRedirect. Mensagem de erro do S3: O bucket está nesta Região: us-east-2. Use esta Região para tentar novamente a solicitação

Ao carregar o pacote de implantação de uma função de um bucket do Amazon S3, o bucket deve estar na mesma Região que a função. Esse problema pode ocorrer ao especificar um objeto do Amazon S3 em uma chamada para [UpdateFunctionCode \(p. 965\)](#) ou usar o pacote e implantar comandos na AWS CLI ou na CLI do AWS SAM. Crie um bucket de artefato de implantação para cada Região em que você desenvolve aplicativos.

Geral: Não é possível localizar, não é possível carregar, não é possível importar, classe não encontrada, o arquivo ou diretório não existe

Erro: não é possível localizar o módulo "function"

Erro: não é possível carregar esse arquivo -- função

Erro: não é possível importar o módulo "function"

Erro: classe não encontrada: function.Handler

Erro: fork/exec /var/task/function: nenhum arquivo ou diretório

Erro: não é possível carregar o tipo "Function.Handler" do assembly "Function".

O nome do arquivo ou classe na configuração do handler da função não corresponde ao seu código. Consulte a entrada a seguir para obter mais informações.

Geral: Handler de método indefinido

Erro: index.handler está indefinido ou não foi exportado

Erro: Handler "handler" ausente no módulo "function"

Erro: método indefinido "handler" para #<LambdaHandler:0x000055b76ccebf98>

Erro: nenhum método público chamado handleRequest com a assinatura de método apropriada encontrada na classe function.Handler

Erro: não foi possível encontrar o método "handleRequest" no tipo "Function.Handler" do assembly "Function".

O nome do método do handler na configuração do handler da função não corresponde ao seu código. Cada tempo de execução define uma convenção de nomenclatura para os handlers, como [filename.methodname](#). O manipulador é o método no código da função que é executado pelo tempo de execução quando a função é invocada.

Para alguns idiomas, o Lambda fornece uma biblioteca com uma interface que espera que um método de handler tenha um nome específico. Para obter detalhes sobre a nomenclatura de handlers para cada idioma, consulte os tópicos a seguir.

- [Criar funções do Lambda com Node.js \(p. 511\)](#)
- [Criar funções do Lambda com Python \(p. 538\)](#)
- [Construir funções do Lambda com Ruby \(p. 567\)](#)
- [Construir funções do Lambda com Java \(p. 592\)](#)
- [Criar funções do Lambda com Go \(p. 633\)](#)
- [Construir funções do Lambda com C# \(p. 661\)](#)
- [Construir funções do Lambda com o PowerShell \(p. 692\)](#)

Lambda: InvalidParameterValueException ou RequestEntityToolargeException

Erro: InvalidParameterValueException: o Lambda não conseguiu configurar as variáveis de ambiente porque as variáveis de ambiente fornecidas excederam o limite de 4 KB. String medida: {"A1": "uSFeY5cyPiPn7AtnX5BsM..."}

Erro: RequestEntityTooLargeException: a solicitação deve ser menor do que 5.120 bytes para a operação UpdateFunctionConfiguration

O tamanho máximo do objeto de variáveis que é armazenado na configuração da função não deve exceder 4096 bytes. Isso inclui nomes de chaves, valores, aspas, vírgulas e colchetes. O tamanho total do corpo da solicitação HTTP também é limitado.

```
{  
    "FunctionName": "my-function",  
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:my-function",  
    "Runtime": "nodejs12.x",  
    "Role": "arn:aws:iam::123456789012:role/lambda-role",  
    "Environment": {  
        "Variables": {  
            "BUCKET": "my-bucket",  
            "KEY": "file.txt"  
        }  
    },  
    ...  
}
```

Neste exemplo, o objeto tem 39 caracteres e ocupa 39 bytes quando é armazenado (sem espaços em branco) como a string {"BUCKET": "my-bucket", "KEY": "file.txt"}. Caracteres ASCII padrão em valores de variáveis de ambiente usam um byte cada. Caracteres ASCII e Unicode estendidos podem usar entre 2 bytes e 4 bytes por caractere.

Lambda: InvalidParameterValueException

Erro: InvalidParameterValueException: o Lambda não conseguiu configurar as variáveis de ambiente porque as variáveis de ambiente fornecidas contêm chaves reservadas que atualmente não têm suporte para modificação.

O Lambda reserva algumas chaves de variáveis de ambiente para uso interno. Por exemplo, AWS_REGION é usada pelo tempo de execução para determinar a região atual e não pode ser substituída. Outras variáveis, como PATH, são usadas pelo tempo de execução, mas podem ser estendidas na configuração

de função. Para obter uma lista completa, consulte [Variáveis de ambiente com tempo de execução definido \(p. 102\)](#).

Solucionar problemas de invocação no Lambda

Quando você invoca uma função do Lambda, o Lambda valida a solicitação e verifica a capacidade de escalabilidade antes de enviar o evento para a função ou, na invocação assíncrona, para a fila de eventos. Erros de invocação podem ser causados por problemas com os parâmetros da solicitação, a estrutura do evento, as configurações da função, as permissões do usuário, as permissões de recursos ou os limites.

Se você invocar a função diretamente, verá todos os erros de invocação na resposta do Lambda. Se invocar a função de forma assíncrona, com um mapeamento de origem de evento ou por meio de outro serviço, você poderá encontrar erros em logs, em uma fila de mensagens mortas ou em um destino de evento com falha. As opções de tratamento de erros e o comportamento de repetição variam dependendo de como você invoca a função e do tipo de erro.

Para obter uma lista de tipos de erro que a operação `Invoke` pode retornar, consulte [Invoke \(p. 875\)](#).

IAM: lambda:InvokeFunction não autorizado

Erro: usuário: arn:aws:iam::123456789012:user/developer não tem autorização para executar:
`lambda:InvokeFunction` no recurso: my-function

O seu usuário do AWS Identity and Access Management (IAM), ou a função que você assume, precisa ter permissão para invocar uma função. Esse requisito também se aplica a funções do Lambda e a outros recursos de computação que invocam funções. Adicione a política gerenciada da AWS `AWSLambdaRole` ao seu usuário do IAM, ou adicione uma política personalizada que permita a ação `lambda:InvokeFunction` na função de destino.

Note

Ao contrário de outras operações da API do Lambda, o nome da ação do IAM (`lambda:InvokeFunction`) não corresponde ao nome da operação da API (`Invoke`) para invocar uma função.

Para obter mais informações, consulte [AWS LambdaPermissões \(p. 56\)](#).

Lambda: A operação não pode ser executada ResourceConflictException

Erro: ResourceConflictException: não é possível executar a operação neste momento. A função está atualmente no seguinte estado: Pendente

Quando você conecta uma função a uma virtual private cloud (VPC) no momento da criação, a função entra em estado `Pending` enquanto o Lambda cria interfaces de rede elástica. Durante esse período, não será possível invocar ou modificar sua função. Se conectar sua função a uma VPC após a criação, você poderá invocá-la enquanto a atualização estiver pendente, mas não poderá modificar seu código ou configuração.

Para obter mais informações, consulte [Monitorar o estado de uma função com a API do Lambda \(p. 174\)](#).

Lambda: A função está paralisada em Pendente

Erro: uma função está presa no estado `Pending` há vários minutos.

Se uma função ficar paralisada no estado `Pending` por mais de seis minutos, chame uma das operações da API a seguir para desbloqueá-la:

- [UpdateFunctionCode \(p. 965\)](#)
- [UpdateFunctionConfiguration \(p. 974\)](#)
- [PublishVersion \(p. 919\)](#)

O Lambda cancela a operação pendente e coloca a função no estado `Failed`. Depois, você pode excluir a função e criá-la novamente ou tentar outra atualização.

Lambda: Uma função está usando toda a simultaneidade

Problema: uma função está usando toda a simultaneidade disponível, fazendo com que outras funções fiquem limitadas.

Para dividir a simultaneidade disponível da sua conta da AWS em uma região da AWS em grupos, use a [simultaneidade reservada \(p. 113\)](#). A simultaneidade reservada garante que uma função sempre possa escalar para a sua simultaneidade atribuída, e que ela não escale além da simultaneidade atribuída.

Geral: Não é possível invocar a função com outras contas ou serviços

Problema: você consegue invocar sua função diretamente, mas ela não é executada quando outro serviço ou conta a invoca.

Você concede a [outros serviços \(p. 277\)](#) e contas permissão para invocar uma função na [política baseada em recursos \(p. 62\)](#) da função. Se o invocador estiver em outra conta, esse usuário também precisará da permissão [para invocar funções \(p. 67\)](#).

Geral: A invocação da função está em loop

Problema: a função é invocada continuamente em um loop.

Isso geralmente ocorre quando a sua função gerencia recursos no mesmo serviço da AWS que o aciona. Por exemplo, é possível criar uma função que armazena um objeto em um bucket do Amazon Simple Storage Service (Amazon S3) que é configurado com uma [notificação que invoca a função novamente \(p. 431\)](#). Para interromper a execução da função, na [página de configuração da função \(p. 95\)](#), escolha Throttle (Restringir). Em seguida, identifique o caminho de código ou erro de configuração que gerou a invocação recursiva.

Lambda: Roteamento de alias com simultaneidade provisionada

Problema: Chamadas de spillover de simultaneidade provisionadas durante o roteamento de alias.

O Lambda usa um modelo probabilístico simples para distribuir o tráfego entre as duas versões de função. Em níveis de tráfego baixos, você pode ver uma alta variação entre a porcentagem configurada e real de tráfego em cada versão. Se sua função usa simultaneidade provisionada, você pode evitar [Invocações de transbordamento \(p. 717\)](#) configurando um número maior de instâncias de simultaneidade provisionadas durante o tempo em que o roteamento de alias está ativo.

Lambda: As inicializações a frio começam com simultaneidade provisionada

Problema: Você vê inicializações a frio depois de habilitar a simultaneidade provisionada.

Quando o número de execuções simultâneas em uma função é menor ou igual ao [nível configurado de simultaneidade provisionada \(p. 116\)](#), não deve haver nenhuma inicialização a frio. Para ajudar a confirmar se a simultaneidade provisionada está operando normalmente, faça o seguinte:

- [Verifique se a simultaneidade provisionada está habilitada \(p. 116\)](#) na versão ou alias da função.

Note

A simultaneidade provisionada não é configurável na [versão \\$LATEST \(p. 90\)](#).

- Certifique-se de que seus gatilhos invoquem a versão ou alias correto da função. Por exemplo, se você estiver usando o Amazon API Gateway, verifique se ele invoca a versão ou o alias da função com simultaneidade provisionada, e não a \$LATEST. Para confirmar se a simultaneidade provisionada está em uso, você pode verificar a [métrica ProvisionedConcurrencyInvocations do Amazon CloudWatch \(p. 717\)](#). Um valor diferente de zero indica que a função está processando invocações em ambientes de execução inicializados.
- Determine se a simultaneidade da função excede o nível configurado de simultaneidade provisionada verificando a [métrica ProvisionedConcurrencySpilloverInvocations do CloudWatch \(p. 717\)](#). Um valor diferente de zero indica que toda a simultaneidade provisionada está em uso e que ocorreu alguma invocação com inicialização a frio.
- Verifique a sua [frequência de invocação \(p. 53\)](#) (solicitações por segundo). Funções com simultaneidade provisionada apresentam uma taxa máxima de dez solicitações por segundo por simultaneidade provisionada. Por exemplo, uma função configurada com 100 casos de simultaneidade provisionada pode lidar com 1.000 solicitações por segundo. Se a taxa de invocação exceder 1.000 solicitações por segundo, podem ocorrer algumas inicializações a frio.

Note

Há um problema conhecido em que a primeira invocação em um ambiente de execução inicializado relata uma métrica diferente de zero Init Duration (Duração de inicialização) no CloudWatch Logs, mesmo que não ocorra nenhuma inicialização a frio. Estamos desenvolvendo uma correção para a geração de relatórios para o CloudWatch Logs.

Lambda: Variabilidade de latência com simultaneidade provisionada

Problema: Você vê a variabilidade de latência na primeira invocação depois de habilitar a simultaneidade provisionada.

Dependendo do tempo de execução e da configuração de memória da sua função, é possível ver alguma variabilidade de latência na primeira invocação em um ambiente de execução inicializado. Por exemplo, o .NET e outros tempos de execução JIT podem carregar recursos na primeira invocação de forma lenta, provocando alguma variabilidade de latência (geralmente dezenas de milissegundos). Essa variabilidade é mais aparente em funções de 128 MiB. Você mitiga isso aumentando a memória configurada da função.

Lambda: As inicializações a frio começam com novas versões

Problema: Você vê inicializações a frio ao implantar novas versões da sua função.

Quando você atualiza um alias da função, o Lambda automaticamente muda a simultaneidade provisionada para a nova versão com base nos pesos configurados no alias.

Erro: KMSDisabledException: o Lambda não conseguiu descriptografar as variáveis de ambiente porque a chave do KMS usada está desabilitada. Verifique as configurações da chave do KMS da função.

Esse erro pode ocorrer se a sua chave do AWS Key Management Service (AWS KMS) estiver desabilitada ou se a concessão que permite que o Lambda use a chave for revogada. Se a concessão estiver ausente, configure a função para usar outra chave. Em seguida, reatribua a chave personalizada para reciar a concessão.

EFS: A função não pôde montar o sistema de arquivos do EFS

Erro: EFSMountFailureException: a função não pôde montar o sistema de arquivos do EFS com o ponto de acesso arn:aws:elasticfilesystem:us-east-2:123456789012:access-point/fsap-015cxmplb72b405fd.

A solicitação de montagem para o [sistema de arquivos \(p. 139\)](#) da função foi rejeitada. Verifique as permissões da função e confirme se o sistema de arquivos e o ponto de acesso existem e estão prontos para uso.

EFS: A função não pôde se conectar ao sistema de arquivos do EFS

Erro: EFSMountConnectivityException: a função não pôde se conectar ao sistema de arquivos do Amazon EFS com o ponto de acesso arn:aws:elasticfilesystem:us-east-2:123456789012:access-point/fsap-015cxmplb72b405fd. Verifique a configuração de rede e tente novamente.

A função não pôde estabelecer uma conexão com o [sistema de arquivos \(p. 139\)](#) da função com o protocolo NFS (TCP porta 2049). Verifique a [configuração do grupo de segurança e de roteamento](#) das sub-redes da VPC.

EFS: A função não pôde montar o sistema de arquivos do EFS devido ao tempo limite

Erro: EFSMountTimeoutException: A função não pôde montar o sistema de arquivos do EFS com o ponto de acesso {arn:aws:elasticfilesystem:us-east-2:123456789012:access-point/fsap-015cxmplb72b405fd} devido ao tempo limite da montagem.

A função pôde se conectar ao [sistema de arquivos \(p. 139\)](#) da função, mas a operação de montagem atingiu o tempo limite. Tente novamente após um breve intervalo e considere limitar a [simultaneidade \(p. 113\)](#) da função para reduzir a carga no sistema de arquivos.

Lambda: O Lambda detectou um processo de E/S que estava demorando muito

EFSIOException: esta instância de função foi interrompida porque o Lambda detectou um processo de E/S que estava demorando muito.

Uma invocação anterior atingiu o tempo limite e o Lambda não conseguiu finalizar o handler de funções. Esse problema pode ocorrer quando um sistema de arquivos anexado fica sem crédito de intermitência e a taxa de transferência de linha de base é insuficiente. Para aumentar a taxa de transferência, você pode

aumentar o tamanho do sistema de arquivos ou usar a taxa de transferência provisionada. Para obter mais informações, consulte [Throughput \(p. 373\)](#).

Solucionar problemas de execução no Lambda

Quando o tempo de execução do Lambda executa o código de função, o evento pode ser processado em uma instância da função que esteja processando eventos por algum tempo, ou pode exigir que uma nova instância seja inicializada. Podem ocorrer erros durante a inicialização da função, quando o código do manipulador processa o evento, ou quando a função retorna (ou falha ao retornar) uma resposta.

Erros de execução de função podem ser causados por problemas com o código, a configuração de função, recursos de downstream ou permissões. Se invocar sua função diretamente, você verá erros de função na resposta do Lambda. Se invocar a função de forma assíncrona, com um mapeamento de origem de evento ou por meio de outro serviço, você poderá encontrar erros em logs, em uma fila de mensagens mortas ou em um destino em caso de falha. As opções de tratamento de erros e o comportamento de repetição variam dependendo de como você invoca a função e do tipo de erro.

Quando o código de função ou o tempo de execução do Lambda retornar um erro, o código de status na resposta do Lambda será 200 OK. A presença de um erro na resposta é indicada por um cabeçalho chamado `X-Amz-Function-Error`. Os códigos de status das séries 400 e 500 são reservados para [Erros de invocação \(p. 744\)](#).

Lambda: A execução leva muito tempo

Problema: a execução da função leva muito tempo.

Se o código demorar muito mais tempo para ser executado no Lambda do que na sua máquina local, ele poderá ser restrito pela memória ou pela potência de processamento disponível para a função. Configure a função com memória adicional (p. 95) para aumentar a memória e a CPU.

Lambda: Os logs ou rastreamentos não aparecem

Problema: Os logs não aparecem no CloudWatch Logs.

Problema: os rastreamentos não aparecem no AWS X-Ray.

Sua função precisa de permissão para chamar o CloudWatch Logs e o X-Ray. Atualize a função de execução (p. 57) para conceder permissão a ela. Adicione as políticas gerenciadas a seguir para habilitar logs e rastreamento.

- `AWSLambdaBasicExecutionRole`
- `AWSXRayDaemonWriteAccess`

Ao adicionar permissões à sua função, também atualize o código ou a configuração. Isso força as instâncias em execução da função, com credenciais desatualizadas, a serem encerradas e substituídas.

Note

Pode levar de 5 a 10 minutos para que os logs apareçam após uma invocação de função.

Lambda: A função retorna antes da conclusão da execução

Problema: (Node.js) a função retorna antes que o código termine de ser executado

Muitas bibliotecas, incluindo o AWS SDK, operam de forma assíncrona. Ao fazer uma chamada de rede ou executar outra operação que exija aguardar uma resposta, as bibliotecas retornam um objeto chamado de promessa que rastreia o progresso da operação em segundo plano.

Para aguardar que a promessa seja resolvida em uma resposta, use a palavra-chave `await`. Isso impedirá que o código do handler seja executado até que a promessa seja resolvida em um objeto que contenha a resposta. Se não precisar usar os dados da resposta em seu código, você poderá retornar a promessa diretamente para o tempo de execução.

Algumas bibliotecas não retornam promessas, mas podem ser encapsuladas em um código que retorne. Para obter mais informações, consulte [AWS LambdaManipulador da função do em Node.js \(p. 514\)](#).

AWS SDK: versões e atualizações

Problema: o AWS SDK incluído no tempo de execução não é a versão mais recente

Problema: o AWS SDK incluído no tempo de execução é atualizado automaticamente

Os tempos de execução para linguagens de desenvolvimento de scripts incluem o AWS SDK e são atualizados periodicamente para a versão mais recente. A versão atual para cada tempo de execução está listada na [página de tempo de execução \(p. 214\)](#). Para usar uma versão mais recente do AWS SDK ou para bloquear suas funções em uma versão específica, agrupe a biblioteca com seu código de função ou [crie uma camada do Lambda \(p. 85\)](#). Para obter detalhes sobre como criar um pacote de implantação com dependências, consulte os seguintes tópicos:

- [Implantar funções do Lambda em Node.js com arquivos .zip \(p. 517\)](#)
- [Implantar funções do Lambda em Python com arquivos .zip \(p. 543\)](#)
- [Implantar funções do Lambda em Ruby com arquivos .zip \(p. 571\)](#)
- [Implantar funções do Lambda em Java com arquivos .zip ou JAR \(p. 599\)](#)
- [Implantar funções do Lambda em Go com arquivos .zip \(p. 640\)](#)
- [Implantar funções do Lambda em C# com arquivos .zip \(p. 668\)](#)
- [Implantar funções do Lambda para PowerShell com arquivos .zip \(p. 694\)](#)

Python: As bibliotecas carregam incorretamente

Problema: (Python) algumas bibliotecas não carregam corretamente do pacote de implantação

Bibliotecas com módulos de extensão escritos em C ou C++ devem ser compiladas em um ambiente com a mesma arquitetura de processador que o Lambda (Amazon Linux). Para obter mais informações, consulte [Implantar funções do Lambda em Python com arquivos .zip \(p. 543\)](#).

Solucionar problemas de redes no Lambda

Por padrão, o Lambda executa suas funções em uma Virtual Private Cloud (VPC) interna com conectividade aos serviços da AWS e à Internet. Para acessar os recursos da rede local, é possível [configurar sua função para se conectar a uma VPC em sua conta \(p. 124\)](#). Ao usar esse recurso, você gerencia o acesso da função à Internet e a conectividade de rede com recursos da VPC.

Erros de conectividade de rede podem ser originados de problemas na configuração de roteamento, regras de grupo de segurança, permissões de função, conversão de endereço de rede ou disponibilidade de recursos, como endereços IP ou interfaces de rede. Esses itens podem resultar em um erro específico ou, se uma solicitação não puder chegar ao seu destino, em um tempo limite.

VPC: A função perde o acesso à Internet ou atinge o tempo limite

Problema: a função perde o acesso à Internet depois de se conectar a uma VPC

Erro: Erro: conexão ETIMEDOUT 176.32.98.189:443

Erro: Erro: a tarefa expirou após 10,00 segundos

Ao conectar uma função a uma VPC, todas as solicitações de saída passam pela VPC. Para se conectar à Internet, configure a VPC para enviar tráfego de saída da sub-rede da função para um gateway NAT em uma sub-rede pública. Para obter mais informações e exemplos de configurações de VPC, consulte [Acesso aos serviços e à Internet para funções conectadas à VPC \(p. 129\)](#).

VPC: a função precisa de acesso aos serviços da AWS sem usar a Internet

Problema: a função precisa de acesso aos serviços da AWS sem usar a Internet

Para se conectar aos serviços da AWS de uma sub-rede privada sem acesso à Internet, use VPC endpoints. Para obter um modelo de exemplo com endpoints da VPC para o DynamoDB e o Amazon S3, consulte [???](#) (p. 130).

VPC: O limite foi atingido para a VPC da função

Erro: ENILimitReachedException: o limite de interface de rede elástica foi atingido para a VPC da função.

Quando você conecta uma função a uma VPC, o Lambda cria uma interface de rede elástica para cada combinação de sub-rede e grupo de segurança anexado à função. Essas interfaces de rede são limitadas a 250 por VPC, mas esse limite pode ser aumentado. Para solicitar um aumento, use o [Console do centro de suporte](#).

Solucionar problemas de imagem de contêiner no Lambda

Contêiner: ocorre um erro no tempo de execução InvalidEntryPoint

Problema: você recebe uma mensagem de erro Runtime.ExitError, ou uma mensagem de erro com "errorType": "Runtime.InvalidEntryPoint".

Verifique se o ENTRYPOINT para sua imagem de contêiner inclui o caminho absoluto como o local. Verifique também se a imagem não contém um link simbólico (symlink) como ENTRYPOINT.

Lambda: capacidade adicional de provisionamento do sistema

Erro: "Error: We currently do not have sufficient capacity in the region you requested. Our system will be working on provisioning additional capacity. (Erro: no momento, não temos capacidade suficiente na região solicitada. Nosso sistema estará trabalhando no provisionamento de capacidade adicional.)

Tente invocar a função novamente. Se essa tentativa falhar, verifique se os arquivos necessários para executar o código da função podem ser lidos por qualquer usuário. O Lambda define um usuário padrão do Linux com permissões menos privilegiadas. Verifique se o código da aplicação não depende de arquivos com restrição de execução definida por outros usuários do Linux.

CloudFormation: ENTRYPPOINT está sendo substituído por um valor nulo ou vazio

Erro: You are using an AWS CloudFormation template, and your container ENTRYPPOINT is being overridden with a null or empty value. (Você está usando um modelo do CloudFormation e o ENTRYPPOINT do contêiner está sendo substituído por um valor nulo ou vazio.)

Revise o recurso `ImageConfig` no modelo do AWS CloudFormation. Se você declarar um recurso `ImageConfig` em seu modelo, deverá fornecer valores não vazios para todas as três propriedades.

AWS Lambda Versões do

A tabela a seguir descreve as alterações importantes feitas no Guia do desenvolvedor do AWS Lambda desde maio de 2018. Para receber notificações sobre atualizações dessa documentação, inscreva-se no [feed RSS](#).

update-history-change	atualização da descrição do histórico	atualização da data do histórico
Tempo de execução do Python 3.9	O Lambda é agora compatível com um novo tempo de execução para o Python 3.9. Para obter detalhes, consulte Tempos de execução do Lambda .	16 de agosto de 2021
Novas versões de tempo de execução para Node.js, Python e Java	Novas versões de tempo de execução estão disponíveis para Node.js, Python e Java. Para obter detalhes, consulte Tempos de execução do Lambda .	21 de julho de 2021
Compatível com RabbitMQ como uma fonte de eventos no Lambda	Agora, o Lambda é compatível com Amazon MQ for RabbitMQ como uma fonte de eventos. O Amazon MQ é um serviço gerenciado de agente de mensagem para o Apache ActiveMQ e o RabbitMQ que facilita a configuração e operação de agentes de mensagem na nuvem. Para obter detalhes, consulte Usar o Lambda com o Amazon MQ .	7 de julho de 2021
Autenticação SASL/PLAIN para Kafka autogerenciado no Lambda	Agora, o SASL/PLAIN é um mecanismo de autenticação compatível com fontes de eventos Kafka autogerenciadas no Lambda. Os clientes que já usam o SASL/PLAIN em seu cluster Kafka autogerenciado podem usar facilmente o Lambda para criar aplicações de consumidor sem ter que modificar a maneira como elas se autenticam. Para obter mais detalhes, consulte Usar o Lambda com Apache Kafka autogerenciado .	29 de junho de 2021
API de extensões do Lambda	Disponibilidade geral de extensões do Lambda. Use extensões para ampliar as funções do Lambda. É possível	24 de maio de 2021

	usar extensões fornecidas por parceiros do Lambda ou criar suas próprias extensões do Lambda. Para obter detalhes, consulte API de extensões do Lambda .	
Nova experiência de console do Lambda (p. 752)	O console do Lambda foi reprojetado para melhorar a performance e a consistência.	2 de março de 2021
Runtime do Node.js 14	Agora o Lambda é compatível com um novo tempo de execução para o Node.js 14. O Node.js 14 usa o Amazon Linux 2. Para obter detalhes, consulte Criar funções do Lambda com o Node.js .	27 de janeiro de 2021
Imagens de contêiner do Lambda	Agora o Lambda é compatível com funções definidas como imagens de contêiner. Você pode combinar a flexibilidade das ferramentas de contêineres com a agilidade e a simplicidade operacional do Lambda para criar aplicações. Para obter detalhes, consulte Usar imagens de contêiner com o Lambda .	1º de dezembro de 2020
Assinatura de código para funções do Lambda	O Lambda já é compatível com assinatura de código. Os administradores podem configurar funções do Lambda para aceitar apenas código assinado na implantação. O Lambda verifica as assinaturas para garantir que o código não seja alterado ou adulterado. Além disso, o Lambda confirma que o código seja assinado por desenvolvedores confiáveis antes de aceitar a implantação. Para obter detalhes, consulte Configurar assinatura de código para o Lambda .	23 de novembro de 2020
Previsualização: API Runtime Logs do Lambda	Agora o Lambda oferece suporte à API Runtime Logs. As extensões do Lambda podem usar a API Runtime Logs para assinar transmissões de logs no ambiente de execução. Para obter detalhes, consulte API Runtime Logs do Lambda .	12 de novembro de 2020

Nova fonte de eventos para o Amazon MQ	Agora, o Lambda é compatível com o Amazon MQ como uma fonte de eventos. Use uma função do Lambda para processar registros do seu agente de mensagens do Amazon MQ. Para obter detalhes, consulte Usar o Lambda com o Amazon MQ .	5 de novembro de 2020
Previsualização: API de extensões do Lambda	Use extensões do Lambda para ampliar as funções do Lambda. É possível usar extensões fornecidas por parceiros do Lambda ou criar suas próprias extensões do Lambda. Para obter detalhes, consulte API de extensões do Lambda .	8 de outubro de 2020
Suporte para Java 8 e tempos de execução personalizados no AL2	O Lambda já oferece suporte ao Java 8 e a tempos de execução personalizados no Amazon Linux 2. Para obter detalhes, consulte Tempos de execução do Lambda .	12 de agosto de 2020
Nova fonte de eventos para Amazon Managed Streaming for Apache Kafka	Agora, o Lambda é compatível com o Amazon MSK como uma fonte de eventos. Use uma função do Lambda com o Amazon MSK para processar registros em um tópico do Kafka. Para obter detalhes, consulte Usar o Lambda com o Amazon MSK .	11 de agosto de 2020
Chaves de condição do IAM para configurações da Amazon VPC	Agora você pode usar chaves de condição específicas do Lambda para configurações da VPC. Por exemplo, você pode exigir que todas as funções em sua organização estejam conectadas a uma VPC. Você também pode especificar as sub-redes e os grupos de segurança que os usuários da função podem e não podem usar. Para obter detalhes, consulte Configurar a VPC para funções do IAM .	10 de agosto de 2020

Configurações de simultaneidade para consumidores de transmissões HTTP/2 do Kinesis	Agora é possível usar as seguintes configurações de simultaneidade para consumidores do Kinesis com distribuição avançada (transmissões HTTP/2): ParallelizationFactor, MaximumRetryAttempts, MaximumRecordAgeInSeconds, DestinationConfig e BisectBatchOnFunctionError. Para obter detalhes, consulte Usar o AWS Lambda com o Amazon Kinesis .	7 de julho de 2020
Janela de lote para consumidores de transmissões do HTTP/2 do Kinesis	Agora é possível configurar uma janela de lote (MaximumBatchingWindowInSeconds) para transmissões HTTP/2. O Lambda lê registros da transmissão até coletar um lote inteiro, ou até que a janela de lote expire. Para obter detalhes, consulte Usar o AWS Lambda com o Amazon Kinesis .	18 de junho de 2020
Compatível com sistemas de arquivos do Amazon EFS	Agora é possível conectar um sistema de arquivos do Amazon EFS às funções do Lambda para acesso compartilhado aos arquivos de rede. Para obter detalhes, consulte Configurar o acesso ao sistema de arquivos para funções do Lambda .	16 de junho de 2020
AWS CDK aplicações de exemplo no console do Lambda	O console do Lambda agora inclui aplicações de exemplo que usam o AWS Cloud Development Kit (CDK) para TypeScript. O AWS CDK é um framework que permite definir os recursos de sua aplicação em TypeScript, Python, Java ou .NET. Para obter um tutorial sobre como criar aplicativos, consulte Criar um aplicativo com entrega contínua no console do Lambda .	1 de junho de 2020
Compatível com o tempo de execução do .NET Core 3.1.0 em AWS Lambda	AWS LambdaAgora, o oferece suporte para o tempo de execução do .NET Core 3.1.0. Para obter detalhes, consulte CLI do .NET Core .	31 de março de 2020

Compatível com APIs HTTP do API Gateway	Documentação atualizada e expandida para uso do Lambda com o API Gateway, incluindo suporte para APIs HTTP. Adição de um aplicativo de exemplo que cria uma API e uma função com o AWS CloudFormation. Para obter detalhes, consulte Usar o Lambda com o Amazon API Gateway .	23 de março de 2020
Ruby 2.7	Um novo tempo de execução está disponível para Ruby 2.7, ruby2.7, que é o primeiro tempo de execução Ruby a usar o Amazon Linux 2. Para obter detalhes, consulte Criar funções do Lambda com o Ruby .	19 de fevereiro de 2020
Métricas de simultaneidade	Agora o Lambda gera relatórios sobre a métrica <code>ConcurrentExecutions</code> para todas as funções, aliases e versões. É possível visualizar um gráfico para essa métrica na página de monitoramento da função. Anteriormente, a métrica <code>ConcurrentExecutions</code> só era relatada no nível de conta e para funções que usam simultaneidade reservada. Para obter detalhes, consulte Métricas de função do AWS Lambda .	18 de fevereiro de 2020

Atualização dos estados de função	<p>Os estados de função agora são impostos para todas as funções por padrão. Quando você conecta uma função a uma VPC, o Lambda cria interfaces de rede elásticas compartilhadas. Isso permite que sua função aumente sem criar interfaces de rede adicionais. Durante esse período, não será possível executar operações adicionais na função, incluindo a atualização de sua configuração e a publicação de versões. Em alguns casos, a invocação também será afetada. Detalhes sobre o estado atual de uma função estão disponíveis na API do Lambda.</p> <p>Essa atualização está sendo lançada em fases. Para obter detalhes, consulte Updated Lambda states lifecycle for VPC networking no blog de computação da AWS. Para obter mais informações sobre estados, consulte Estados de função do AWS Lambda.</p>	24 de janeiro de 2020
Atualizações na saída da API de configuração de função	<p>Foram adicionados códigos de motivo para StateReasonCode (<code>InvalidSubnet</code>, <code>InvalidSecurityGroup</code>) and LastUpdateStatusReasonCode (<code>SubnetOutOfRangeAddresses</code>, <code>InvalidSubnet</code>, <code>InvalidSecurityGroup</code>) para funções que se conectam a uma VPC. Para obter mais informações sobre estados, consulte Estados de função do AWS Lambda.</p>	20 de janeiro de 2020
Simultaneidade provisionada	<p>Agora é possível alocar a simultaneidade provisionada para um alias ou uma versão de função. A simultaneidade provisionada permite que uma função seja dimensionada sem flutuações na latência. Para obter detalhes, consulte Gerenciar a simultaneidade para uma função do Lambda.</p>	3 de dezembro de 2019

Criar um proxy de banco de dados	Agora é possível usar o console do Lambda para criar um proxy de banco de dados para uma função do Lambda. Um proxy de banco de dados permite que uma função atinja altos níveis de simultaneidade sem esgotar as conexões de banco de dados. Para obter detalhes, consulte Configurar o acesso ao banco de dados para uma função do Lambda .	3 de dezembro de 2019
Suporte a percentis para a métrica de duração	Agora é possível filtrar a métrica de duração com base em percentis. Para obter detalhes, consulte Métricas do AWS Lambda .	26 de novembro de 2019
Aumento da simultaneidade para origens de eventos de fluxos	Uma nova opção para mapeamentos de fontes de eventos de transmissão do DynamoDB e de transmissão do Kinesis permite processar mais de um lote por vez de cada fragmento. Ao aumentar o número de lotes simultâneos por estilhaço, a simultaneidade da sua função pode ser até 10 vezes o número de estilhaços no fluxo. Para obter detalhes, consulte Mapeamentos da fonte de eventos do AWS Lambda .	25 de novembro de 2019
Estados de função	Quando uma função é criada ou atualizada, ela entra em um estado pendente enquanto o Lambda provisiona recursos para oferecer suporte a ela. Se você conectar sua função a uma VPC, o Lambda poderá criar uma interface de rede elástica compartilhada imediatamente, em vez de criar interfaces de rede quando a função for invocada. Isso resulta em melhor performance para funções conectadas à VPC, mas pode exigir uma atualização para sua automação. Para obter detalhes, consulte Estados de função do AWS Lambda .	25 de novembro de 2019

Opções de manipulação de erros para invocação assíncrona	Novas opções de configuração estão disponíveis para invocação assíncrona. É possível configurar o Lambda para limitar novas tentativas e definir uma idade máxima do evento. Para obter detalhes, consulte Configurar o tratamento de erros para invocação assíncrona .	25 de novembro de 2019
Tratamento de erros para origens de eventos de fluxos	Novas opções de configuração estão disponíveis para mapeamentos de origem de evento que leem de fluxos. É possível configurar mapeamentos de fontes de eventos de transmissão do DynamoDB e transmissão do Kinesis para limitar as novas tentativas e definir uma idade máxima para os registros. Quando ocorrem erros, você pode configurar o mapeamento de origem de eventos para dividir lotes antes de tentar novamente e para enviar registros de invocação de lotes com falha para uma fila ou tópico. Para obter detalhes, consulte Mapeamentos da fonte de eventos do AWS Lambda .	25 de novembro de 2019
Destinos para invocação assíncrona	Agora é possível configurar o Lambda para enviar registros de invocações assíncronas a outro serviço. Os registros de invocação contêm detalhes sobre o evento, o contexto e a resposta da função. Você pode enviar registros de invocação para uma fila do SQS, um tópico do SNS, uma função do Lambda ou um barramento de eventos EventBridge. Para obter detalhes, consulte Configurar destinos para invocação assíncrona .	25 de novembro de 2019
Novos tempos de execução para Node.js, Python e Java	Novos tempos de execução estão disponíveis para Node.js 12, Python 3.8 e Java 11. Para obter detalhes, consulte Tempos de execução do Lambda .	18 de novembro de 2019

Suporte a fila FIFO para origens de eventos do Amazon SQS	Agora é possível criar um mapeamento de origem de eventos que leia a partir de uma fila FIFO (primeiro a entrar, primeiro a sair). Anteriormente, havia suporte apenas para filas padrão. Para obter detalhes, consulte Usar o Lambda com o Amazon SQS .	18 de novembro de 2019
Criar aplicações no console do Lambda	A criação de aplicações no console do Lambda já está disponível ao público em geral. Para obter instruções, consulte Criar uma aplicação com entrega contínua no console do Lambda .	31 de outubro de 2019
Criar aplicações no console do Lambda (beta)	Agora você pode criar uma aplicação do Lambda com um pipeline de entrega contínua integrado no console do Lambda. O console fornece aplicativos de exemplo que você pode usar como ponto de partida para seu próprio projeto. Escolha entre AWS CodeCommit e GitHub para controle de origem. Cada vez que você envia alterações para o repositório, o pipeline incluído faz sua compilação e implantação automaticamente. Para obter instruções, consulte Criar uma aplicação com entrega contínua no console do Lambda .	3 de outubro de 2019
Melhorias de performance para funções conectadas à VPC	Agora o Lambda usa um novo tipo de interface de rede elástica que é compartilhada por todas as funções em uma sub-rede da nuvem privada virtual (VPC). Quando você conecta uma função a uma VPC, o Lambda cria uma interface de rede para cada combinação de grupo de segurança e sub-rede escolhida. Quando as interfaces de rede compartilhadas estão disponíveis, a função não precisa mais criar interfaces de rede adicionais, já que ela é ampliada. Isso melhora significativamente os tempos de inicialização. Para obter detalhes, consulte Configurar uma função do Lambda para acessar recursos em uma VPC .	3 de setembro de 2019

Configurações de lote de fluxo	Agora você pode configurar uma janela de lote para mapeamentos de fontes de eventos do Amazon DynamoDB e do Amazon Kinesis. Configure uma janela de lote de até cinco minutos para armazenar em buffer os registros de entrada até que um lote completo esteja disponível. Isso reduz o número de vezes que sua função é invocada quando o fluxo está menos ativo.	29 de agosto de 2019
Integração de insights do CloudWatch Logs	A página de monitoramento no console do Lambda agora inclui relatórios do Amazon CloudWatch Logs Insights. Para obter detalhes, consulte Monitorar funções no console do AWS Lambda .	18 de junho de 2019
Amazon Linux 2018.03	O ambiente de execução do Lambda está sendo atualizado para usar o Amazon Linux 2018.03. Para obter detalhes, consulte Ambiente de execução .	21 de maio de 2019
Node.js 10	Um novo tempo de execução está disponível para Node.js 10, nodejs10.x. Esse tempo de execução usa o Node.js 10.15 e será atualizado com a última versão de ponto do Node.js 10 periodicamente. O Node.js 10 também é o primeiro tempo de execução para usar o Amazon Linux 2. Para obter detalhes, consulte Criar funções do Lambda com o Node.js .	13 de maio de 2019
API GetLayerVersionByArn	Use a API GetLayerVersionByArn para fazer download das informações sobre a versão da camada com o ARN da versão como entrada. Comparada com o GetLayerVersion, GetLayerVersionByArn permite que você use ARN diretamente, em vez de analisá-la para obter o nome da camada e o número da versão.	25 de abril de 2019
Ruby	AWS Lambda Agora o oferece suporte ao Ruby 2.5 com um novo tempo de execução. Para obter detalhes, consulte Criar funções do Lambda com o Ruby .	29 de novembro de 2018

Camadas	Com camadas do Lambda, é possível empacotar e implantar bibliotecas, tempos de execução personalizados e outras dependências separadamente do seu código da função. Compartilhe camadas com suas outras contas ou com o mundo inteiro. Para obter detalhes, consulte Camadas do Lambda .	29 de novembro de 2018
Tempos de execução personalizados	Construa um tempo de execução personalizado para executar funções do Lambda em sua linguagem de programação de preferência. Para obter detalhes, consulte Tempos de execução personalizados do Lambda .	29 de novembro de 2018
Triggers do Application Load Balancer	O Elastic Load Balancing é compatível com funções do Lambda como destino para um平衡ador de carga da aplicação. Para obter detalhes, consulte Usar o Lambda com balanceadores de carga da aplicação .	29 de novembro de 2018
Usar os consumidores de streaming do Kinesis HTTP/2 como um trigger	Você pode usar os consumidores de streaming de dados do Kinesis HTTP/2 para enviar eventos para o AWS Lambda. Os consumidores de streaming dedicam a taxa de transferência de leitura de cada estilhaço no seu streaming de dados e usam o HTTP/2 para minimizar a latência. Para obter detalhes, consulte Usar o Lambda com o Kinesis .	19 de novembro de 2018
Python 3.7	AWS Lambda agora oferece suporte ao Python 3.7 com um novo tempo de execução. Para obter mais informações, consulte Criar funções do Lambda com Python .	19 de novembro de 2018
Aumento do limite de carga da invocação da função assíncrona	O tamanho máximo de carga útil para invocações assíncronas aumentou de 128 KB para 256 KB, o que corresponde ao tamanho máximo da mensagem de um acionador do Amazon SNS. Para obter detalhes, consulte Cotas do Lambda .	16 de novembro de 2018

AWS Região GovCloud (Leste dos EUA)	Agora, o AWS Lambda está disponível na região GovCloud (EUA-Leste) da AWS.	12 de novembro de 2018
Transferência dos tópicos do AWS SAM para um Guia do desenvolvedor separado.	Vários tópicos estavam concentrados na criação de aplicativos sem servidor usando o AWS Serverless Application Model (AWS SAM). Esses tópicos foram movidos para o Guia do desenvolvedor do AWS Serverless Application Model .	25 de outubro de 2018
Visualizar aplicações do Lambda no console	Você pode visualizar o status de suas aplicações do Lambda na página Applications (Aplicações) no console do Lambda. Essa página mostra o status da pilha do AWS CloudFormation. Ela inclui links para páginas nas quais você pode visualizar mais informações sobre os recursos na pilha. Você também pode visualizar métricas agregadas para o aplicativo e criar painéis de monitoramento personalizados.	11 de outubro de 2018
Limite de tempo limite de execução da função	Para permitir funções de longa duração, o tempo limite máximo de execução configurável aumentou de 5 minutos para 15 minutos. Para obter detalhes, consulte Limites do Lambda .	10 de outubro de 2018
Suporte à linguagem do PowerShell Core no AWS Lambda	AWS Lambda Agora, o oferece suporte à linguagem do PowerShell Core. Para obter mais informações, consulte Modelo de programação para criação de funções do Lambda no PowerShell .	11 de setembro de 2018
Suporte ao tempo de execução do .NET Core 2.1.0 no AWS Lambda	AWS Lambda Agora, o oferece suporte para o tempo de execução do .NET Core 2.1.0. Para obter mais informações, consulte CLI do .NET Core .	9 de julho de 2018
Atualizações agora disponíveis em RSS	Agora você pode assinar um feed RSS para seguir as versões deste guia.	5 de julho de 2018

Compatível com o Amazon SQS como uma fonte de eventos	Agora o AWS Lambda oferece suporte ao Amazon Simple Queue Service (Amazon SQS) como uma fonte de eventos. Para obter mais informações, consulte Invocar funções do Lambda .	28 de junho de 2018
Região da China (Ningxia)	Agora o AWS Lambda está disponível na região da China (Ningxia). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	28 de junho de 2018

Atualizações anteriores

A tabela a seguir descreve as alterações importantes em cada versão do Guia do desenvolvedor do AWS Lambda antes de junho de 2018.

Alteração	Descrição	Data
Suporte ao tempo de execução para o Node.js 8.10	AWS Lambda Agora, o oferece suporte ao tempo de execução do Node.js versão 8.10. Para obter mais informações, consulte Criar funções do Lambda com Node.js (p. 511) .	2 de abril de 2018
IDs de revisão de função e de alias	AWS Lambda O agora oferece suporte aos IDs em suas versões e alias de função. Você pode usar esses IDs para monitorar e aplicar atualizações condicionais ao atualizar a versão de sua função ou os recursos de alias.	25 de janeiro de 2018
Suporte ao tempo de execução para o Go e o .NET 2.0	AWS Lambda O adicionou suporte ao tempo de execução para o Go e o .NET 2.0. Para obter mais informações, consulte Criar funções do Lambda com Go (p. 633) e Construir funções do Lambda com C# (p. 661) .	15 de janeiro de 2018
Novo design do console	O AWS Lambda introduziu um novo console do Lambda para simplificar sua experiência e adicionou um editor de código do Cloud9 para melhorar sua capacidade de depurar e revisar seu código de função. Para obter mais informações, consulte Editar código usando o editor do console (p. 40) .	30 de novembro de 2017
Configurar os limites de simultaneidade nas funções individuais	AWS Lambda O agora oferece suporte à definição de limites de simultaneidade nas funções individuais. Para obter mais informações, consulte Gerenciar simultaneidade para uma função do Lambda (p. 113) .	30 de novembro de 2017
Mudar o tráfego com aliases	AWS Lambda O agora oferece suporte à mudança de tráfego com aliases. Para obter mais informações, consulte Implantações contínuas para funções do Lambda (p. 206) .	28 de novembro de 2017
Implantação gradual de código	Agora o AWS Lambda oferece suporte para a implantação segura de novas versões da sua função do Lambda utilizando a implantação de código. Para obter mais informações, consulte Implantação gradual de código .	28 de novembro de 2017

Alteração	Descrição	Data
Região China (Pequim)	Agora o AWS Lambda está disponível na região da China (Pequim). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	9 de novembro de 2017
Introdução ao SAM Local	O AWS Lambda apresenta o SAM Local (conhecido agora como CLI do SAM), uma ferramenta da AWS CLI que oferece um ambiente para desenvolver, testar e analisar as aplicações sem servidor localmente antes de fazer upload delas no tempo de execução do Lambda. Para obter mais informações, consulte Teste e depuração de aplicativos sem servidor .	11 de agosto de 2017
Região do Canadá (Central)	AWS LambdaAgora o está disponível na região Canadá (Central). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	22 de junho de 2017
South America (São Paulo) Region	O AWS Lambda já está disponível na região América do Sul (São Paulo). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	6 de junho de 2017
AWS Lambda Suporte do para AWS X-Ray.	O Lambda apresenta suporte ao X-Ray, o que permite detectar, analisar e otimizar problemas de performance com suas aplicações do Lambda. Para obter mais informações, consulte Usar o AWS Lambda com o AWS X-Ray (p. 477) .	19 de abril de 2017
Asia Pacific (Mumbai) Region	O AWS Lambda já está disponível na região da Ásia-Pacífico (Mumbai). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	28 de março de 2017
AWS LambdaO agora oferece suporte ao tempo de execução do Node.js v6.10	AWS LambdaO adicionou suporte ao tempo de execução do Node.js v6.10. Para obter mais informações, consulte Criar funções do Lambda com Node.js (p. 511) .	22 de março de 2017
Região Europa (Londres)	Agora o AWS Lambda está disponível na região Europa (Londres). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	1 de fevereiro de 2017
AWS LambdaSuporte do ao tempo de execução do .NET, o Lambda@Edge (versão prévia), Dead Letter Queues e implantação automatizada de aplicativos sem servidor.	AWS LambdaO adicionou suporte ao C#. Para obter mais informações, consulte Construir funções do Lambda com C# (p. 661) . O Lambda@Edge permite que você execute funções do Lambda em todos os pontos de presença da AWS em resposta a eventos do CloudFront. Para obter mais informações, consulte Como usar o AWS Lambda com o Lambda@Edge do CloudFront (p. 325) .	3 de dezembro de 2016
O AWS Lambda adiciona o Amazon Lex como uma fonte de eventos compatível.	Usando o Lambda e o Amazon Lex, você pode construir rapidamente bots de bate-papo para vários serviços como o Slack e o Facebook. Para obter mais informações, consulte O uso do AWS LambdaCom o Amazon Lex (p. 408) .	30 de novembro de 2016

Alteração	Descrição	Data
US West (N. California) Region	O AWS Lambda já está disponível na região Oeste dos EUA (Norte da Califórnia). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	21 de novembro de 2016
Introduzido o AWS SAM para criação e implantação de aplicações baseadas no Lambda e usando variáveis de ambiente para definições de configuração da função do Lambda.	<p>AWS SAM: agora você pode usar o AWS SAM para definir a sintaxe para expressar recursos dentro de uma aplicação sem servidor. Para implantar o aplicativo, basta especificar os recursos de que você precisa como parte de seu aplicativo, junto com suas políticas de permissões associadas em um arquivo de modelo do AWS CloudFormation (escrito em JSON ou YAML), empacotar os artefatos de sua implantação e implantar o modelo. Para obter mais informações, consulte AWS LambdaAplicativos do (p. 192).</p> <p>Variáveis de ambiente: você pode usar variáveis de ambiente para especificar as definições de configuração para sua função do Lambda fora do código da função. Para obter mais informações, consulte Usar variáveis de ambiente do AWS Lambda (p. 99).</p>	18 de novembro de 2016
Asia Pacific (Seoul) Region	O AWS Lambda já está disponível na região da Ásia-Pacífico (Seul). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	29 de agosto de 2016
Asia Pacific (Sydney) Region	O Lambda já está disponível na região Ásia-Pacífico (Sydney). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	23 de junho de 2016
Atualizações no console do Lambda	O console do Lambda foi atualizado para simplificar o processo de criação de função. Para obter mais informações, consulte Criar uma função do Lambda com o console (p. 9) .	23 de junho de 2016
AWS LambdaO agora oferece suporte ao tempo de execução do Node.js v4.3	AWS LambdaO adicionou suporte ao tempo de execução do Node.js v4.3. Para obter mais informações, consulte Criar funções do Lambda com Node.js (p. 511) .	07 de abril de 2016
Região Europa (Frankfurt)	O Lambda já está disponível na região Europa (Frankfurt). Para obter mais informações sobre regiões e endpoints do Lambda, consulte Regiões e endpoints na Referência geral da AWS.	14 de março de 2016
Suporte à VPC	Agora você pode configurar uma função do Lambda para acessar recursos em sua VPC. Para obter mais informações, consulte Configurar uma função do Lambda para acessar recursos em uma VPC (p. 124) .	11 de fevereiro de 2016
O tempo de execução do Lambda foi atualizado.	O ambiente de execução (p. 214) foi atualizado.	4 de novembro de 2015

Alteração	Descrição	Data
Suporte ao versionamento, Python para desenvolvimento de código para funções do Lambda, eventos programados e aumento do tempo de execução	<p>Agora você pode desenvolver o código de sua função do Lambda usando o Python. Para obter mais informações, consulte Criar funções do Lambda com Python (p. 538).</p> <p>Versionamento: você pode manter uma ou mais versões de sua função do Lambda. O versionamento permite que você controle qual função de Lambda é executada em diferentes ambientes (por exemplo, desenvolvimento, teste ou produção). Para obter mais informações, consulte Versões da função do Lambda (p. 106).</p> <p>Eventos programados: você também pode configurar o Lambda para invocar seu código em base regular e programada usando o console do Lambda. Você pode especificar uma taxa fixa (número de horas, dias ou semanas) ou especificar uma expressão Cron. Para ver um exemplo, consulte O uso do AWS LambdaCom Amazon CloudWatch Events (p. 315).</p> <p>Aumento no tempo de execução: agora você pode configurar suas funções do Lambda para executar por até cinco minutos, permitindo funções em execução por mais tempo, como ingestão de dados de grande volume e trabalhos de processamento.</p>	08 de outubro de 2015
Suporte ao DynamoDB Streams	O DynamoDB Streams já está disponível e você pode usá-lo em todas as regiões onde o DynamoDB estiver disponível. Você pode habilitar o DynamoDB Streams para sua tabela e usar uma função de Lambda como um trigger para a tabela. Os triggers são ações personalizadas executadas em resposta a atualizações feitas na tabela do DynamoDB. Para ver uma demonstração de exemplo, consult Tutorial: Usar o AWS Lambda com o Amazon DynamoDB Streams (p. 345) .	14 de julho de 2015
O Lambda agora oferece suporte para invocar funções do Lambda com clientes compatíveis com REST.	<p>Até agora, para invocar a sua função do Lambda na Web, dispositivo móvel ou aplicação IoT, você precisava de AWS SDKs (por exemplo, AWS SDK for Java, AWS SDK for Android ou AWS SDK for iOS). Agora, o Lambda oferece suporte à invocação de uma função do Lambda com clientes compatíveis com REST por meio de uma API personalizada que você pode criar usando o Amazon API Gateway. Você pode enviar solicitações para a URL de endpoint de sua função Lambda. Você pode configurar a segurança no endpoint para permitir acesso aberto, utilizar o AWS Identity and Access Management (IAM) para autorizar acesso ou usar chaves de API para mensurar o acesso de outras pessoas às suas funções do Lambda.</p> <p>Para ver um exemplo de exercício de Conceitos básicos, consulte Usar o AWS Lambda com o Amazon API Gateway (p. 281).</p> <p>Para obter mais informações sobre o Amazon API Gateway, consultehttps://aws.amazon.com/api-gateway/.</p>	09 de julho de 2015

Alteração	Descrição	Data
O console do Lambda agora oferece esquemas para criar e testar funções do Lambda facilmente.	O console do Lambda fornece um conjunto de esquemas. Cada esquema fornece uma configuração de exemplo de fonte de evento e o código de exemplo para a função Lambda que você pode usar para criar aplicativos baseados em Lambda facilmente. Todos os exercícios de conceitos básicos do Lambda agora usam os esquemas. Para obter mais informações, consulte Conceitos básicos do Lambda (p. 8) .	09 de julho de 2015
O Lambda agora oferece suporte a Java para criar suas funções do Lambda.	Agora você pode criar código do Lambda em Java. Para obter mais informações, consulte Construir funções do Lambda com Java (p. 592) .	15 de junho de 2015
O Lambda agora oferece suporte à especificação de um objeto do Amazon S3, como a função .zip, ao criar ou atualizar uma função do Lambda.	Você pode fazer upload de um pacote de implantação de função Lambda (arquivo .zip) para um bucket do Amazon S3 na mesma região em que você deseja criar uma função Lambda. Em seguida, você pode especificar o nome do bucket e o nome da chave do objeto ao criar ou atualizar uma função Lambda.	28 de maio de 2015
O Lambda agora está disponível com suporte adicional para backends móveis	<p>O Lambda agora está disponível para uso em produção. A versão também apresenta novos recursos que tornam ainda mais fácil criar backends para celular, tablet e Internet das Coisas (IoT) usando o Lambda que é dimensionado automaticamente, sem provisionamento ou gerenciamento da infraestrutura. Agora, o Lambda oferece suporte a eventos em tempo real (síncronos) e com eventos assíncronos. Recursos adicionais incluem configuração e gerenciamento mais fáceis de origens de eventos. O modelo de permissões e o modelo de programação foram simplificados pela introdução de políticas de recursos para suas funções Lambda.</p> <p>A documentação foi atualizada de maneira adequada. Para obter informações, consulte os tópicos a seguir:</p> <p style="margin-left: 20px;">Conceitos básicos do Lambda (p. 8)</p> <p style="margin-left: 20px;">AWS Lambda</p>	9 de abril de 2015
Versão de visualização	Versão de visualização do (Developer Guide) Guia do desenvolvedor do AWS Lambda.	13 de novembro de 2014

Referência de API

Esta seção contém a documentação de referência da API AWS Lambda. Ao fazer chamadas de API, você precisará autenticar sua solicitação fornecendo uma assinatura. AWS Lambda dá suporte ao Signature versão 4. Para obter mais informações, consulte o [Processo de cadastro no Signature versão 4](#) na Referência geral da Amazon Web Services.

Para obter uma visão geral do serviço, consulte [O que é o AWS Lambda? \(p. 1\)](#).

Você pode usar a AWS CLI para explorar a API do AWS Lambda. Este guia fornece vários tutoriais que usam a AWS CLI.

Tópicos

- [Actions \(p. 769\)](#)
- [Tipos de dados \(p. 988\)](#)

Actions

As ações a seguir são compatíveis:

- [AddLayerVersionPermission \(p. 771\)](#)
- [AddPermission \(p. 775\)](#)
- [CreateAlias \(p. 779\)](#)
- [CreateCodeSigningConfig \(p. 783\)](#)
- [CreateEventSourceMapping \(p. 786\)](#)
- [CreateFunction \(p. 796\)](#)
- [DeleteAlias \(p. 808\)](#)
- [DeleteCodeSigningConfig \(p. 810\)](#)
- [DeleteEventSourceMapping \(p. 812\)](#)
- [DeleteFunction \(p. 818\)](#)
- [DeleteFunctionCodeSigningConfig \(p. 820\)](#)
- [DeleteFunctionConcurrency \(p. 822\)](#)
- [DeleteFunctionEventInvokeConfig \(p. 824\)](#)
- [DeleteLayerVersion \(p. 826\)](#)
- [DeleteProvisionedConcurrencyConfig \(p. 828\)](#)
- [GetAccountSettings \(p. 830\)](#)
- [GetAlias \(p. 832\)](#)
- [GetCodeSigningConfig \(p. 835\)](#)
- [GetEventSourceMapping \(p. 837\)](#)
- [GetFunction \(p. 842\)](#)
- [GetFunctionCodeSigningConfig \(p. 846\)](#)
- [GetFunctionConcurrency \(p. 849\)](#)
- [GetFunctionConfiguration \(p. 851\)](#)
- [GetFunctionEventInvokeConfig \(p. 858\)](#)
- [GetLayerVersion \(p. 861\)](#)

- [GetLayerVersionByArn \(p. 864\)](#)
- [GetLayerVersionPolicy \(p. 867\)](#)
- [GetPolicy \(p. 869\)](#)
- [GetProvisionedConcurrencyConfig \(p. 872\)](#)
- [Invoke \(p. 875\)](#)
- [InvokeAsync \(p. 881\)](#)
- [ListAliases \(p. 883\)](#)
- [ListCodeSigningConfigs \(p. 886\)](#)
- [ListEventSourceMappings \(p. 888\)](#)
- [ListFunctionEventInvokeConfigs \(p. 891\)](#)
- [ListFunctions \(p. 894\)](#)
- [ListFunctionsByCodeSigningConfig \(p. 898\)](#)
- [ListLayers \(p. 900\)](#)
- [ListLayerVersions \(p. 903\)](#)
- [ListProvisionedConcurrencyConfigs \(p. 906\)](#)
- [ListTags \(p. 909\)](#)
- [ListVersionsByFunction \(p. 911\)](#)
- [PublishLayerVersion \(p. 915\)](#)
- [PublishVersion \(p. 919\)](#)
- [PutFunctionCodeSigningConfig \(p. 927\)](#)
- [PutFunctionConcurrency \(p. 930\)](#)
- [PutFunctionEventInvokeConfig \(p. 933\)](#)
- [PutProvisionedConcurrencyConfig \(p. 937\)](#)
- [RemoveLayerVersionPermission \(p. 940\)](#)
- [RemovePermission \(p. 942\)](#)
- [TagResource \(p. 945\)](#)
- [UntagResource \(p. 947\)](#)
- [UpdateAlias \(p. 949\)](#)
- [UpdateCodeSigningConfig \(p. 953\)](#)
- [UpdateEventSourceMapping \(p. 956\)](#)
- [UpdateFunctionCode \(p. 965\)](#)
- [UpdateFunctionConfiguration \(p. 974\)](#)
- [UpdateFunctionEventInvokeConfig \(p. 985\)](#)

AddLayerVersionPermission

Adiciona permissões à política baseada em recursos de uma versão de uma [camada do AWS Lambda](#). Use essa ação para conceder permissão de uso da camada a outras contas. Você pode conceder permissão a uma única conta, a todas as contas da AWS em uma organização ou a todas as contas da AWS.

Para revogar a permissão, chame [RemoveLayerVersionPermission](#) (p. 940) com o ID da instrução que você especificou quando adicionou essa permissão.

Sintaxe da solicitação

```
POST /2018-10-31/layers/LayerName/versions/VersionNumber/policy?RevisionId=RevisionId
HTTP/1.1
Content-type: application/json

{
  "Action": "string",
  "OrganizationId": "string",
  "Principal": "string",
  "StatementId": "string"
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[LayerName](#) (p. 771)

O nome ou o nome de recurso da Amazon (ARN) da camada.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (`arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_+]|[a-zA-Z0-9-_]+`)

Obrigatório: sim

[RevisionId](#) (p. 771)

Atualize a política somente se o ID da revisão corresponder ao ID especificado. Use essa opção para evitar a modificação de uma política que foi alterada desde a última leitura.

[VersionNumber](#) (p. 771)

O número da versão.

Obrigatório: sim

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[Action](#) (p. 771)

A ação da API que concede acesso à camada. Por exemplo, `lambda:GetLayerVersion`.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 22.

Pattern: `lambda:GetLayerVersion`

Obrigatório: sim

[OrganizationId \(p. 771\)](#)

Com o principal definido como *, conceda permissão a todas as contas na organização especificada.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 34.

Pattern: `o-[a-zA-Z0-9]{10,32}`

Exigido: Não

[Principal \(p. 771\)](#)

Um ID de conta ou * para conceder permissão de uso da camada a todas as contas de uma organização ou a todas as contas da AWS (se organizationId não for especificado). Para o último caso, certifique-se de que você realmente deseja que todas as contas da AWS tenham permissão de uso para essa camada.

Tipo: sequência

Pattern: `\d{12}|*|arn:(aws[a-zA-Z-]*):iam::\d{12}:root`

Obrigatório: sim

[StatementId \(p. 771\)](#)

Um identificador que distingue uma política de outras na mesma versão de camada.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 100.

Pattern: `([a-zA-Z0-9-_]+)`

Obrigatório: sim

Sintaxe da resposta

```
HTTP/1.1 201
Content-type: application/json

{
  "RevisionId": "string",
  "Statement": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço retornará uma resposta HTTP 201.

Os seguintes dados são retornados no formato JSON pelo serviço.

[RevisionId \(p. 772\)](#)

Um identificador exclusivo da revisão atual da política.

Tipo: sequência
[Statement \(p. 772\)](#)

A declaração da permissão.

Tipo: sequência

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

PolicyLengthExceededException

A política de permissões do recurso é muito grande. [Saiba mais](#)

Código de status HTTP: 400

PreconditionFailedException

O RevisionId fornecido não corresponde ao RevisionId mais recente da função ou do alias do Lambda. Chame a API GetFunction ou GetAlias para recuperar o RevisionId mais recente para o seu recurso.

Código de status HTTP: 412

ResourceConflictException

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)

- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

AddPermission

Concede a um serviço da AWS ou outra conta a permissão para usar uma função. Você pode aplicar a política no nível da função ou especificar um qualificador para restringir o acesso a uma única versão ou alias. Se você usar um qualificador, o chamador deverá usar o nome de recurso da Amazon (ARN) completo da versão ou alias para invocar a função. Observação: o Lambda não oferece suporte à adição de políticas à versão \$LATEST.

Para conceder permissão a outra conta, especifique o ID da conta como o `Principal`. Para serviços da AWS, o principal é um identificador em estilo de domínio definido pelo serviço, como `s3.amazonaws.com` ou `sns.amazonaws.com`. Para serviços da AWS, você também pode especificar o ARN do recurso associado como o `SourceArn`. Se você conceder permissão a um principal do serviço sem especificar a origem, outras contas poderão potencialmente configurar recursos em suas contas para invocar sua função do Lambda.

Essa ação adiciona uma instrução a uma política de permissões baseada em recursos de sua função. Para obter mais informações sobre políticas de função, consulte [Políticas de função do Lambda](#).

Sintaxe da solicitação

```
POST /2015-03-31/functions/FunctionName/policy?Qualifier=Qualifier HTTP/1.1
Content-type: application/json

{
  "Action": "string",
  "EventSourceToken": "string",
  "Principal": "string",
  "RevisionId": "string",
  "SourceAccount": "string",
  "SourceArn": "string",
  "StatementId": "string"
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName \(p. 775\)](#)

O nome da função, versão ou alias do Lambda.

Formatos de nome

- Function name - `my-function` (somente nome), `my-function:v1` (com alias).
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- ARN parcial - `123456789012:function:my-function`.

Você pode anexar um número de versão ou alias a qualquer um dos formatos. A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (`arn:(aws[a-zA-Z-]*)?:lambda:`)?([a-z]{2}(-gov)?-[a-z]+\-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$\LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[Qualifier \(p. 775\)](#)

Especifique uma versão ou alias para adicionar permissões a uma versão publicada da função.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: (| [a-zA-Z0-9\$_-]+)

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[Action \(p. 775\)](#)

A ação que o principal pode usar na função. Por exemplo, o `lambda:InvokeFunction` ou o `lambda:GetFunction`.

Tipo: sequência

Pattern: (`lambda:[*]` | `lambda:[a-zA-Z]+[*]`)

Obrigatório: sim

[EventSourceToken \(p. 775\)](#)

Em funções do Alexa Smart Home, um token que deve ser fornecido pelo chamador.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

Pattern: [a-zA-Z0-9._\-\-]+

Exigido: Não

[Principal \(p. 775\)](#)

O serviço ou conta da AWS que invoca a função. Se você especificar um serviço, use `SourceArn` ou `SourceAccount` para limitar quem pode invocar a função por meio desse serviço.

Tipo: sequência

Pattern: [^\s]+

Obrigatório: sim

[RevisionId \(p. 775\)](#)

Atualize a política somente se o ID da revisão corresponder ao ID especificado. Use essa opção para evitar a modificação de uma política que foi alterada desde a última leitura.

Tipo: string

Exigido: Não

[SourceAccount \(p. 775\)](#)

Para o Amazon S3, o ID da conta que é a proprietária do recurso. Use em conjunto com `SourceArn` para garantir que o recurso é de propriedade da conta especificada. É possível que um bucket do Amazon S3 seja excluído pelo proprietário e recriado por outra conta.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 12.

Pattern: \d{12}

Exigido: Não

[SourceArn \(p. 775\)](#)

Em serviços da AWS, o ARN do recurso da AWS que invoca a função. Por exemplo, um bucket do Amazon S3 ou um tópico do Amazon SNS.

Observe que o Lambda configura a comparação usando o operador `StringLike`.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*)([a-zA-Z0-9\-])+([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12}):(.*)

Exigido: Não

[StatementId \(p. 775\)](#)

Um identificador de instrução que diferencia a instrução de outras na mesma política.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 100.

Pattern: ([a-zA-Z0-9_-]+)

Obrigatório: sim

Sintaxe da resposta

```
HTTP/1.1 201
Content-type: application/json

{
  "Statement": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço retornará uma resposta HTTP 201.

Os seguintes dados são retornados no formato JSON pelo serviço.

[Statement \(p. 777\)](#)

A instrução da permissão que é adicionada à política da função.

Tipo: sequência

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400
`PolicyLengthExceeded`
exception

A política de permissões do recurso é muito grande. [Saiba mais](#)

Código de status HTTP: 400
`PreconditionFailed`
exception

O `RevisionId` fornecido não corresponde ao `RevisionId` mais recente da função ou do alias do Lambda. Chame a API `GetFunction` ou `GetAlias` para recuperar o `RevisionId` mais recente para o seu recurso.

Código de status HTTP: 412
`ResourceConflict`
exception

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409
`ResourceNotFound`
exception

O recurso especificado na solicitação não existe.

Código de status HTTP: 404
`Service`
exception

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500
`TooManyRequests`
exception

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

CreateAlias

Cria um [alias](#) para uma versão da função do Lambda. Use alias para fornecer aos clientes um identificador de função que você pode atualizar para invocar uma versão diferente.

Você também pode mapear um alias para dividir solicitações de invocação entre duas versões. Use o parâmetro `RoutingConfig` para especificar uma segunda versão e a porcentagem de solicitações de invocação que ela recebe.

Sintaxe da solicitação

```
POST /2015-03-31/functions/FunctionName/aliases HTTP/1.1
Content-type: application/json

{
  "Description": "string",
  "FunctionVersion": "string",
  "Name": "string",
  "RoutingConfig": {
    "AdditionalVersionWeights": {
      "string" : number
    }
  }
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 779)

O nome da função Lambda.

Formatos de nome

- Nome da função - `MyFunction`.
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- ARN parcial - `123456789012:function:MyFunction`.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: `(arn:(aws[a-zA-Z-]*):lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Obrigatório: sim

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[Description](#) (p. 779)

Uma descrição do alias.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

Exigido: Não

[FunctionVersion \(p. 779\)](#)

A versão da função que o alias invoca.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Pattern: (\\$LATEST | [0-9]+)

Obrigatório: sim

[Name \(p. 779\)](#)

O nome do alias.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: (?![0-9]+\$)([a-zA-Z0-9-_]+)

Obrigatório: sim

[RoutingConfig \(p. 779\)](#)

A configuração de roteamento do alias.

Tipo: objeto [AliasRoutingConfiguration \(p. 995\)](#)

Exigido: Não

Sintaxe da resposta

```
HTTP/1.1 201
Content-type: application/json

{
    "AliasArn": "string",
    "Description": "string",
    "FunctionVersion": "string",
    "Name": "string",
    "RevisionId": "string",
    "RoutingConfig": {
        "AdditionalVersionWeights": {
            "string" : number
        }
    }
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço retornará uma resposta HTTP 201.

Os seguintes dados são retornados no formato JSON pelo serviço.

[AliasArn \(p. 780\)](#)

O nome do recurso da Amazon (ARN) do alarme do alias.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[Description \(p. 780\)](#)

Uma descrição do alias.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

[FunctionVersion \(p. 780\)](#)

A versão da função que o alias invoca.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Pattern: (\\$LATEST|[0-9]+)

[Name \(p. 780\)](#)

O nome do alias.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: (?![0-9]+\$)([a-zA-Z0-9-_]+)

[RevisionId \(p. 780\)](#)

Um identificador exclusivo que muda ao atualizar o alias.

Tipo: sequência

[RoutingConfig \(p. 780\)](#)

A [configuração de roteamento](#) do alias.

Tipo: objeto [AliasRoutingConfiguration \(p. 995\)](#)

Errors

[InvalidOperationException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ResourceConflictException](#)

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

[ResourceNotFoundException](#)

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

CreateCodeSigningConfig

Cria uma configuração de assinatura de código. Uma [configuração de assinatura de código](#) define uma lista de perfis de assinatura permitidos e define a política de validação da assinatura de código (ação a ser tomada se as verificações de validação da implantação falharem).

Sintaxe da solicitação

```
POST /2020-04-22/code-signing-configs/ HTTP/1.1
Content-type: application/json

{
    "AllowedPublishers": {
        "SigningProfileVersionArns": [ "string" ]
    },
    "CodeSigningPolicies": {
        "UntrustedArtifactOnDeployment": "string"
    },
    "Description": "string"
}
```

Parâmetros da solicitação de URI

A solicitação não usa nenhum parâmetro de URI.

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[AllowedPublishers](#) (p. 783)

Assinatura de perfis para esta configuração de assinatura de código.

Tipo: objeto [AllowedPublishers](#) (p. 996)

Obrigatório: sim

[CodeSigningPolicies](#) (p. 783)

As políticas de assinatura de código definem as ações a serem executadas se as verificações de validação falharem.

Tipo: objeto [CodeSigningPolicies](#) (p. 999)

Exigido: Não

[Description](#) (p. 783)

Nome descritivo para essa configuração de assinatura de código.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

Exigido: Não

Sintaxe da resposta

```
HTTP/1.1 201
```

```
Content-type: application/json

{
    "CodeSigningConfig": {
        "AllowedPublishers": {
            "SigningProfileVersionArns": [ "string" ]
        },
        "CodeSigningConfigArn": "string",
        "CodeSigningConfigId": "string",
        "CodeSigningPolicies": {
            "UntrustedArtifactOnDeployment": "string"
        },
        "Description": "string",
        "LastModified": "string"
    }
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço retornará uma resposta HTTP 201.

Os seguintes dados são retornados no formato JSON pelo serviço.

[CodeSigningConfig \(p. 783\)](#)

A configuração de assinatura de código.

Tipo: objeto [CodeSigningConfig \(p. 997\)](#)

Errors

InvalidOperationException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

CreateEventSourceMapping

Cria um mapeamento entre uma origem de eventos e uma função do AWS Lambda. O Lambda lê itens da origem de eventos e aciona a função.

Para obter detalhes sobre cada tipo de origem de evento, consulte os tópicos a seguir.

- [Configurar um stream do Dynamo DB como uma origem de eventos](#)
- [Configurar um stream do Kinesis como uma origem de eventos](#)
- [Configurar uma fila do Amazon SQS como uma fonte do evento](#)
- [Configurar um agente do MQ como uma origem de eventos](#)
- [Configurar o MSK como uma origem de eventos](#)
- [Configurar o Apache Kafka Autogerenciado como uma origem de eventos](#)

As seguintes opções de tratamento de erros estão disponíveis apenas para fontes de stream (DynamoDB e Kinesis):

- `BisectBatchOnFunctionError`: se a função retornar um erro, divida o lote em dois e tente novamente.
- `DestinationConfig`: envie registros descartados para uma fila do Amazon SQS ou para um tópico do Amazon SNS.
- `MaximumRecordAgeInSeconds`: descarta registros mais antigos que a idade especificada. O valor padrão é infinito (-1). Quando definido como infinito (-1), são feitas novas tentativas para os registros com falha até o registro expirar
- `MaximumRetryAttempts`: descarta registros após o número especificado de tentativas. O valor padrão é infinito (-1). Quando definido como infinito (-1), são feitas novas tentativas para os registros com falha até o registro expirar.
- `ParallelizationFactor`: processe vários lotes de cada fragmento simultaneamente.

Sintaxe da solicitação

```
POST /2015-03-31/event-source-mappings/ HTTP/1.1
Content-type: application/json

{
    "BatchSize": number,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "Enabled": boolean,
    "EventSourceArn": "string",
    "FunctionName": "string",
    "FunctionResponseTypes": [ "string" ],
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "Queues": [ "string" ],
    "SelfManagedEventSource": {
        "Endpoints": {
            "Region": "string",
            "Type": "string"
        }
    }
}
```

```
        "string" : [ "string" ]
    },
    "SourceAccessConfigurations": [
        {
            "Type": "string",
            "URI": "string"
        }
    ],
    "StartingPosition": "string",
    "StartingPositionTimestamp": number,
    "Topics": [ "string" ],
    "TumblingWindowInSeconds": number
}
```

Parâmetros da solicitação de URI

A solicitação não usa nenhum parâmetro de URI.

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[BatchSize \(p. 786\)](#)

O número máximo de registros em cada batch que o Lambda extrai da sua transmissão ou fila e envia para sua função. O Lambda transmite todos os registros no batch para a função em uma única chamada até o limite de carga útil para invocação síncrona (6 MB).

- Amazon Kinesis - padrão 100. No máximo 10.000.
- Amazon DynamoDB Streams - padrão 100. No máximo 1.000.
- Amazon Simple Queue Service - padrão 10. Para filas padrão, o máximo é 10.000. Para filas FIFO, o máximo é 10.
- Amazon Managed Streaming for Apache Kafka – padrão 100. No máximo 10.000.
- Apache Kafka autogerenciado - padrão 100. No máximo 10.000.

Type: inteiro

Faixa válida: valor mínimo de 1. Valor máximo de 10000.

Exigido: Não

[BisectBatchOnFunctionError \(p. 786\)](#)

(Somente streams) Se a função retornar um erro, divida o lote em dois e tente novamente.

Type: booliano

Exigido: Não

[DestinationConfig \(p. 786\)](#)

(Somente streams) Uma fila do Amazon SQS ou um destino de tópico do Amazon SNS para registros descartados.

Tipo: objeto [DestinationConfig \(p. 1002\)](#)

Exigido: Não

[Enabled \(p. 786\)](#)

Quando verdadeiro, o mapeamento da fonte do evento estará ativo. Quando falso, o Lambda pausará a sondagem e a invocação.

Padrão: True

Type: booliano

Exigido: Não

[EventSourceArn \(p. 786\)](#)

O nome de recurso da Amazon (ARN) da origem do evento.

- Amazon Kinesis - o ARN do stream de dados ou um consumidor de stream.
- Amazon DynamoDB Streams - o ARN do stream.
- Amazon Simple Queue Service - o ARN da fila.
- Amazon Managed Streaming for Apache Kafka – o ARN do cluster.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*)([a-zA-Z0-9\-.])+([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12})?:(.*)

Exigido: Não

[FunctionName \(p. 786\)](#)

O nome da função Lambda.

Formatos de nome

- Nome da função - MyFunction.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- ARN da versão ou alias - arn:aws:lambda:us-west-2:123456789012:function:MyFunction:PROD.
- ARN parcial - 123456789012:function:MyFunction.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[FunctionResponseTypes \(p. 786\)](#)

(Somente transmissões) Uma lista de enums de tipos de resposta atuais aplicados ao mapeamento de fontes de eventos.

Tipo: matriz de strings

Membros da matriz: número mínimo de 0 itens. Número máximo de 1 item.

Valores válidos: ReportBatchItemFailures

Exigido: Não

[MaximumBatchingWindowInSeconds \(p. 786\)](#)

(Transmissões e filas padrão do Amazon SQS) O tempo máximo usado pelo Lambda, em segundos, para reunir os registros antes de invocar a função.

Padrão: 0

Configuração relacionada: quando você define `BatchSize` como um valor maior que 10, deve definir `MaximumBatchingWindowInSeconds` como pelo menos 1.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 300.

Exigido: Não

[MaximumRecordAgeInSeconds \(p. 786\)](#)

(Somente streams) Descarta registros mais antigos que a idade especificada. O valor padrão é infinito (-1).

Type: inteiro

Intervalo válido: valor mínimo de -1. Valor máximo de 604800.

Exigido: Não

[MaximumRetryAttempts \(p. 786\)](#)

(Somente streams) Descarta registros após o número especificado de novas tentativas. O valor padrão é infinito (-1). Quando definido como infinito (-1), serão feitas novas tentativas para os registros com falha até o registro expirar.

Type: inteiro

Intervalo válido: valor mínimo de -1. Valor máximo de 10000.

Exigido: Não

[ParallelizationFactor \(p. 786\)](#)

(Somente streams) O número de lotes a serem processados de cada fragmento simultaneamente.

Type: inteiro

Faixa válida: valor mínimo de 1. Valor máximo de 10.

Exigido: Não

[Queues \(p. 786\)](#)

(MQ) O nome da fila de destino do agente do Amazon MQ a ser consumido.

Tipo: matriz de strings

Membros da matriz: número fixo de 1 item.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1000.

Pattern: [\s\S]*

Exigido: Não

[SelfManagedEventSource \(p. 786\)](#)

O cluster autogerenciado do Apache Kafka para enviar registros.

Tipo: objeto [SelfManagedEventSource \(p. 1036\)](#)

Exigido: Não

[SourceAccessConfigurations \(p. 786\)](#)

Uma matriz de protocolos de autenticação ou componentes da VPC necessária para proteger a origem do evento.

Tipo: matriz de [SourceAccessConfiguration \(p. 1037\)](#) objetos

Membros da matriz: número mínimo de 0 itens. Número máximo de 22 itens.

Exigido: Não

[StartingPosition \(p. 786\)](#)

A posição em um fluxo da qual você deseja iniciar a leitura. Obrigatório para fontes de fluxos do Amazon Kinesis, Amazon DynamoDB e Amazon MSK. AT_TIMESTAMP só é compatível com o Amazon Kinesis Streams.

Tipo: sequência

Valores válidos: TRIM_HORIZON | LATEST | AT_TIMESTAMP

Exigido: Não

[StartingPositionTimestamp \(p. 786\)](#)

Com StartingPosition definido como AT_TIMESTAMP, o tempo a partir do qual iniciar a leitura em segundos no horário do Unix.

Type: timestamp

Exigido: Não

[Topics \(p. 786\)](#)

O nome do tópico do Kafka.

Tipo: matriz de strings

Membros da matriz: número fixo de 1 item.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 249.

Pattern: ^[^ .]([a-zA-Z0-9\-_\.]+)

Exigido: Não

[TumblingWindowInSeconds \(p. 786\)](#)

(Somente streams) A duração, em segundos, de uma janela de processamento. O intervalo é entre 1 segundo até 900 minutos.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 900.

Exigido: Não

Sintaxe da resposta

```
HTTP/1.1 202
Content-type: application/json
```

```
{  
    "BatchSize": number,  
    "BisectBatchOnFunctionError": boolean,  
    "DestinationConfig": {  
        "OnFailure": {  
            "Destination": "string"  
        },  
        "OnSuccess": {  
            "Destination": "string"  
        }  
    },  
    "EventSourceArn": "string",  
    "FunctionArn": "string",  
    "FunctionResponseTypes": [ "string" ],  
    "LastModified": number,  
    "LastProcessingResult": "string",  
    "MaximumBatchingWindowInSeconds": number,  
    "MaximumRecordAgeInSeconds": number,  
    "MaximumRetryAttempts": number,  
    "ParallelizationFactor": number,  
    "Queues": [ "string" ],  
    "SelfManagedEventSource": {  
        "Endpoints": {  
            "string": [ "string" ]  
        }  
    },  
    "SourceAccessConfigurations": [  
        {  
            "Type": "string",  
            "URI": "string"  
        }  
    ],  
    "StartingPosition": "string",  
    "StartingPositionTimestamp": number,  
    "State": "string",  
    "StateTransitionReason": "string",  
    "Topics": [ "string" ],  
    "TumblingWindowInSeconds": number,  
    "UUID": "string"  
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço retornará uma resposta HTTP 202.

Os seguintes dados são retornados no formato JSON pelo serviço.

BatchSize (p. 790)

O número máximo de registros em cada batch que o Lambda extrai da sua transmissão ou fila e envia para sua função. O Lambda transmite todos os registros no batch para a função em uma única chamada até o limite de carga útil para invocação síncrona (6 MB).

Valor padrão: varia de acordo com o serviço. Para o Amazon SQS, o padrão é 10. Para todos os outros serviços, o padrão é 100.

Configuração relacionada: quando você define `BatchSize` como um valor maior que 10, deve definir `MaximumBatchingWindowInSeconds` como pelo menos 1.

Type: inteiro

Faixa válida: valor mínimo de 1. Valor máximo de 10000.

[BisectBatchOnFunctionError \(p. 790\)](#)

(Somente streams) Se a função retornar um erro, divida o lote em dois e tente novamente. O valor padrão é falso.

Tipo: booliano

[DestinationConfig \(p. 790\)](#)

(Somente streams) Uma fila do Amazon SQS ou um destino de tópico do Amazon SNS para registros descartados.

Tipo: objeto [DestinationConfig \(p. 1002\)](#)

[EventSourceArn \(p. 790\)](#)

O nome de recurso da Amazon (ARN) da origem do evento.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*)([a-zA-Z0-9\-])+([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12})?:(.*?)

[FunctionArn \(p. 790\)](#)

O ARN da função Lambda.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[FunctionResponseTypes \(p. 790\)](#)

(Somente transmissões) Uma lista de enums de tipos de resposta atuais aplicados ao mapeamento de fontes de eventos.

Tipo: matriz de strings

Membros da matriz: número mínimo de 0 itens. Número máximo de 1 item.

Valores válidos: [ReportBatchItemFailures](#)

[LastModified \(p. 790\)](#)

A data em que o mapeamento de fontes de eventos foi atualizado pela última vez ou seu estado mudou, em segundos no horário do Unix.

Type: timestamp

[LastProcessingResult \(p. 790\)](#)

O resultado da última invocação do Lambda da sua função.

Tipo: sequência

[MaximumBatchingWindowInSeconds \(p. 790\)](#)

(Transmissões e filas padrão do Amazon SQS) O tempo máximo usado pelo Lambda, em segundos, para reunir os registros antes de invocar a função.

Padrão: 0

Configuração relacionada: quando você define `BatchSize` como um valor maior que 10, deve definir `MaximumBatchingWindowInSeconds` como pelo menos 1.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 300.

[MaximumRecordAgeInSeconds \(p. 790\)](#)

(Somente streams) Descarta registros mais antigos que a idade especificada. O valor padrão é -1, o que define a idade máxima como infinito. Quando o valor é definido como infinito, o Lambda nunca descarta registros antigos.

Type: inteiro

Intervalo válido: valor mínimo de -1. Valor máximo de 604800.

[MaximumRetryAttempts \(p. 790\)](#)

(Somente streams) Descarta registros após o número especificado de novas tentativas. O valor padrão é -1, o que define o número máximo de tentativas como infinito. Quando MaximumRetryAttempts é infinito, o Lambda tenta executar novamente os registros com falha até que o registro expire na fonte de eventos.

Type: inteiro

Intervalo válido: valor mínimo de -1. Valor máximo de 10000.

[ParallelizationFactor \(p. 790\)](#)

(Somente transmissões) O número de lotes a serem processados de cada fragmento simultaneamente. O valor padrão é 1.

Type: inteiro

Faixa válida: valor mínimo de 1. Valor máximo de 10.

[Queues \(p. 790\)](#)

(Amazon MQ) O nome da fila de destino do agente do Amazon MQ a ser consumido.

Tipo: matriz de strings

Membros da matriz: número fixo de 1 item.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1000.

Pattern: [\s\S]*

[SelfManagedEventSource \(p. 790\)](#)

O cluster autogerenciado do Apache Kafka para sua fonte de eventos.

Tipo: objeto [SelfManagedEventSource \(p. 1036\)](#)

[SourceAccessConfigurations \(p. 790\)](#)

Uma matriz do protocolo de autenticação, os componentes da VPC ou o host virtual para proteger e definir a fonte de eventos.

Tipo: matriz de [SourceAccessConfiguration \(p. 1037\)](#) objetos

Membros da matriz: número mínimo de 0 itens. Número máximo de 22 itens.

[StartingPosition \(p. 790\)](#)

A posição em um fluxo da qual você deseja iniciar a leitura. Obrigatório para origens de fluxo do Amazon Kinesis, Amazon DynamoDB e Amazon MSK. AT_TIMESTAMP só é compatível com o Amazon Kinesis Streams.

Tipo: sequência

Valores válidos: TRIM_HORIZON | LATEST | AT_TIMESTAMP

[StartingPositionTimestamp \(p. 790\)](#)

Com StartingPosition definido como AT_TIMESTAMP, o tempo a partir do qual iniciar a leitura em segundos no horário do Unix.

Type: timestamp

[State \(p. 790\)](#)

O estado do mapeamento da fonte de eventos. Pode ser um destes: Creating, Enabling, Enabled, Disabling, Disabled, Updating ou Deleting.

Tipo: sequência

[StateTransitionReason \(p. 790\)](#)

Indica se um usuário ou o Lambda fez a última alteração no mapeamento de fontes de eventos.

Tipo: sequência

[Topics \(p. 790\)](#)

O nome do tópico do Kafka.

Tipo: matriz de strings

Membros da matriz: número fixo de 1 item.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 249.

Pattern: ^[^.]([a-zA-Z0-9\-.]+)

[TumblingWindowInSeconds \(p. 790\)](#)

(Somente transmissões) A duração, em segundos, de uma janela de processamento. O intervalo é 1 a 900 segundos.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 900.

[UUID \(p. 790\)](#)

O identificador do mapeamento de fontes de eventos.

Tipo: sequência

Errors

InvalidArgumentException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceConflictException

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

CreateFunction

Cria uma função do Lambda. Para criar uma função, você precisa de um [pacote de implantação](#) e de uma [função de execução](#). O pacote de implantação é um arquivamento de arquivo.zip ou uma imagem de contêiner que contém seu código de função. A função de execução concede à função permissão para usar os serviços da AWS, como o Amazon CloudWatch Logs para streaming de logs e o X-Ray para rastreamento de solicitações.

Você define o tipo de pacote como `Image` se o pacote de implantação for uma [imagem de contêiner](#). Para uma imagem de contêiner, a propriedade `code` deve incluir o URI de uma imagem de contêiner no registro do Amazon ECR. Você não precisa especificar as propriedades do manipulador e do tempo de execução.

Você define o tipo de pacote como `Zip` se o pacote de implantação for um arquivamento de [arquivo.zip](#). Para um arquivamento de arquivo.zip, a propriedade `code` especifica a localização do arquivo.zip. Você também deve especificar as propriedades do manipulador e do tempo de execução.

Quando você cria uma função, o Lambda provisiona uma instância da função e seus recursos de suporte. Se sua função se conecta a uma VPC, esse processo pode demorar mais ou menos um minuto. Durante esse período, não será possível invocar ou modificar a função. Os campos `State`, `StateReason` e `StateReasonCode` na resposta de [GetFunctionConfiguration](#) (p. 851) indicam quando a função está pronta para ser invocada. Para obter mais informações, consulte [Estados da função](#).

Uma função tem uma versão não publicada e pode ter versões e aliases publicados. A versão não publicada muda quando você atualiza o código e a configuração da função. Uma versão publicada é um snapshot do código e da configuração da função que não pode ser alterado. Um alias é um recurso nomeado que mapeia para uma versão e pode ser alterado para mapear para uma versão diferente. Use o parâmetro `Publish` para criar a versão 1 de sua função com base em sua configuração inicial.

Os outros parâmetros permitem que você defina configurações específicas da versão e em nível de função. Você pode modificar configurações específicas da versão posteriormente com [UpdateFunctionConfiguration](#) (p. 974). As configurações em nível de função se aplicam às versões não publicadas e publicadas da função e incluem etiquetas ([TagResource](#) (p. 945)) e limites de simultaneidade por função ([PutFunctionConcurrency](#) (p. 930)).

Você pode usar a assinatura de código se o pacote de implantação for um arquivamento do arquivo .zip. Para ativar a assinatura de código para essa função, especifique o ARN de uma configuração de assinatura de código. Quando um usuário tenta implantar um pacote de código com [UpdateFunctionCode](#) (p. 965), o Lambda verifica se o pacote de código tem uma assinatura válida de um fornecedor confiável. A configuração de assinatura de código inclui um conjunto de perfis de assinatura que define os editores confiáveis para essa função.

Se outra conta ou um serviço da AWS invoca sua função, use [AddPermission](#) (p. 775) para conceder permissão criando uma política do IAM baseada em recursos. Você pode conceder permissões em nível de função, em uma versão ou em um alias.

Para invocar sua função diretamente, use [Invoke](#) (p. 875). Para invocar sua função em resposta a eventos em outros serviços da AWS, crie um mapeamento da origem do evento ([CreateEventSourceMapping](#) (p. 786)) ou configure um acionador de função no outro serviço. Para obter mais informações, consulte [Invocar funções](#).

Sintaxe da solicitação

```
POST /2015-03-31/functions HTTP/1.1
Content-type: application/json

{
  "Code": {
    "ImageUri": "string",
```

```
        "S3Bucket": "string",
        "S3Key": "string",
        "S3ObjectVersion": "string",
        "ZipFile": blob
    },
    "CodeSigningConfigArn": "string",
    "DeadLetterConfig": {
        "TargetArn": "string"
    },
    "Description": "string",
    "Environment": {
        "Variables": {
            "string" : "string"
        }
    },
    "FileSystemConfigs": [
        {
            "Arn": "string",
            "LocalMountPath": "string"
        }
    ],
    "FunctionName": "string",
    "Handler": "string",
    "ImageConfig": {
        "Command": [ "string" ],
        "EntryPoint": [ "string" ],
        "WorkingDirectory": "string"
    },
    "KMSKeyArn": "string",
    "Layers": [ "string" ],
    "MemorySize": number,
    "PackageType": "string",
    "Publish": boolean,
    "Role": "string",
    "Runtime": "string",
    "Tags": {
        "string" : "string"
    },
    "Timeout": number,
    "TracingConfig": {
        "Mode": "string"
    },
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
        "SubnetIds": [ "string" ]
    }
}
```

Parâmetros da solicitação de URI

A solicitação não usa nenhum parâmetro de URI.

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[Code \(p. 796\)](#)

O código da função.

Tipo: objeto [FunctionCode \(p. 1012\)](#)

Obrigatório: sim

[CodeSigningConfigArn \(p. 796\)](#)

Para ativar a assinatura de código para essa função, especifique o ARN de uma configuração de assinatura de código. Uma configuração de assinatura de código inclui um conjunto de perfis de assinatura que define os editores confiáveis para essa função.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 200.

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}((-gov)|(-iso(b?)))?-+[a-z]+\d{1}:\d{12}:code-signing-config:csc-[a-zA-Z0-9]{17}`

Exigido: Não

[DeadLetterConfig \(p. 796\)](#)

Uma configuração de dead letter queue que especifica a fila ou tópico em que o Lambda envia eventos assíncronos quando eles falham no processamento. Para obter mais informações, consulte [Filas de mensagens mortas](#).

Tipo: objeto [DeadLetterConfig \(p. 1001\)](#)

Exigido: Não

[Description \(p. 796\)](#)

Uma descrição da função.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

Exigido: Não

[Environment \(p. 796\)](#)

As variáveis de ambiente que são acessíveis pelo código de função durante a execução.

Tipo: objeto [Environment \(p. 1003\)](#)

Exigido: Não

[FileSystemConfigs \(p. 796\)](#)

Configurações de conexão para um sistema de arquivos do Amazon EFS.

Tipo: matriz de [FileSystemConfig \(p. 1011\)](#) objetos

Membros da matriz: número máximo de 1 item.

Exigido: Não

[FunctionName \(p. 796\)](#)

O nome da função Lambda.

Formatos de nome

- Nome da função - `my-function`.
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- ARN parcial - `123456789012:function:my-function`.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Obrigatório: sim

[Handler \(p. 796\)](#)

O nome do método em seu código que o Lambda chama para executar a função. O formato inclui o nome do arquivo. Ele também pode incluir namespaces e outros qualificadores, dependendo do tempo de execução. Para obter mais informações, consulte [Modelo de programação](#).

Tipo: sequência

Restrições de tamanho: tamanho máximo de 128.

Pattern: `[^\s]+`

Exigido: Não

[ImageConfig \(p. 796\)](#)

[Valores de configuração](#) da imagem do contêiner que substituem os valores do Dockerfile da imagem do contêiner.

Tipo: objeto [ImageConfig \(p. 1023\)](#)

Exigido: Não

[KMSKeyArn \(p. 796\)](#)

O ARN da chave do AWS Key Management Service (AWS KMS) usada para criptografar variáveis de ambiente de sua função. Se não for fornecido, o AWS Lambda usa uma chave de serviço padrão.

Tipo: sequência

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:[.]*)|()`

Exigido: Não

[Layers \(p. 796\)](#)

Uma lista de [camadas de função](#) para adicionar ao ambiente de execução da função. Especifique cada camada por seu ARN, incluindo a versão.

Tipo: matriz de strings

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: `arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+`

Exigido: Não

[MemorySize \(p. 796\)](#)

A quantidade de [memória disponível para a função](#) no tempo de execução. Aumentar a memória de função também aumenta sua alocação de CPU. O valor padrão é 128 MB. O valor pode ser qualquer múltiplo de 1 MB.

Type: inteiro

Intervalo válido: valor mínimo de 128. Valor máximo de 10240.

Exigido: Não

[PackageType \(p. 796\)](#)

O tipo de pacote de implantação. Defina como `Image` para imagem de contêiner e defina `Zip` para arquivo ZIP.

Tipo: sequência

Valores válidos: `Zip` | `Image`

Exigido: Não

[Publish \(p. 796\)](#)

Defina como “true” para publicar a primeira versão da função durante a criação.

Type: booleano

Exigido: Não

[Role \(p. 796\)](#)

O nome de recurso da Amazon (ARN) da função de execução da função.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\\-_/+]`

Obrigatório: sim

[Runtime \(p. 796\)](#)

O identificador do tempo de execução da função.

Tipo: sequência

Valores válidos: `nodejs` | `nodejs4.3` | `nodejs6.10` | `nodejs8.10` | `nodejs10.x` | `nodejs12.x` | `nodejs14.x` | `java8` | `java8.al2` | `java11` | `python2.7` | `python3.6` | `python3.7` | `python3.8` | `python3.9` | `dotnetcore1.0` | `dotnetcore2.0` | `dotnetcore2.1` | `dotnetcore3.1` | `nodejs4.3-edge` | `go1.x` | `ruby2.5` | `ruby2.7` | `provided` | `provided.al2`

Exigido: Não

[Tags \(p. 796\)](#)

Uma lista de `tags` para aplicar à função.

Tipo: mapa de string para string

Exigido: Não

[Timeout \(p. 796\)](#)

O valor do tempo que o Lambda permite que uma função seja executada antes de encerrá-la. O padrão é 3 segundos. O valor máximo permitido é de 900 segundos. Para obter informações adicionais, consulte [Ambiente de execução do Lambda](#).

Type: inteiro

Faixa válida: valor mínimo de 1.

Exigido: Não

TracingConfig (p. 796)

Defina Mode como Active para criar uma amostra e rastrear um subconjunto de solicitações de entrada com o [x-Ray](#).

Tipo: objeto [TracingConfig](#) (p. 1039)

Exigido: Não

VpcConfig (p. 796)

Para a conectividade de rede para os recursos da AWS em uma VPC, especifique uma lista de grupos de segurança e sub-redes na VPC. Quando você conecta uma função a uma VPC, ela só pode acessar recursos e a Internet por meio dessa VPC. Para obter mais informações, consulte [Configurações da VPC](#).

Tipo: objeto [VpcConfig](#) (p. 1041)

Exigido: Não

Sintaxe da resposta

```
HTTP/1.1 201
Content-type: application/json

{
    "CodeSha256": "string",
    "CodeSize": number,
    "DeadLetterConfig": {
        "TargetArn": "string"
    },
    "Description": "string",
    "Environment": {
        "Error": {
            "ErrorCode": "string",
            "Message": "string"
        },
        "Variables": {
            "string" : "string"
        }
    },
    "FileSystemConfigs": [
        {
            "Arn": "string",
            "LocalMountPath": "string"
        }
    ],
    "FunctionArn": "string",
    "FunctionName": "string",
    "Handler": "string",
    "ImageConfigResponse": {
        "Error": {
            "ErrorCode": "string",
            "Message": "string"
        },
        "ImageConfig": {
            "Command": [ "string" ],
            "EntryPoint": [ "string" ],
            "WorkingDirectory": "string"
        }
    },
    "KMSKeyArn": "string",
    "LastModified": "string",
    "LastUpdateStatus": "string",
    "MemorySize": number,
    "Role": "string",
    "Runtime": "string",
    "Timeout": number
}
```

```
"LastUpdateStatus": "string",
"LastUpdateStatusReason": "string",
"LastUpdateStatusReasonCode": "string",
"Layers": [
    {
        "Arn": "string",
        "CodeSize": number,
        "SigningJobArn": "string",
        "SigningProfileVersionArn": "string"
    }
],
"MasterArn": "string",
"MemorySize": number,
"PackageType": "string",
"RevisionId": "string",
"Role": "string",
"Runtime": "string",
"SigningJobArn": "string",
"SigningProfileVersionArn": "string",
"State": "string",
"StateReason": "string",
"StateReasonCode": "string",
"Timeout": number,
"TracingConfig": {
    "Mode": "string"
},
"Version": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço retornará uma resposta HTTP 201.

Os seguintes dados são retornados no formato JSON pelo serviço.

[CodeSha256 \(p. 801\)](#)

O hash SHA256 do pacote de implantação da função.

Tipo: sequência

[CodeSize \(p. 801\)](#)

O tamanho do pacote de implantação da função em bytes.

Type: longo

[DeadLetterConfig \(p. 801\)](#)

A fila de mensagens mortas da função.

Tipo: objeto [DeadLetterConfig \(p. 1001\)](#)

[Description \(p. 801\)](#)

A descrição da função.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

[Environment](#) (p. 801)

As [variáveis de ambiente](#) da função.

Tipo: objeto [EnvironmentResponse](#) (p. 1005)

[FileSystemConfigs](#) (p. 801)

Configurações de conexão para um [sistema de arquivos do Amazon EFS](#).

Tipo: matriz de [FileSystemConfig](#) (p. 1011)objetos

Membros da matriz: número máximo de 1 item.

[FunctionArn](#) (p. 801)

O nome do recurso da Amazon (ARN) da função.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[FunctionName](#) (p. 801)

Nome da função.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 170.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Handler](#) (p. 801)

A função que o Lambda chama para começar a executar a função.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 128.

Pattern: `[^\s]+`

[ImageConfigResponse](#) (p. 801)

Os valores de configuração da imagem da função.

Tipo: objeto [ImageConfigResponse](#) (p. 1025)

[KMSKeyArn](#) (p. 801)

A chave KMS usada para criptografar as variáveis do ambiente da função. Esta chave é retornada somente se você tiver configurado uma CMK gerenciada pelo cliente.

Tipo: sequência

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-_\.]+:*)|()`

[LastModified](#) (p. 801)

A data e a hora em que a função foi atualizada, no [formato ISO-8601](#) (AAAA-MM-DDThh:mm:ss.sTZD).

Tipo: sequência

[LastUpdateStatus \(p. 801\)](#)

O status da última atualização que foi executada na função. Ele é definido pela primeira vez como `Successful` após a conclusão da criação da função.

Tipo: sequência

Valores válidos: `Successful` | `Failed` | `InProgress`

[LastUpdateStatusReason \(p. 801\)](#)

O motivo pelo qual foi realizada a última atualização na função.

Tipo: sequência

[LastUpdateStatusReasonCode \(p. 801\)](#)

O código do motivo pelo qual foi realizada a última atualização na função.

Tipo: sequência

Valores válidos: `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup` | `ImageDeleted` | `ImageAccessDenied` | `InvalidImage`

[Layers \(p. 801\)](#)

As [camadas](#) da função.

Tipo: matriz de [Layer \(p. 1026\)](#) objetos

[MasterArn \(p. 801\)](#)

Para funções do Lambda@Edge, o ARN da função mestre.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\${LATEST}|[a-zA-Z0-9-_]+))?`

[MemorySize \(p. 801\)](#)

A quantidade de memória disponível para a função no tempo de execução.

Type: inteiro

Intervalo válido: valor mínimo de 128. Valor máximo de 10240.

[PackageType \(p. 801\)](#)

O tipo de pacote de implantação. Defina como `Image` para imagem de contêiner e defina `Zip` para arquivo de documento `.zip`.

Tipo: sequência

Valores válidos: `zip` | `Image`

[RevisionId \(p. 801\)](#)

A última revisão atualizada da função ou do alias.

Tipo: sequência

[Role \(p. 801\)](#)

A função de execução da função.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@/-/_]+
[Runtime \(p. 801\)](#)

O ambiente de execução da função do Lambda.

Tipo: sequência

Valores válidos: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | nodejs10.x | nodejs12.x | nodejs14.x | java8 | java8.al2 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | python3.9 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | dotnetcore3.1 | nodejs4.3-edge | go1.x | ruby2.5 | ruby2.7 | provided | provided.al2

[SigningJobArn \(p. 801\)](#)

O ARN do trabalho de assinatura.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-\-])+:(a-z){2}(-gov)?-[a-z]+\-\d{1}):\:(\d{12}):\:(.*)

[SigningProfileVersionArn \(p. 801\)](#)

O ARN da versão do perfil de assinatura.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-\-])+:(a-z){2}(-gov)?-[a-z]+\-\d{1}):\:(\d{12}):\:(.*)

[State \(p. 801\)](#)

O estado atual da função. Quando o estado é `Inactive`, você pode invocar a função para reativá-la.

Tipo: sequência

Valores válidos: Pending | Active | Inactive | Failed

[StateReason \(p. 801\)](#)

O motivo para o estado atual da função.

Tipo: sequência

[StateReasonCode \(p. 801\)](#)

O código do motivo para o estado atual da função. Quando o código for `Creating`, não será possível invocar ou modificar a função.

Tipo: sequência

Valores válidos: Idle | Creating | Restoring | EniLimitExceeded | InsufficientRolePermissions | InvalidConfiguration | InternalError | SubnetOutOfIPAddresses | InvalidSubnet | InvalidSecurityGroup | ImageDeleted | ImageAccessDenied | InvalidImage

[Timeout \(p. 801\)](#)

A quantidade de tempo, em segundos, que o Lambda permite que uma função seja executada antes de encerrá-la.

Type: inteiro

Faixa válida: valor mínimo de 1.

[TracingConfig \(p. 801\)](#)

A configuração de rastreamento de AWS X-Ray da função.

Tipo: objeto [TracingConfigResponse \(p. 1040\)](#)

[Version \(p. 801\)](#)

A versão da função do Lambda.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 801\)](#)

A configuração de rede da função.

Tipo: objeto [VpcConfigResponse \(p. 1042\)](#)

Errors

[CodeSigningConfigNotFoundException](#)

A configuração de assinatura de código especificada não existe.

Código de status HTTP: 404

[CodeStorageExceedededException](#)

Você excedeu seu tamanho máximo total de código por conta. [Saiba mais](#)

Código de status HTTP: 400

[CodeVerificationFailedException](#)

A assinatura de código falhou em uma ou mais verificações de validação em relação à incompatibilidade ou expiração de assinatura, e a política de assinatura de código está definida como ENFORCE. O Lambda bloqueia a implantação.

Código de status HTTP: 400

[InvalidCodeSignatureException](#)

A assinatura de código falhou na verificação de integridade. O Lambda sempre bloqueia a implantação se a verificação de integridade falhar, mesmo que a diretiva de assinatura de código esteja definida como WARN.

Código de status HTTP: 400

[InvalidParameterValueException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ResourceConflictException](#)

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteAlias

Exclui um [alias](#) de função do Lambda.

Sintaxe da solicitação

```
DELETE /2015-03-31/functions/FunctionName/aliases/Name HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 808)

O nome da função Lambda.

Formatos de nome

- Nome da função - MyFunction.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- ARN parcial - 123456789012:function:MyFunction.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[Name](#) (p. 808)

O nome do alias.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: (?![0-9]+\\$)([a-zA-Z0-9-_]+)

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 204
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 204 com um corpo HTTP vazio.

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceConflictException

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteCodeSigningConfig

Exclui a configuração de assinatura de código. Uma configuração de assinatura de código pode ser excluída somente se nenhuma função estiver usando-a.

Sintaxe da solicitação

```
DELETE /2020-04-22/code-signing-configs/CodeSigningConfigArn HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

CodeSigningConfigArn (p. 810)

O nome do recurso da Amazon (ARN) da configuração de assinatura de código.

Restrições de tamanho: tamanho máximo de 200.

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}((-gov)|(-iso(b?)))?- [a-z]+-\d{1}:\d{12}:code-signing-config:csc-[a-z0-9]{17}

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 204
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 204 com um corpo HTTP vazio.

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceConflictException

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteEventSourceMapping

Exclui um [mapeamento de origem do evento](#). Você pode obter o identificador de um mapeamento a partir da saída de [ListEventSourceMappings](#) (p. 888).

Quando você exclui um mapeamento de origem do evento, ele entra no estado `Deleting` e pode não ser completamente excluído por vários segundos.

Sintaxe da solicitação

```
DELETE /2015-03-31/event-source-mappings/UUID HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

UUID (p. 812)

O identificador do mapeamento da origem do evento.

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 202
Content-type: application/json

{
    "BatchSize": number,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "FunctionResponseTypes": [ "string" ],
    "LastModified": number,
    "LastProcessingResult": "string",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "Queues": [ "string" ],
    "SelfManagedEventSource": {
        "Endpoints": {
            "string": [ "string" ]
        }
    },
    "SourceAccessConfigurations": [
```

```
{  
    "Type": "string",  
    "URI": "string"  
}  
],  
"StartingPosition": "string",  
"StartingPositionTimestamp": number,  
"State": "string",  
"StateTransitionReason": "string",  
"Topics": [ "string" ],  
"TumblingWindowInSeconds": number,  
"UUID": "string"  
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço retornará uma resposta HTTP 202.

Os seguintes dados são retornados no formato JSON pelo serviço.

[BatchSize \(p. 812\)](#)

O número máximo de registros em cada batch que o Lambda extrai da sua transmissão ou fila e envia para sua função. O Lambda transmite todos os registros no batch para a função em uma única chamada até o limite de carga útil para invocação síncrona (6 MB).

Valor padrão: varia de acordo com o serviço. Para o Amazon SQS, o padrão é 10. Para todos os outros serviços, o padrão é 100.

Configuração relacionada: quando você define `BatchSize` como um valor maior que 10, deve definir `MaximumBatchingWindowInSeconds` como pelo menos 1.

Type: inteiro

Faixa válida: valor mínimo de 1. Valor máximo de 10000.

[BisectBatchOnFunctionError \(p. 812\)](#)

(Somente streams) Se a função retornar um erro, divida o lote em dois e tente novamente. O valor padrão é falso.

Type: booleano

[DestinationConfig \(p. 812\)](#)

(Somente streams) Uma fila do Amazon SQS ou um destino de tópico do Amazon SNS para registros descartados.

Tipo: objeto [DestinationConfig \(p. 1002\)](#)

[EventSourceArn \(p. 812\)](#)

O nome de recurso da Amazon (ARN) da origem do evento.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+:(\w{2})(-\w{3})?-(\w{2})+-\d{1}):\d{12}:(\w*)`

[FunctionArn \(p. 812\)](#)

O ARN da função Lambda.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?
[FunctionResponseTypes](#) (p. 812)

(Somente transmissões) Uma lista de enums de tipos de resposta atuais aplicados ao mapeamento de fontes de eventos.

Tipo: matriz de strings

Membros da matriz: número mínimo de 0 itens. Número máximo de 1 item.

Valores válidos: [ReportBatchItemFailures](#)

[LastModified](#) (p. 812)

A data em que o mapeamento de fontes de eventos foi atualizado pela última vez ou seu estado mudou, em segundos no horário do Unix.

Type: timestamp

[LastProcessingResult](#) (p. 812)

O resultado da última invocação do Lambda da sua função.

Tipo: sequência

[MaximumBatchingWindowInSeconds](#) (p. 812)

(Transmissões e filas padrão do Amazon SQS) O tempo máximo usado pelo Lambda, em segundos, para reunir os registros antes de invocar a função.

Padrão: 0

Configuração relacionada: quando você define `BatchSize` como um valor maior que 10, deve definir `MaximumBatchingWindowInSeconds` como pelo menos 1.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 300.

[MaximumRecordAgeInSeconds](#) (p. 812)

(Somente streams) Descarta registros mais antigos que a idade especificada. O valor padrão é -1, o que define a idade máxima como infinito. Quando o valor é definido como infinito, o Lambda nunca descarta registros antigos.

Type: inteiro

Intervalo válido: valor mínimo de -1. Valor máximo de 604800.

[MaximumRetryAttempts](#) (p. 812)

(Somente streams) Descarta registros após o número especificado de novas tentativas. O valor padrão é -1, o que define o número máximo de tentativas como infinito. Quando `MaximumRetryAttempts` é infinito, o Lambda tenta executar novamente os registros com falha até que o registro expire na fonte de eventos.

Type: inteiro

Intervalo válido: valor mínimo de -1. Valor máximo de 10000.

[ParallelizationFactor](#) (p. 812)

(Somente transmissões) O número de lotes a serem processados de cada fragmento simultaneamente. O valor padrão é 1.

Type: inteiro

Faixa válida: valor mínimo de 1. Valor máximo de 10.

[Queues \(p. 812\)](#)

(Amazon MQ) O nome da fila de destino do agente do Amazon MQ a ser consumido.

Tipo: matriz de strings

Membros da matriz: número fixo de 1 item.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1000.

Pattern: [\s\S]*

[SelfManagedEventSource \(p. 812\)](#)

O cluster autogerenciado do Apache Kafka para sua fonte de eventos.

Tipo: objeto [SelfManagedEventSource \(p. 1036\)](#)

[SourceAccessConfigurations \(p. 812\)](#)

Uma matriz do protocolo de autenticação, os componentes da VPC ou o host virtual para proteger e definir a fonte de eventos.

Tipo: matriz de [SourceAccessConfiguration \(p. 1037\)](#) objetos

Membros da matriz: número mínimo de 0 itens. Número máximo de 22 itens.

[StartingPosition \(p. 812\)](#)

A posição em um fluxo da qual você deseja iniciar a leitura. Obrigatório para origens de fluxo do Amazon Kinesis, Amazon DynamoDB e Amazon MSK. AT_TIMESTAMP só é compatível com o Amazon Kinesis Streams.

Tipo: sequência

Valores válidos: TRIM_HORIZON | LATEST | AT_TIMESTAMP

[StartingPositionTimestamp \(p. 812\)](#)

Com StartingPosition definido como AT_TIMESTAMP, o tempo a partir do qual iniciar a leitura em segundos no horário do Unix.

Type: timestamp

[State \(p. 812\)](#)

O estado do mapeamento da fonte de eventos. Pode ser um destes: Creating, Enabling, Enabled, Disabling, Disabled, Updating ou Deleting.

Tipo: sequência

[StateTransitionReason \(p. 812\)](#)

Indica se um usuário ou o Lambda fez a última alteração no mapeamento de fontes de eventos.

Tipo: sequência

[Topics \(p. 812\)](#)

O nome do tópico do Kafka.

Tipo: matriz de strings

Membros da matriz: número fixo de 1 item.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 249.

Pattern: ^[^ .]([a-zA-Z0-9\-_\.]+)

[TumblingWindowInSeconds \(p. 812\)](#)

(Somente transmissões) A duração, em segundos, de uma janela de processamento. O intervalo é 1 a 900 segundos.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 900.

[UUID \(p. 812\)](#)

O identificador do mapeamento de fontes de eventos.

Tipo: sequência

Errors

[InvalidParameterValueException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ResourceInUseException](#)

A operação entra em conflito com a disponibilidade do recurso. Por exemplo, você tentou atualizar um mapeamento EventSource no estado “CREATING” ou tentou excluir um mapeamento EventSource atualmente no estado “UPDATING”.

Código de status HTTP: 400

[ResourceNotFoundException](#)

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

[TooManyRequestsException](#)

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)

- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteFunction

Exclui uma função do Lambda. Para excluir uma versão de função específica, use o parâmetro `Qualifier`. Caso contrário, todas as versões e os aliases serão excluídos.

Para excluir mapeamentos de origem de evento do Lambda que invocam uma função, use [DeleteEventSourceMapping \(p. 812\)](#). Para serviços e recursos da AWS que invocam sua função diretamente, exclua o acionador no serviço em que você o configurou originalmente.

Sintaxe da solicitação

```
DELETE /2015-03-31/functions/FunctionName?Qualifier=Qualifier HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName \(p. 818\)](#)

O nome da função ou versão do Lambda.

Formatos de nome

- Nome da função: `my-function` (somente nome), `my-function:1` (com a versão).
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- ARN parcial - `123456789012:function:my-function`.

Você pode anexar um número de versão ou alias a qualquer um dos formatos. A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Obrigatório: sim

[Qualifier \(p. 818\)](#)

Especifique a versão a ser excluída. Não é possível excluir uma versão referenciada por um alias.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: `(|[a-zA-Z0-9$-_]+)`

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 204
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 204 com um corpo HTTP vazio.

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceConflictException

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteFunctionCodeSigningConfig

Remove a configuração de assinatura de código da função.

Sintaxe da solicitação

```
DELETE /2020-06-30/functions/FunctionName/code-signing-config HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

FunctionName (p. 820)

O nome da função Lambda.

Formatos de nome

- Nome da função - MyFunction.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- ARN parcial - 123456789012:function:MyFunction.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+\-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 204
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 204 com um corpo HTTP vazio.

Errors

CodeSigningConfigNotFoundException

A configuração de assinatura de código especificada não existe.

Código de status HTTP: 404

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

- Código de status HTTP: 400
`ResourceConflictException`
O recurso já existe ou outra operação está em andamento.
- Código de status HTTP: 409
`ResourceNotFoundException`
O recurso especificado na solicitação não existe.
- Código de status HTTP: 404
`ServiceException`
O serviço AWS Lambda encontrou um erro interno.
- Código de status HTTP: 500
`TooManyRequestsException`
O limite de taxa de transferência da solicitação foi excedido.
- Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteFunctionConcurrency

Remove um limite de execução simultânea de uma função.

Sintaxe da solicitação

```
DELETE /2017-10-31/functions/FunctionName/concurrency HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

FunctionName (p. 822)

O nome da função Lambda.

Formatos de nome

- Nome da função - my-function.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- ARN parcial - 123456789012:function:my-function.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+\-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 204
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 204 com um corpo HTTP vazio.

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceConflictException

O recurso já existe ou outra operação está em andamento.

- Código de status HTTP: 409
`ResourceNotFoundException`
 - O recurso especificado na solicitação não existe.
- Código de status HTTP: 404
`ServiceException`
 - O serviço AWS Lambda encontrou um erro interno.
- Código de status HTTP: 500
`TooManyRequestsException`
 - O limite de taxa de transferência da solicitação foi excedido.
- Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteFunctionEventInvokeConfig

Exclui a configuração para invocação assíncrona de uma função, uma versão ou um alias.

Para configurar opções de invocação assíncrona, use [PutFunctionEventInvokeConfig](#) (p. 933).

Sintaxe da solicitação

```
DELETE /2019-09-25/functions/FunctionName/event-invoke-config?Qualifier=Qualifier HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 824)

O nome da função, versão ou alias do Lambda.

Formatos de nome

- Function name - `my-function` (somente nome), `my-function:v1` (com alias).
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- ARN parcial - `123456789012:function:my-function`.

Você pode anexar um número de versão ou alias a qualquer um dos formatos. A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Obrigatório: sim

[Qualifier](#) (p. 824)

Um número de versão ou nome de alias.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: `(|[a-zA-Z0-9$-_]+)`

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 204
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 204 com um corpo HTTP vazio.

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceConflictException

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteLayerVersion

Exclui uma versão de uma [camada do AWS Lambda](#). Versões excluídas não podem mais ser visualizadas ou adicionadas a funções. Para evitar quebrar funções, uma cópia da versão permanece no Lambda até que nenhuma função se refira a ela.

Sintaxe da solicitação

```
DELETE /2018-10-31/layers/LayerName/versions/VersionNumber HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[LayerName](#) (p. 826)

O nome ou o nome de recurso da Amazon (ARN) da camada.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (`arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\:\d{12}:layer:[a-zA-Z0-9-_+]|[a-zA-Z0-9-_]+`)

Obrigatório: sim

[VersionNumber](#) (p. 826)

O número da versão.

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 204
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 204 com um corpo HTTP vazio.

Errors

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

[TooManyRequestsException](#)

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteProvisionedConcurrencyConfig

Exclui a configuração de simultaneidade provisionada para uma função.

Sintaxe da solicitação

```
DELETE /2019-09-30/functions/FunctionName/provisioned-concurrency?Qualifier=Qualifier
HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 828)

O nome da função Lambda.

Formatos de nome

- Nome da função - my-function.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- ARN parcial - 123456789012:function:my-function.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*):lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[Qualifier](#) (p. 828)

O número de versão ou nome de alias.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 204
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 204 com um corpo HTTP vazio.

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceConflictException

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetAccountSettings

Recupera detalhes sobre os [limites](#) de sua conta e o uso em uma região da AWS.

Sintaxe da solicitação

```
GET /2016-08-19/account-settings/ HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação não usa nenhum parâmetro de URI.

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
    "AccountLimit": {
        "CodeSizeUnzipped": number,
        "CodeSizeZipped": number,
        "ConcurrentExecutions": number,
        "TotalCodeSize": number,
        "UnreservedConcurrentExecutions": number
    },
    "AccountUsage": {
        "FunctionCount": number,
        "TotalCodeSize": number
    }
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[AccountLimit](#) (p. 830)

Limites relacionados à simultaneidade e ao armazenamento de código.

Tipo: objeto [AccountLimit](#) (p. 990)

[AccountUsage](#) (p. 830)

O número de funções e a quantidade de armazenamento em uso.

Tipo: objeto [AccountUsage](#) (p. 992)

Errors

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500
TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetAlias

Retorna detalhes sobre o [alias](#) de uma função do Lambda.

Sintaxe da solicitação

```
GET /2015-03-31/functions/FunctionName/aliases/Name HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 832)

O nome da função Lambda.

Formatos de nome

- Nome da função - `MyFunction`.
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- ARN parcial - `123456789012:function:MyFunction`.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Obrigatório: sim

[Name](#) (p. 832)

O nome do alias.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: `(?!^[\d-]+$)([a-zA-Z0-9-_]+)`

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
    "AliasArn": "string",
    "Description": "string",
    "FunctionVersion": "string",
```

```
"Name": "string",
"RevisionId": "string",
"RoutingConfig": {
    "AdditionalVersionWeights": {
        "string" : number
    }
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[AliasArn](#) (p. 832)

O nome do recurso da Amazon (ARN) do alarme do alias.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$\\$LATEST|[a-zA-Z0-9-_]+))?

[Description](#) (p. 832)

Uma descrição do alias.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

[FunctionVersion](#) (p. 832)

A versão da função que o alias invoca.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Pattern: (\\$\\$LATEST|[0-9]+)

[Name](#) (p. 832)

O nome do alias.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: (? !^ [0-9]+ \\$) ([a-zA-Z0-9-_]+)

[RevisionId](#) (p. 832)

Um identificador exclusivo que muda ao atualizar o alias.

Tipo: sequência

[RoutingConfig](#) (p. 832)

A configuração de roteamento do alias.

Tipo: objeto [AliasRoutingConfiguration](#) (p. 995)

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetCodeSigningConfig

Retorna informações sobre a configuração de assinatura de código especificada.

Sintaxe da solicitação

```
GET /2020-04-22/code-signing-configs/CodeSigningConfigArn HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[CodeSigningConfigArn](#) (p. 835)

O nome do recurso da Amazon (ARN) da configuração de assinatura de código.

Restrições de tamanho: tamanho máximo de 200.

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}((-gov)|(-iso(b?)))?- [a-z]+-\d{1}:\d{12}:code-signing-config:csc-[a-zA-Z0-9]{17}

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
  "CodeSigningConfig": {
    "AllowedPublishers": {
      "SigningProfileVersionArns": [ "string" ]
    },
    "CodeSigningConfigArn": "string",
    "CodeSigningConfigId": "string",
    "CodeSigningPolicies": {
      "UntrustedArtifactOnDeployment": "string"
    },
    "Description": "string",
    "LastModified": "string"
  }
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[CodeSigningConfig](#) (p. 835)

A configuração de assinatura de código

Tipo: objeto [CodeSigningConfig](#) (p. 997)

Errors

InvalidOperationException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetEventSourceMapping

Retorna detalhes sobre um mapeamento de origem do evento. Você pode obter o identificador de um mapeamento a partir da saída de [ListEventSourceMappings](#) (p. 888).

Sintaxe da solicitação

```
GET /2015-03-31/event-source-mappings/UUID HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

UUID (p. 837)

O identificador do mapeamento da origem do evento.

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
    "BatchSize": number,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "FunctionResponseTypes": [ "string" ],
    "LastModified": number,
    "LastProcessingResult": "string",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "Queues": [ "string" ],
    "SelfManagedEventSource": {
        "Endpoints": {
            "string": [ "string" ]
        }
    },
    "SourceAccessConfigurations": [
        {
            "Type": "string",
            "URI": "string"
        }
    ]
}
```

```
        },
        "StartingPosition": "string",
        "StartingPositionTimestamp": number,
        "State": "string",
        "StateTransitionReason": "string",
        "Topics": [ "string" ],
        "TumblingWindowInSeconds": number,
        "UUID": "string"
    }
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[BatchSize \(p. 837\)](#)

O número máximo de registros em cada batch que o Lambda extrai da sua transmissão ou fila e envia para sua função. O Lambda transmite todos os registros no batch para a função em uma única chamada até o limite de carga útil para invocação síncrona (6 MB).

Valor padrão: varia de acordo com o serviço. Para o Amazon SQS, o padrão é 10. Para todos os outros serviços, o padrão é 100.

Configuração relacionada: quando você define `BatchSize` como um valor maior que 10, deve definir `MaximumBatchingWindowInSeconds` como pelo menos 1.

Tipo: inteiro

Faixa válida: valor mínimo de 1. Valor máximo de 10000.

[BisectBatchOnFunctionError \(p. 837\)](#)

(Somente streams) Se a função retornar um erro, divida o lote em dois e tente novamente. O valor padrão é falso.

Tipo: booliano

[DestinationConfig \(p. 837\)](#)

(Somente streams) Uma fila do Amazon SQS ou um destino de tópico do Amazon SNS para registros descartados.

Tipo: objeto [DestinationConfig \(p. 1002\)](#)

[EventSourceArn \(p. 837\)](#)

O nome de recurso da Amazon (ARN) da origem do evento.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+:([a-z]{2}(-gov)?-[a-z]+\d{1})?:(\d{12})?:(.*)

[FunctionArn \(p. 837\)](#)

O ARN da função Lambda.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[FunctionResponseTypes \(p. 837\)](#)

(Somente transmissões) Uma lista de enums de tipos de resposta atuais aplicados ao mapeamento de fontes de eventos.

Tipo: matriz de strings

Membros da matriz: número mínimo de 0 itens. Número máximo de 1 item.

Valores válidos: `ReportBatchItemFailures`

[LastModified \(p. 837\)](#)

A data em que o mapeamento de fontes de eventos foi atualizado pela última vez ou seu estado mudou, em segundos no horário do Unix.

Type: timestamp

[LastProcessingResult \(p. 837\)](#)

O resultado da última invocação do Lambda da sua função.

Tipo: sequência

[MaximumBatchingWindowInSeconds \(p. 837\)](#)

(Transmissões e filas padrão do Amazon SQS) O tempo máximo usado pelo Lambda, em segundos, para reunir os registros antes de invocar a função.

Padrão: 0

Configuração relacionada: quando você define `BatchSize` como um valor maior que 10, deve definir `MaximumBatchingWindowInSeconds` como pelo menos 1.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 300.

[MaximumRecordAgeInSeconds \(p. 837\)](#)

(Somente streams) Descarta registros mais antigos que a idade especificada. O valor padrão é -1, o que define a idade máxima como infinito. Quando o valor é definido como infinito, o Lambda nunca descarta registros antigos.

Type: inteiro

Intervalo válido: valor mínimo de -1. Valor máximo de 604800.

[MaximumRetryAttempts \(p. 837\)](#)

(Somente streams) Descarta registros após o número especificado de novas tentativas. O valor padrão é -1, o que define o número máximo de tentativas como infinito. Quando `MaximumRetryAttempts` é infinito, o Lambda tenta executar novamente os registros com falha até que o registro expire na fonte de eventos.

Type: inteiro

Intervalo válido: valor mínimo de -1. Valor máximo de 10000.

[ParallelizationFactor \(p. 837\)](#)

(Somente transmissões) O número de lotes a serem processados de cada fragmento simultaneamente. O valor padrão é 1.

Type: inteiro

Faixa válida: valor mínimo de 1. Valor máximo de 10.

[Queues \(p. 837\)](#)

(Amazon MQ) O nome da fila de destino do agente do Amazon MQ a ser consumido.

Tipo: matriz de strings

Membros da matriz: número fixo de 1 item.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1000.

Pattern: [\s\S]*

[SelfManagedEventSource \(p. 837\)](#)

O cluster autogerenciado do Apache Kafka para sua fonte de eventos.

Tipo: objeto [SelfManagedEventSource \(p. 1036\)](#)

[SourceAccessConfigurations \(p. 837\)](#)

Uma matriz do protocolo de autenticação, os componentes da VPC ou o host virtual para proteger e definir a fonte de eventos.

Tipo: matriz de [SourceAccessConfiguration \(p. 1037\)](#) objetos

Membros da matriz: número mínimo de 0 itens. Número máximo de 22 itens.

[StartingPosition \(p. 837\)](#)

A posição em um fluxo da qual você deseja iniciar a leitura. Obrigatório para origens de fluxo do Amazon Kinesis, Amazon DynamoDB e Amazon MSK. AT_TIMESTAMP só é compatível com o Amazon Kinesis Streams.

Tipo: sequência

Valores válidos: TRIM_HORIZON | LATEST | AT_TIMESTAMP

[StartingPositionTimestamp \(p. 837\)](#)

Com `StartingPosition` definido como AT_TIMESTAMP, o tempo a partir do qual iniciar a leitura em segundos no horário do Unix.

Type: timestamp

[State \(p. 837\)](#)

O estado do mapeamento da fonte de eventos. Pode ser um destes: Creating, Enabling, Enabled, Disabling, Disabled, Updating ou Deleting.

Tipo: sequência

[StateTransitionReason \(p. 837\)](#)

Indica se um usuário ou o Lambda fez a última alteração no mapeamento de fontes de eventos.

Tipo: sequência

[Topics \(p. 837\)](#)

O nome do tópico do Kafka.

Tipo: matriz de strings

Membros da matriz: número fixo de 1 item.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 249.

Pattern: ^[^.]([a-zA-Z0-9\-_\.]+)

[TumblingWindowInSeconds \(p. 837\)](#)

(Somente transmissões) A duração, em segundos, de uma janela de processamento. O intervalo é 1 a 900 segundos.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 900.

[UUID \(p. 837\)](#)

O identificador do mapeamento de fontes de eventos.

Tipo: sequência

Errors

[InvalidParameterValueException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ResourceNotFoundException](#)

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

[TooManyRequestsException](#)

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetFunction

Retorna informações sobre a função ou versão da função, com um link para fazer download do pacote de implantação que é válido por 10 minutos. Se você especificar uma versão da função, somente os detalhes específicos dela serão retornados.

Sintaxe da solicitação

```
GET /2015-03-31/functions/FunctionName?Qualifier=Qualifier HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 842)

O nome da função, versão ou alias do Lambda.

Formatos de nome

- Function name - `my-function` (somente nome), `my-function:v1` (com alias).
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- ARN parcial - `123456789012:function:my-function`.

Você pode anexar um número de versão ou alias a qualquer um dos formatos. A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 170.

Pattern: (`arn:(aws[a-zA-Z-]*)?:lambda:`)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[Qualifier](#) (p. 842)

Especifique uma versão ou alias para obter detalhes sobre uma versão publicada da função.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
  "Code": {
    "ImageUri": "string",
    "Location": "string",
```

```
    "RepositoryType": "string",
    "ResolvedImageUri": "string"
},
"Concurrency": {
    "ReservedConcurrentExecutions": number
},
"Configuration": {
    "CodeSha256": "string",
    "CodeSize": number,
    "DeadLetterConfig": {
        "TargetArn": "string"
    },
    "Description": "string",
    "Environment": {
        "Error": {
            "ErrorCode": "string",
            "Message": "string"
        },
        "Variables": {
            "string" : "string"
        }
    },
    "FileSystemConfigs": [
        {
            "Arn": "string",
            "LocalMountPath": "string"
        }
    ],
    "FunctionArn": "string",
    "FunctionName": "string",
    "Handler": "string",
    "ImageConfigResponse": {
        "Error": {
            "ErrorCode": "string",
            "Message": "string"
        },
        "ImageConfig": {
            "Command": [ "string" ],
            "EntryPoint": [ "string" ],
            "WorkingDirectory": "string"
        }
    },
    "KMSKeyArn": "string",
    "LastModified": "string",
    "LastUpdateStatus": "string",
    "LastUpdateStatusReason": "string",
    "LastUpdateStatusReasonCode": "string",
    "Layers": [
        {
            "Arn": "string",
            "CodeSize": number,
            "SigningJobArn": "string",
            "SigningProfileVersionArn": "string"
        }
    ],
    "MasterArn": "string",
    "MemorySize": number,
    "PackageType": "string",
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "string",
    "SigningJobArn": "string",
    "SigningProfileVersionArn": "string",
    "State": "string",
    "StateReason": "string",
    "StateReasonCode": "string",
    "Timeout": "string"
}
```

```
    "Timeout": number,
    "TracingConfig": {
        "Mode": "string"
    },
    "Version": "string",
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
        "SubnetIds": [ "string" ],
        "VpcId": "string"
    }
},
"Tags": {
    "string": "string"
}
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[Code](#) (p. 842)

O pacote de implantação da função ou versão.

Tipo: objeto [FunctionCodeLocation](#) (p. 1014)

[Concurrency](#) (p. 842)

A [simultaneidade reservada](#) da função.

Tipo: objeto [Concurrency](#) (p. 1000)

[Configuration](#) (p. 842)

A configuração da função ou versão.

Tipo: objeto [FunctionConfiguration](#) (p. 1015)

[Tags](#) (p. 842)

As [etiquetas](#) da função.

Tipo:: mapa de string para string

Errors

[InvalidParameterValueException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ResourceNotFoundException](#)

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500
`TooManyRequestsException`

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetFunctionCodeSigningConfig

Retorna a configuração de assinatura de código para a função especificada.

Sintaxe da solicitação

```
GET /2020-06-30/functions/FunctionName/code-signing-config HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

FunctionName (p. 846)

O nome da função Lambda.

Formatos de nome

- Nome da função - MyFunction.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- ARN parcial - 123456789012:function:MyFunction.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*):lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
  "CodeSigningConfigArn": "string",
  "FunctionName": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

CodeSigningConfigArn (p. 846)

O nome do recurso da Amazon (ARN) da configuração de assinatura de código.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 200.

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}((-gov)|(-iso(b?)))?-+[a-z]+-\d{1}:\d{12}:code-signing-config:csc-[a-zA-Z0-9]{17}

[FunctionName \(p. 846\)](#)

O nome da função Lambda.

Formatos de nome

- Nome da função - MyFunction.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- ARN parcial - 123456789012:function:MyFunction.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Errors

InvalidArgumentException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK para JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

GetFunctionConcurrency

Retorna detalhes sobre a configuração de simultaneidade reservada para uma função. Para definir um limite de simultaneidade para uma função, use [PutFunctionConcurrency](#) (p. 930).

Sintaxe da solicitação

```
GET /2019-09-30/functions/FunctionName/concurrency HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 849)

O nome da função Lambda.

Formatos de nome

- Nome da função - my-function.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- ARN parcial - 123456789012:function:my-function.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*):lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
  "ReservedConcurrentExecutions": number
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[ReservedConcurrentExecutions](#) (p. 849)

O número de execuções simultâneas que estão reservadas para a função.

Type: inteiro

Faixa válida: valor mínimo de 0.

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetFunctionConfiguration

Retorna as configurações específicas da versão de uma função ou versão do Lambda. A saída inclui apenas opções que podem variar entre as versões de uma função. Para modificar essas configurações, use [UpdateFunctionConfiguration](#) (p. 974).

Para obter todos os detalhes de uma função, incluindo configurações em nível de função, use [GetFunction](#) (p. 842).

Sintaxe da solicitação

```
GET /2015-03-31/functions/FunctionName/configuration?Qualifier=Qualifier HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 851)

O nome da função, versão ou alias do Lambda.

Formatos de nome

- Function name - `my-function` (somente nome), `my-function:v1` (com alias).
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- ARN parcial - `123456789012:function:my-function`.

Você pode anexar um número de versão ou alias a qualquer um dos formatos. A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 170.

Pattern: (`arn:(aws[a-zA-Z-]*)?:lambda:`)?([a-z]{2}(-gov)?-[a-z]+\-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[Qualifier](#) (p. 851)

Especifique uma versão ou alias para obter detalhes sobre uma versão publicada da função.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json
{
```

```
"CodeSha256": "string",
"CodeSize": number,
"DeadLetterConfig": {
    "TargetArn": "string"
},
"Description": "string",
"Environment": {
    "Error": {
        "ErrorCode": "string",
        "Message": "string"
    },
    "Variables": {
        "string" : "string"
    }
},
"FileSystemConfigs": [
    {
        "Arn": "string",
        "LocalMountPath": "string"
    }
],
"FunctionArn": "string",
"FunctionName": "string",
"Handler": "string",
"ImageConfigResponse": {
    "Error": {
        "ErrorCode": "string",
        "Message": "string"
    },
    "ImageConfig": {
        "Command": [ "string" ],
        "EntryPoint": [ "string" ],
        "WorkingDirectory": "string"
    }
},
"KMSKeyArn": "string",
"LastModified": "string",
"LastUpdateStatus": "string",
"LastUpdateStatusReason": "string",
"LastUpdateStatusReasonCode": "string",
"Layers": [
    {
        "Arn": "string",
        "CodeSize": number,
        "SigningJobArn": "string",
        "SigningProfileVersionArn": "string"
    }
],
"MasterArn": "string",
"MemorySize": number,
"PackageType": "string",
"RevisionId": "string",
"Role": "string",
"Runtime": "string",
"SigningJobArn": "string",
"SigningProfileVersionArn": "string",
"State": "string",
"StateReason": "string",
"StateReasonCode": "string",
"Timeout": number,
"TracingConfig": {
    "Mode": "string"
},
"Version": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ]
}
```

```
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[CodeSha256 \(p. 851\)](#)

O hash SHA256 do pacote de implantação da função.

Tipo: sequência

[CodeSize \(p. 851\)](#)

O tamanho do pacote de implantação da função em bytes.

Tipo: longo

[DeadLetterConfig \(p. 851\)](#)

A fila de mensagens mortas da função.

Tipo: objeto [DeadLetterConfig \(p. 1001\)](#)

[Description \(p. 851\)](#)

A descrição da função.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

[Environment \(p. 851\)](#)

As [variáveis de ambiente](#) da função.

Tipo: objeto [EnvironmentResponse \(p. 1005\)](#)

[FileSystemConfigs \(p. 851\)](#)

Configurações de conexão para um [sistema de arquivos do Amazon EFS](#).

Tipo: matriz de [FileSystemConfig \(p. 1011\)](#) objetos

Membros da matriz: número máximo de 1 item.

[FunctionArn \(p. 851\)](#)

O nome do recurso da Amazon (ARN) da função.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[FunctionName \(p. 851\)](#)

Nome da função.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 170.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Handler \(p. 851\)](#)

A função que o Lambda chama para começar a executar a função.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 128.

Pattern: `[^\s]+`

[ImageConfigResponse \(p. 851\)](#)

Os valores de configuração da imagem da função.

Tipo: objeto [ImageConfigResponse \(p. 1025\)](#)

[KMSKeyArn \(p. 851\)](#)

A chave KMS usada para criptografar as variáveis do ambiente da função. Esta chave é retornada somente se você tiver configurado uma CMK gerenciada pelo cliente.

Tipo: sequência

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:[^.]+|()`

[LastModified \(p. 851\)](#)

A data e a hora em que a função foi atualizada, no [formato ISO-8601](#) (AAAA-MM-DDThh:mm:ss.sTZD).

Tipo: sequência

[LastUpdateStatus \(p. 851\)](#)

O status da última atualização que foi executada na função. Ele é definido pela primeira vez como `Successful` após a conclusão da criação da função.

Tipo: sequência

Valores válidos: `Successful` | `Failed` | `InProgress`

[LastUpdateStatusReason \(p. 851\)](#)

O motivo pelo qual foi realizada a última atualização na função.

Tipo: sequência

[LastUpdateStatusReasonCode \(p. 851\)](#)

O código do motivo pelo qual foi realizada a última atualização na função.

Tipo: sequência

Valores válidos: `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup` | `ImageDeleted` | `ImageAccessDenied` | `InvalidImage`

[Layers \(p. 851\)](#)

As [camadas](#) da função.

Tipo: matriz de [Layer \(p. 1026\)](#) objetos

[MasterArn \(p. 851\)](#)

Para funções do Lambda@Edge, o ARN da função mestre.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[MemorySize \(p. 851\)](#)

A quantidade de memória disponível para a função no tempo de execução.

Type: inteiro

Intervalo válido: valor mínimo de 128. Valor máximo de 10240.

[PackageType \(p. 851\)](#)

O tipo de pacote de implantação. Defina como `Image` para imagem de contêiner e defina `Zip` para arquivo de documento `.zip`.

Tipo: sequência

Valores válidos: `zip` | `Image`

[RevisionId \(p. 851\)](#)

A última revisão atualizada da função ou do alias.

Tipo: sequência

[Role \(p. 851\)](#)

A função de execução da função.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@/-/_]+

[Runtime \(p. 851\)](#)

O ambiente de execução da função do Lambda.

Tipo: sequência

Valores válidos: `nodejs` | `nodejs4.3` | `nodejs6.10` | `nodejs8.10` | `nodejs10.x` | `nodejs12.x` | `nodejs14.x` | `java8` | `java8.al2` | `java11` | `python2.7` | `python3.6` | `python3.7` | `python3.8` | `python3.9` | `dotnetcore1.0` | `dotnetcore2.0` | `dotnetcore2.1` | `dotnetcore3.1` | `nodejs4.3-edge` | `go1.x` | `ruby2.5` | `ruby2.7` | `provided` | `provided.al2`

[SigningJobArn \(p. 851\)](#)

O ARN do trabalho de assinatura.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-\-])+:([a-z]{2}(-gov)?-[a-z]+\d{1})?:(\d{12})?:(.*)

[SigningProfileVersionArn \(p. 851\)](#)

O ARN da versão do perfil de assinatura.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+:([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12})?:(.*)

[State \(p. 851\)](#)

O estado atual da função. Quando o estado é `Inactive`, você pode invocar a função para reativá-la.

Tipo: sequência

Valores válidos: `Pending` | `Active` | `Inactive` | `Failed`

[StateReason \(p. 851\)](#)

O motivo para o estado atual da função.

Tipo: sequência

[StateReasonCode \(p. 851\)](#)

O código do motivo para o estado atual da função. Quando o código for `Creating`, não será possível invocar ou modificar a função.

Tipo: sequência

Valores válidos: `Idle` | `Creating` | `Restoring` | `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfRange` | `InvalidSubnet` | `InvalidSecurityGroup` | `ImageDeleted` | `ImageAccessDenied` | `InvalidImage`

[Timeout \(p. 851\)](#)

A quantidade de tempo, em segundos, que o Lambda permite que uma função seja executada antes de encerrá-la.

Type: inteiro

Faixa válida: valor mínimo de 1.

[TracingConfig \(p. 851\)](#)

A configuração de rastreamento de AWS X-Ray da função.

Tipo: objeto [TracingConfigResponse \(p. 1040\)](#)

[Version \(p. 851\)](#)

A versão da função do Lambda.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 851\)](#)

A configuração de rede da função.

Tipo: objeto [VpcConfigResponse \(p. 1042\)](#)

Errors

InvalidOperationException

Um dos parâmetros na solicitação é inválido.

- Código de status HTTP: 400
`ResourceNotFoundException`
 - O recurso especificado na solicitação não existe.
- Código de status HTTP: 404
`ServiceException`
 - O serviço AWS Lambda encontrou um erro interno.
- Código de status HTTP: 500
`TooManyRequestsException`
 - O limite de taxa de transferência da solicitação foi excedido.
- Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetFunctionEventInvokeConfig

Recupera a configuração para invocação assíncrona de uma função, uma versão ou um alias.

Para configurar opções de invocação assíncrona, use [PutFunctionEventInvokeConfig](#) (p. 933).

Sintaxe da solicitação

```
GET /2019-09-25/functions/FunctionName/event-invoke-config?Qualifier=Qualifier HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 858)

O nome da função, versão ou alias do Lambda.

Formatos de nome

- Function name - `my-function` (somente nome), `my-function:v1` (com alias).
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- ARN parcial - `123456789012:function:my-function`.

Você pode anexar um número de versão ou alias a qualquer um dos formatos. A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: `(arn:(aws[a-zA-Z-]*):lambda:)([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Obrigatório: sim

[Qualifier](#) (p. 858)

Um número de versão ou nome de alias.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: `(|[a-zA-Z0-9$-_]+)`

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
  "DestinationConfig": {
    "OnFailure": {
```

```
        "Destination": "string"
    },
    "OnSuccess": {
        "Destination": "string"
    }
},
"FunctionArn": "string",
"LastModified": number,
"MaximumEventAgeInSeconds": number,
"MaximumRetryAttempts": number
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[DestinationConfig \(p. 858\)](#)

Um destino para eventos depois que eles foram enviados a uma função para processamento.

Destinations

- Function (Função) – o nome de recurso da Amazon (ARN) da função do Lambda.
- Queue (Fila) – o ARN de uma fila do SQS.
- Topic (Tópico) – o ARN de um tópico do SNS.
- Event Bus (Barramento de eventos) – o ARN de um barramento de eventos do Amazon EventBridge.

Tipo: objeto [DestinationConfig \(p. 1002\)](#)

[FunctionArn \(p. 858\)](#)

O nome de recurso da Amazon (ARN) da função.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?

[LastModified \(p. 858\)](#)

A data e a hora em que a configuração foi atualizada pela última vez, em segundos no tempo do Unix.

Type: timestamp

[MaximumEventAgeInSeconds \(p. 858\)](#)

A idade máxima de uma solicitação que o Lambda envia a uma função para processamento.

Type: inteiro

Intervalo válido: valor mínimo de 60. Valor máximo de 21600.

[MaximumRetryAttempts \(p. 858\)](#)

O número máximo de vezes para tentar novamente quando a função retorna um erro.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 2.

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetLayerVersion

Retorna informações sobre uma versão de uma [camada do AWS Lambda](#), com um link para fazer download do arquivo da camada que é válido por 10 minutos.

Sintaxe da solicitação

```
GET /2018-10-31/layers/LayerName/versions/VersionNumber HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[LayerName](#) (p. 861)

O nome ou o nome de recurso da Amazon (ARN) da camada.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_+])|[a-zA-Z0-9-_+]

Obrigatório: sim

[VersionNumber](#) (p. 861)

O número da versão.

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
    "CompatibleRuntimes": [ "string" ],
    "Content": {
        "CodeSha256": "string",
        "CodeSize": number,
        "Location": "string",
        "SigningJobArn": "string",
        "SigningProfileVersionArn": "string"
    },
    "CreatedDate": "string",
    "Description": "string",
    "LayerArn": "string",
    "LayerVersionArn": "string",
    "LicenseInfo": "string",
    "Version": number
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[CompatibleRuntimes](#) (p. 861)

Os tempos de execução compatíveis da camada.

Tipo: matriz de strings

Membros da matriz: número máximo de 15 itens.

Valores válidos: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | nodejs10.x | nodejs12.x | nodejs14.x | java8 | java8.al2 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | python3.9 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | dotnetcore3.1 | nodejs4.3-edge | go1.x | ruby2.5 | ruby2.7 | provided | provided.al2

[Content](#) (p. 861)

Detalhes sobre a versão da camada.

Tipo: objeto [LayerVersionContentOutput](#) (p. 1029)

[CreatedDate](#) (p. 861)

A data em que a versão da camada foi criada, em [formato ISO-8601](#) (AAAA-MM-DDThh:mm:ss.sTZD).

Tipo: sequência

[Description](#) (p. 861)

A descrição da versão.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

[LayerArn](#) (p. 861)

O ARN da camada.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+

[LayerVersionArn](#) (p. 861)

O ARN da versão da camada.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

[LicenseInfo](#) (p. 861)

A licença de software da camada.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 512.

[Version \(p. 861\)](#)

O número da versão.

Type: longo

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetLayerVersionByArn

Retorna informações sobre uma versão de uma [camada do AWS Lambda](#), com um link para fazer download do arquivo da camada que é válido por 10 minutos.

Sintaxe da solicitação

```
GET /2018-10-31/layers?find=LayerVersion&Arn=Arn HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

Arn (p. 864)

O ARN da versão da camada.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
    "CompatibleRuntimes": [ "string" ],
    "Content": {
        "CodeSha256": "string",
        "CodeSize": number,
        "Location": "string",
        "SigningJobArn": "string",
        "SigningProfileVersionArn": "string"
    },
    "CreatedDate": "string",
    "Description": "string",
    "LayerArn": "string",
    "LayerVersionArn": "string",
    "LicenseInfo": "string",
    "Version": number
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[CompatibleRuntimes \(p. 864\)](#)

Os tempos de execução compatíveis da camada.

Tipo: matriz de strings

Membros da matriz: número máximo de 15 itens.

Valores válidos: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | nodejs10.x | nodejs12.x | nodejs14.x | java8 | java8.al2 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | python3.9 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | dotnetcore3.1 | nodejs4.3-edge | go1.x | ruby2.5 | ruby2.7 | provided | provided.al2

[Content \(p. 864\)](#)

Detalhes sobre a versão da camada.

Tipo: objeto [LayerVersionContentOutput \(p. 1029\)](#)

[CreatedDate \(p. 864\)](#)

A data em que a versão da camada foi criada, em [formato ISO-8601](#) (AAAA-MM-DDThh:mm:ss.sTZD).

Tipo: sequência

[Description \(p. 864\)](#)

A descrição da versão.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

[LayerArn \(p. 864\)](#)

O ARN da camada.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+

[LayerVersionArn \(p. 864\)](#)

O ARN da versão da camada.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

[LicenseInfo \(p. 864\)](#)

A licença de software da camada.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 512.

[Version \(p. 864\)](#)

O número da versão.

Type: longo

Errors

InvalidArgumentException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetLayerVersionPolicy

Retorna a política de permissões de uma versão de uma [camada do AWS Lambda](#). Para obter mais informações, consulte [AddLayerVersionPermission \(p. 771\)](#).

Sintaxe da solicitação

```
GET /2018-10-31/layers/LayerName/versions/VersionNumber/policy HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[LayerName \(p. 867\)](#)

O nome ou o nome de recurso da Amazon (ARN) da camada.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (`arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\d{12}:layer:[a-zA-Z0-9-_+]|[a-zA-Z0-9-_]+`)

Obrigatório: sim

[VersionNumber \(p. 867\)](#)

O número da versão.

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
    "Policy": "string",
    "RevisionId": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[Policy \(p. 867\)](#)

O documento da política.

Tipo: sequência

[RevisionId \(p. 867\)](#)

Um identificador exclusivo da revisão atual da política.

Tipo: sequência

Errors

InvalidOperationException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetPolicy

Retorna a [política do IAM baseada em recursos](#) referente a uma função, uma versão ou um alias.

Sintaxe da solicitação

```
GET /2015-03-31/functions/FunctionName/policy?Qualifier=Qualifier HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 869)

O nome da função, versão ou alias do Lambda.

Formatos de nome

- Function name - `my-function` (somente nome), `my-function:v1` (com alias).
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- ARN parcial - `123456789012:function:my-function`.

Você pode anexar um número de versão ou alias a qualquer um dos formatos. A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 170.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Obrigatório: sim

[Qualifier](#) (p. 869)

Especifique uma versão ou alias para obter a política para esse recurso.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: `(|[a-zA-Z0-9$-_]+)`

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
  "Policy": "string",
  "RevisionId": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[Policy \(p. 869\)](#)

Uma política baseada em recursos.

Tipo: sequência

[RevisionId \(p. 869\)](#)

Um identificador exclusivo da revisão atual da política.

Tipo: sequência

Errors

`InvalidParameterValueException`

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

`ResourceNotFoundException`

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

`ServiceException`

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

`TooManyRequestsException`

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetProvisionedConcurrencyConfig

Recupera a configuração de simultaneidade provisionada para o alias ou a versão de uma função.

Sintaxe da solicitação

```
GET /2019-09-30/functions/FunctionName/provisioned-concurrency?Qualifier=Qualifier HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 872)

O nome da função Lambda.

Formatos de nome

- Nome da função - my-function.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- ARN parcial - 123456789012:function:my-function.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*):lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[Qualifier](#) (p. 872)

O número de versão ou nome de alias.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
    "AllocatedProvisionedConcurrentExecutions": number,
    "AvailableProvisionedConcurrentExecutions": number,
    "LastModified": "string",
    "RequestedProvisionedConcurrentExecutions": number,
    "Status": "string",
```

```
    "StatusReason": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[AllocatedProvisionedConcurrentExecutions \(p. 872\)](#)

A quantidade de simultaneidade provisionada alocada.

Type: inteiro

Faixa válida: valor mínimo de 0.

[AvailableProvisionedConcurrentExecutions \(p. 872\)](#)

A quantidade de simultaneidade provisionada disponível.

Type: inteiro

Faixa válida: valor mínimo de 0.

[LastModified \(p. 872\)](#)

A data e hora em que um usuário atualizou a configuração pela última vez, no [formato ISO 8601](#).

Tipo: sequência

[RequestedProvisionedConcurrentExecutions \(p. 872\)](#)

A quantidade de simultaneidade provisionada solicitada.

Type: inteiro

Faixa válida: valor mínimo de 1.

[Status \(p. 872\)](#)

O status do processo de alocação.

Tipo: sequência

Valores válidos: IN_PROGRESS | READY | FAILED

[StatusReason \(p. 872\)](#)

Para alocações com falha, o motivo pelo qual a simultaneidade provisionada não pôde ser alocada.

Tipo: sequência

Errors

[InvalidOperationException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ProvisionedConcurrencyConfigNotFoundException](#)

A configuração especificada não existe.

- Código de status HTTP: 404
`ResourceNotFoundException`
 - O recurso especificado na solicitação não existe.
- Código de status HTTP: 404
`ServiceException`
 - O serviço AWS Lambda encontrou um erro interno.
- Código de status HTTP: 500
`TooManyRequestsException`
 - O limite de taxa de transferência da solicitação foi excedido.
- Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

Invoke

Invoca uma função do Lambda. Você pode invocar uma função de forma síncrona (e aguardar pela resposta) ou assíncrona. Para invocar uma função de forma assíncrona, defina `InvocationType` como `Event`.

Para a [invocação síncrona](#), detalhes sobre a resposta da função, incluindo erros, são incluídos no corpo da resposta e cabeçalhos. Para qualquer tipo de invocação, você pode ver mais informações no [log de execução](#) e no [rastreamento](#).

Quando ocorre um erro, a função pode ser invocada várias vezes. O comportamento de repetição varia de acordo com o tipo de erro, cliente, origem de evento e tipo de invocação. Por exemplo, se você invocar uma função assíncrona e ela retornar um erro, o Lambda executará a função até duas vezes mais. Para obter mais informações, consulte [Tentar comportamento novamente](#).

Para [invocação assíncrona](#), o Lambda adiciona eventos à fila antes de enviá-los para sua função. Se sua função não tiver capacidade suficiente para acompanhar a fila, os eventos podem ser perdidos. Ocasionalmente, sua função pode receber o mesmo evento várias vezes, mesmo que não ocorra nenhum erro. Para reter eventos que não foram processados, configure sua função com uma [fila de mensagens mortas](#).

O código de status na resposta da API não reflete erros de função. Códigos de erro são reservados para erros que impedem a execução da função, como erros de permissões, [erros de limite](#) ou problemas com o código e a configuração da sua função. Por exemplo, o Lambda retorna `TooManyRequestsException` se executar a função fizer com que você exceda um limite de simultaneidade no nível da conta (`ConcurrentInvocationLimitExceeded`) ou nível de função (`ReservedFunctionConcurrentInvocationLimitExceeded`).

Para funções com um longo tempo limite, o cliente pode ser desconectado durante a invocação síncrona enquanto aguarda por uma resposta. Configure seu cliente HTTP, SDK, firewall, proxy ou sistema operacional para permitir conexões longas com tempo limite ou configurações de ativação.

Essa operação exige permissão para a ação `lambda:InvokeFunction`.

Sintaxe da solicitação

```
POST /2015-03-31/functions/FunctionName/invocations?Qualifier=Qualifier HTTP/1.1
X-Amz-Invocation-Type: InvocationType
X-Amz-Log-Type: LogType
X-Amz-Client-Context: ClientContext

Payload
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[ClientContext](#) (p. 875)

Até 3583 bytes de dados codificados em base64 sobre o cliente que faz a invocação para passar para a função no objeto de contexto.

[FunctionName](#) (p. 875)

O nome da função, versão ou alias do Lambda.

Formatos de nome

- Function name - `my-function` (somente nome), `my-function:v1` (com alias).

- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- ARN parcial - `123456789012:function:my-function`.

Você pode anexar um número de versão ou alias a qualquer um dos formatos. A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 170.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Obrigatório: sim

[InvocationType](#) (p. 875)

Escolha entre as opções a seguir.

- `RequestResponse` (padrão): invoque a função de forma síncrona. Mantenha a conexão aberta até que a função retorne uma resposta ou o tempo limite expire. A resposta da API inclui a resposta da função e dados adicionais.
- `Event`: invoque a função de forma assíncrona. Envie eventos que falham várias vezes para a fila de mensagens mortas da função (se ela estiver configurada). A resposta da API inclui apenas um código de status.
- `DryRun`: valide valores de parâmetros e verifique se o usuário ou a função tem permissão para invocar a função.

Valores válidos: `Event` | `RequestResponse` | `DryRun`

[LogType](#) (p. 875)

Definir como `Tail` para incluir o log de execução na resposta. Aplica-se apenas a funções invocadas de forma síncrona.

Valores válidos: `None` | `Tail`

[Qualifier](#) (p. 875)

Especifique uma versão ou alias para invocar uma versão publicada da função.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: `(|[a-zA-Z0-9$-_]+)`

Corpo da solicitação

A solicitação aceita os dados binários a seguir.

[Payload](#) (p. 875)

O JSON que você quer fornecer para sua função Lambda como entrada.

Sintaxe da resposta

```
HTTP/1.1 StatusCode
X-Amz-Function-Error: FunctionError
X-Amz-Log-Result: LogResult
X-Amz-Executed-Version: ExecutedVersion

Payload
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço retornará a resposta HTTP a seguir.

[StatusCode](#) (p. 876)

O código de status HTTP está no intervalo 200 para uma solicitação bem-sucedida. Para o tipo de invocação `RequestResponse`, esse código de status é 200. Para o tipo de invocação `Event`, esse código de status é 202. Para o tipo de invocação `DryRun`, o código de status é 204.

A resposta retorna os cabeçalhos HTTP a seguir.

[ExecutedVersion](#) (p. 876)

A versão da função que foi executada. Quando você invoca uma função com um alias, isso indica para qual versão o alias foi resolvido.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Pattern: `(\$LATEST|[0-9]+)`

[FunctionError](#) (p. 876)

Se presente, indica que ocorreu um erro durante a execução da função. Detalhes sobre o erro estão incluídos na carga útil da resposta.

[LogResult](#) (p. 876)

Os últimos 4 KB do log de execução, que é codificado em base64.

A resposta retorna as informações a seguir como corpo HTTP.

[Payload](#) (p. 876)

A resposta da função ou um objeto de erro.

Errors

`EC2AccessDeniedException`

São necessárias permissões adicionais para definir as configurações da VPC.

Código de status HTTP: 502

`EC2ThrottledException`

AWS Lambda foi acelerado pelo Amazon EC2 durante a inicialização da função do Lambda usando a função de execução fornecida para a função do Lambda.

Código de status HTTP: 502

`EC2UnexpectedException`

AWS Lambda recebeu uma exceção inesperada do cliente do EC2 ao configurar a função do Lambda.

Código de status HTTP: 502

`EFSIOException`

Ocorreu um erro ao ler ou gravar em um sistema de arquivos conectado.

Código de status HTTP: 410

EFSMountConnectivityException

A função não pôde estabelecer uma conexão de rede com o sistema de arquivos configurado.

Código de status HTTP: 408

EFSMountFailureException

A função não pôde montar o sistema de arquivos configurado devido a um problema de permissão ou configuração.

Código de status HTTP: 403

EFSMountTimeoutException

A função conseguiu se estabelecer uma conexão de rede com o sistema de arquivos configurado, mas a operação de montagem atingiu o tempo limite.

Código de status HTTP: 408

ENILimitReachedException

O AWS Lambda não conseguiu criar uma interface de rede elástica na VPC especificada como parte da configuração da função do Lambda porque o limite de interfaces de rede foi atingido.

Código de status HTTP: 502

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

InvalidRequestContentException

O corpo da solicitação não pôde ser analisado como JSON.

Código de status HTTP: 400

InvalidRuntimeException

O tempo de execução ou a versão do tempo de execução especificada não tem suporte.

Código de status HTTP: 502

InvalidSecurityGroupIDException

O ID do grupo de segurança especificado na configuração de VPC da função do Lambda é inválido.

Código de status HTTP: 502

InvalidSubnetIDException

O ID de sub-rede fornecido na configuração de VPC da função do Lambda é inválido.

Código de status HTTP: 502

InvalidZipFileException

AWS Lambda não pôde descompactar o pacote de implantação.

Código de status HTTP: 502

KMSAccessDeniedException

O Lambda não pôde descriptografar as variáveis de ambiente porque o acesso ao KMS foi negado.

Verifique as permissões do KMS da função Lambda.

Código de status HTTP: 502

KMSDisabledException

O Lambda não conseguiu descriptografar as variáveis de ambiente porque a chave do KMS usada está desabilitada. Verifique as configurações da chave do KMS da função Lambda.

Código de status HTTP: 502

KMSInvalidStateException

O Lambda não conseguiu descriptografar as variáveis de ambiente porque a chave do KMS usada está em um estado inválido para descriptografia. Verifique as configurações da chave do KMS da função.

Código de status HTTP: 502

KMSNotFoundException

O Lambda não conseguiu descriptografar as variáveis de ambiente porque a chave do KMS não foi encontrada. Verifique as configurações da chave do KMS da função.

Código de status HTTP: 502

RequestTooLargeException

A carga útil da solicitação excedeu o limite de entrada JSON do corpo da solicitação `Invoke`. Para obter mais informações, consulte [Limites](#).

Código de status HTTP: 413

ResourceConflictException

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ResourceNotReadyException

A função está inativa e sua conexão da VPC não está mais disponível. Aguarde até que a conexão da VPC seja restabelecida e tente novamente.

Código de status HTTP: 502

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

SubnetIPAddressLimitReachedException

O AWS Lambda não conseguiu configurar o acesso à VPC para a função do Lambda porque uma ou mais sub-redes configuradas não têm endereços IP disponíveis.

Código de status HTTP: 502

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

UnsupportedMediaTypeException

O tipo de conteúdo do corpo da solicitação `Invoke` não é JSON.

Código de status HTTP: 415

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

InvokeAsync

Essa ação está defasada.

Important

Para invocação de função assíncrona, use [Invoke \(p. 875\)](#).

Invoca sua função de forma assíncrona.

Sintaxe da solicitação

```
POST /2014-11-13/functions/FunctionName/invoke-async/ HTTP/1.1  
InvokeArgs
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName \(p. 881\)](#)

O nome da função Lambda.

Formatos de nome

- Nome da função - my-function.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- ARN parcial - 123456789012:function:my-function.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 170.

Pattern: (arn:(aws[a-zA-Z-]*):lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

Corpo da solicitação

A solicitação aceita os dados binários a seguir.

[InvokeArgs \(p. 881\)](#)

O JSON que você quer fornecer para sua função Lambda como entrada.

Obrigatório: sim

Sintaxe da resposta

```
HTTP/1.1 Status
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço retornará a resposta HTTP a seguir.

[Status \(p. 881\)](#)

O código do status.

Errors

`InvalidRequestContentException`

O corpo da solicitação não pôde ser analisado como JSON.

Código de status HTTP: 400

`InvalidRuntimeException`

O tempo de execução ou a versão do tempo de execução especificada não tem suporte.

Código de status HTTP: 502

`ResourceConflictException`

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

`ResourceNotFoundException`

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

`ServiceException`

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListAliases

Retorna uma lista de [aliases](#) de uma função do Lambda.

Sintaxe da solicitação

```
GET /2015-03-31/functions/FunctionName/aliases?  
FunctionVersion=FunctionVersion&Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 883)

O nome da função Lambda.

Formatos de nome

- Nome da função - `MyFunction`.
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- ARN parcial - `123456789012:function:MyFunction`.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (`arn:(aws[a-zA-Z-]*)?:lambda:`)?([a-z]{2}(-gov)?-[a-z]+\-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[FunctionVersion](#) (p. 883)

Especifique uma versão de função para listar apenas aliases que invocam essa versão.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Pattern: (\\$LATEST|[0-9]+)

[Marker](#) (p. 883)

Especifique o token de paginação retornado por uma solicitação anterior para recuperar a próxima página de resultados.

[MaxItems](#) (p. 883)

LIMITAR o número de aliases retornados.

Faixa válida: valor mínimo de 1. Valor máximo de 10000.

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
```

```
Content-type: application/json

{
  "Aliases": [
    {
      "AliasArn": "string",
      "Description": "string",
      "FunctionVersion": "string",
      "Name": "string",
      "RevisionId": "string",
      "RoutingConfig": {
        "AdditionalVersionWeights": {
          "string" : number
        }
      }
    }
  ],
  "NextMarker": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[Aliases \(p. 883\)](#)

Uma lista de aliases.

Tipo: matriz de [AliasConfiguration \(p. 993\)](#) objetos

[NextMarker \(p. 883\)](#)

O token de paginação incluído se houver mais resultados disponíveis.

Tipo: sequência

Errors

`InvalidParameterValueException`

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

`ResourceNotFoundException`

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

`ServiceException`

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

`TooManyRequestsException`

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListCodeSigningConfigs

Retorna uma lista de [configurações de assinatura de código](#). Uma solicitação retorna até 10.000 configurações por chamada. Você pode usar o parâmetro `MaxItems` para retornar menos configurações por chamada.

Sintaxe da solicitação

```
GET /2020-04-22/code-signing-configs/?Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[Marker](#) (p. 886)

Especifique o token de paginação retornado por uma solicitação anterior para recuperar a próxima página de resultados.

[MaxItems](#) (p. 886)

O número máximo de itens a serem retornados.

Faixa válida: valor mínimo de 1. Valor máximo de 10000.

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
  "CodeSigningConfigs": [
    {
      "AllowedPublishers": {
        "SigningProfileVersionArns": [ "string" ]
      },
      "CodeSigningConfigArn": "string",
      "CodeSigningConfigId": "string",
      "CodeSigningPolicies": {
        "UntrustedArtifactOnDeployment": "string"
      },
      "Description": "string",
      "LastModified": "string"
    }
  ],
  "NextMarker": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[CodeSigningConfigs \(p. 886\)](#)

As configurações de assinatura de código

Tipo: matriz de [CodeSigningConfig \(p. 997\)](#) objetos

[NextMarker \(p. 886\)](#)

O token de paginação incluído se houver mais resultados disponíveis.

Tipo: sequência

Errors

[InvalidOperationException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListEventSourceMappings

Lista os mapeamentos de origem do evento. Especifique um `EventSourceArn` para mostrar somente mapeamentos de origem do evento referentes a uma única origem do evento.

Sintaxe da solicitação

```
GET /2015-03-31/event-source-mappings/?  
EventSourceArn=EventSourceArn&FunctionName=FunctionName&Marker=Marker&MaxItems=MaxItems  
HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[EventSourceArn](#) (p. 888)

O nome de recurso da Amazon (ARN) da origem do evento.

- Amazon Kinesis - o ARN do stream de dados ou um consumidor de stream.
- Amazon DynamoDB Streams - o ARN do stream.
- Amazon Simple Queue Service - o ARN da fila.
- Amazon Managed Streaming for Apache Kafka – o ARN do cluster.

Pattern: `arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+([a-z]{2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.*?)`

[FunctionName](#) (p. 888)

O nome da função Lambda.

Formatos de nome

- Nome da função - `MyFunction`.
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- ARN da versão ou alias - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction:PROD`.
- ARN parcial - `123456789012:function:MyFunction`.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+\-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Marker](#) (p. 888)

Um token de paginação retornado por uma chamada anterior.

[MaxItems](#) (p. 888)

O número máximo de mapeamentos de origem do evento a serem retornados. Observe que `ListEventSourceMappings` retorna um máximo de 100 itens em cada resposta, mesmo que você defina um número maior.

Faixa válida: valor mínimo de 1. Valor máximo de 10000.

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
    "EventSourceMappings": [
        {
            "BatchSize": number,
            "BisectBatchOnFunctionError": boolean,
            "DestinationConfig": {
                "OnFailure": {
                    "Destination": "string"
                },
                "OnSuccess": {
                    "Destination": "string"
                }
            },
            "EventSourceArn": "string",
            "FunctionArn": "string",
            "FunctionResponseTypes": [ "string" ],
            "LastModified": number,
            "LastProcessingResult": "string",
            "MaximumBatchingWindowInSeconds": number,
            "MaximumRecordAgeInSeconds": number,
            "MaximumRetryAttempts": number,
            "ParallelizationFactor": number,
            "Queues": [ "string" ],
            "SelfManagedEventSource": {
                "Endpoints": {
                    "string": [ "string" ]
                }
            },
            "SourceAccessConfigurations": [
                {
                    "Type": "string",
                    "URI": "string"
                }
            ],
            "StartingPosition": "string",
            "StartingPositionTimestamp": number,
            "State": "string",
            "StateTransitionReason": "string",
            "Topics": [ "string" ],
            "TumblingWindowInSeconds": number,
            "UUID": "string"
        }
    ],
    "NextMarker": "string
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[EventSourceMappings \(p. 889\)](#)

Uma lista de mapeamentos da origem do evento.

Tipo: matriz de [EventSourceMappingConfiguration \(p. 1006\)](#) objetos

[NextMarker \(p. 889\)](#)

Um token de paginação retornado quando a resposta não contém todos os mapeamentos de origem do evento.

Tipo: sequência

Errors

[InvalidParameterValueException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ResourceNotFoundException](#)

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

[TooManyRequestsException](#)

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListFunctionEventInvokeConfigs

Recupera uma lista de configurações para invocação assíncrona de uma função

Para configurar opções de invocação assíncrona, use [PutFunctionEventInvokeConfig](#) (p. 933).

Sintaxe da solicitação

```
GET /2019-09-25/functions/FunctionName/event-invoke-config/list?  
Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 891)

O nome da função Lambda.

Formatos de nome

- Nome da função - my-function.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- ARN parcial - 123456789012:function:my-function.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*):lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[Marker](#) (p. 891)

Especifique o token de paginação retornado por uma solicitação anterior para recuperar a próxima página de resultados.

[MaxItems](#) (p. 891)

O número máximo de configurações a serem retornadas.

Faixa válida: valor mínimo de 1. Valor máximo de 50.

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200  
Content-type: application/json  
{
```

```
"FunctionEventInvokeConfigs": [
  {
    "DestinationConfig": {
      "OnFailure": {
        "Destination": "string"
      },
      "OnSuccess": {
        "Destination": "string"
      }
    },
    "FunctionArn": "string",
    "LastModified": number,
    "MaximumEventAgeInSeconds": number,
    "MaximumRetryAttempts": number
  }
],
"NextMarker": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[FunctionEventInvokeConfigs \(p. 891\)](#)

Uma lista de configurações.

Tipo: matriz de [FunctionEventInvokeConfig \(p. 1021\)](#) objetos

[NextMarker \(p. 891\)](#)

O token de paginação incluído se houver mais resultados disponíveis.

Tipo: sequência

Errors

[InvalidParameterValueException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ResourceNotFoundException](#)

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

[TooManyRequestsException](#)

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListFunctions

Retorna uma lista de funções do Lambda, com a configuração específica da versão de cada uma. O Lambda retorna até 50 funções por chamada.

Defina `FunctionVersion` como `ALL` para incluir todas as versões publicadas de cada função, além da versão não publicada.

Note

A ação `ListFunctions` retorna um subconjunto dos campos [FunctionConfiguration \(p. 1015\)](#). Para obter os campos adicionais (`State`, `StateReasonCode`, `StateReason`, `LastUpdateStatus`, `LastUpdateStatusReason`, `LastUpdateStatusReasonCode`) de uma função ou versão, use [GetFunction \(p. 842\)](#).

Sintaxe da solicitação

```
GET /2015-03-31/functions/?  
FunctionVersion=FunctionVersion&Marker=Marker&MasterRegion=MasterRegion&MaxItems=MaxItems  
HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionVersion \(p. 894\)](#)

Definir como `ALL` para incluir entradas para todas as versões publicadas de cada função.

Valores válidos: `ALL`

[Marker \(p. 894\)](#)

Especifique o token de paginação retornado por uma solicitação anterior para recuperar a próxima página de resultados.

[MasterRegion \(p. 894\)](#)

Para funções do Lambda@Edge, a Região da AWS da função mestre. Por exemplo, `us-east-1` filtra a lista de funções para incluir apenas funções do Lambda@Edge replicadas de uma função principal no Leste dos EUA (Norte da Virgínia). Se for especificado, é necessário configurar `FunctionVersion` como `ALL`.

Pattern: `ALL|[a-z]{2}(-gov)?-[a-z]+-\d{1}`

[MaxItems \(p. 894\)](#)

O número máximo de funções a ser retornado na resposta. Observe que `ListFunctions` retorna um máximo de 50 itens em cada resposta, mesmo que você defina um número maior.

Faixa válida: valor mínimo de 1. Valor máximo de 10000.

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
```

```
Content-type: application/json

{
  "Functions": [
    {
      "CodeSha256": "string",
      "CodeSize": number,
      "DeadLetterConfig": {
        "TargetArn": "string"
      },
      "Description": "string",
      "Environment": {
        "Error": {
          "ErrorCode": "string",
          "Message": "string"
        },
        "Variables": {
          "string" : "string"
        }
      },
      "FileSystemConfigs": [
        {
          "Arn": "string",
          "LocalMountPath": "string"
        }
      ],
      "FunctionArn": "string",
      "FunctionName": "string",
      "Handler": "string",
      "ImageConfigResponse": {
        "Error": {
          "ErrorCode": "string",
          "Message": "string"
        },
        "ImageConfig": {
          "Command": [ "string" ],
          "EntryPoint": [ "string" ],
          "WorkingDirectory": "string"
        }
      },
      "KMSKeyArn": "string",
      "LastModified": "string",
      "LastUpdateStatus": "string",
      "LastUpdateStatusReason": "string",
      "LastUpdateStatusReasonCode": "string",
      "Layers": [
        {
          "Arn": "string",
          "CodeSize": number,
          "SigningJobArn": "string",
          "SigningProfileVersionArn": "string"
        }
      ],
      "MasterArn": "string",
      "MemorySize": number,
      "PackageType": "string",
      "RevisionId": "string",
      "Role": "string",
      "Runtime": "string",
      "SigningJobArn": "string",
      "SigningProfileVersionArn": "string",
      "State": "string",
      "StateReason": "string",
      "StateReasonCode": "string",
      "Timeout": number,
      "TracingConfig": {
        "SamplingPercentage": number
      }
    }
  ]
}
```

```
        "Mode": "string"
    },
    "Version": "string",
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
        "SubnetIds": [ "string" ],
        "VpcId": "string"
    }
},
"NextMarker": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[Functions \(p. 894\)](#)

Uma lista de funções do Lambda.

Tipo: matriz de [FunctionConfiguration \(p. 1015\)](#) objetos

[NextMarker \(p. 894\)](#)

O token de paginação incluído se houver mais resultados disponíveis.

Tipo: sequência

Errors

[InvalidOperationException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

[TooManyRequestsException](#)

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)

- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListFunctionsByCodeSigningConfig

Lista as funções que usam a configuração de assinatura de código especificada. Você pode usar esse método antes de excluir uma configuração de assinatura de código para verificar se não está sendo utilizada por alguma função.

Sintaxe da solicitação

```
GET /2020-04-22/code-signing-configs/CodeSigningConfigArn/functions?  
Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[CodeSigningConfigArn](#) (p. 898)

O nome do recurso da Amazon (ARN) da configuração de assinatura de código.

Restrições de tamanho: tamanho máximo de 200.

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}((-gov)|(-iso(b?)))?-([a-zA-Z-]+)\d{1}:\d{12}:code-signing-config:csc-[a-zA-Z0-9]{17}

Obrigatório: sim

[Marker](#) (p. 898)

Especifique o token de paginação retornado por uma solicitação anterior para recuperar a próxima página de resultados.

[MaxItems](#) (p. 898)

O número máximo de itens a serem retornados.

Faixa válida: valor mínimo de 1. Valor máximo de 10000.

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200  
Content-type: application/json  
  
{  
    "FunctionArns": [ "string" ],  
    "NextMarker": "string"  
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[FunctionArns \(p. 898\)](#)

Os ARNs da função.

Tipo: matriz de strings

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[NextMarker \(p. 898\)](#)

O token de paginação incluído se houver mais resultados disponíveis.

Tipo: sequência

Errors

InvalidArgumentException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListLayers

Lista as [camadas do AWS Lambda](#) e mostra informações sobre a versão mais recente de cada uma delas. Especifique um [identificador de tempo de execução](#) para listar apenas camadas que indicam que são compatíveis com esse tempo de execução.

Sintaxe da solicitação

```
GET /2018-10-31/layers?CompatibleRuntime=CompatibleRuntime&Marker=Marker&MaxItems=MaxItems
HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

CompatibleRuntime ([p. 900](#))

Um identificador de tempo de execução. Por exemplo, go1.x.

Valores válidos: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | nodejs10.x | nodejs12.x | nodejs14.x | java8 | java8.al2 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | python3.9 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | dotnetcore3.1 | nodejs4.3-edge | go1.x | ruby2.5 | ruby2.7 | provided | provided.al2

Marker ([p. 900](#))

Um token de paginação retornado por uma chamada anterior.

MaxItems ([p. 900](#))

O número máximo de camadas a serem retornadas.

Faixa válida: valor mínimo de 1. Valor máximo de 50.

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
    "Layers": [
        {
            "LatestMatchingVersion": {
                "CompatibleRuntimes": [ "string" ],
                "CreatedDate": "string",
                "Description": "string",
                "LayerVersionArn": "string",
                "LicenseInfo": "string",
                "Version": number
            },
            "LayerArn": "string",
            "LayerName": "string"
        }
    ]
}
```

```
    ],
    "NextMarker": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[Layers \(p. 900\)](#)

Uma lista de camadas da função.

Tipo: matriz de [LayersListItem \(p. 1027\)](#) objetos

[NextMarker \(p. 900\)](#)

Um token de paginação retornado quando a resposta não contém todas as camadas.

Tipo: sequência

Errors

[InvalidOperationException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

[TooManyRequestsException](#)

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListLayerVersions

Lista as versões de um [camada AWS Lambda](#). As versões que foram excluídas não estão listadas. Especifique um [identificador de tempo de execução](#) para listar apenas versões que indicam que são compatíveis com esse tempo de execução.

Sintaxe da solicitação

```
GET /2018-10-31/layers/LayerName/versions?  
CompatibleRuntime=CompatibleRuntime&Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[CompatibleRuntime](#) (p. 903)

Um identificador de tempo de execução. Por exemplo, `go1.x`.

Valores válidos: `nodejs` | `nodejs4.3` | `nodejs6.10` | `nodejs8.10` | `nodejs10.x` | `nodejs12.x` | `nodejs14.x` | `java8` | `java8.al2` | `java11` | `python2.7` | `python3.6` | `python3.7` | `python3.8` | `python3.9` | `dotnetcore1.0` | `dotnetcore2.0` | `dotnetcore2.1` | `dotnetcore3.1` | `nodejs4.3-edge` | `go1.x` | `ruby2.5` | `ruby2.7` | `provided` | `provided.al2`

[LayerName](#) (p. 903)

O nome ou o nome de recurso da Amazon (ARN) da camada.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: `(arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\:\d{12}:layer:[a-zA-Z0-9-_]+|[a-zA-Z0-9-_]+)`

Obrigatório: sim

[Marker](#) (p. 903)

Um token de paginação retornado por uma chamada anterior.

[MaxItems](#) (p. 903)

O número máximo de versões a serem retornadas.

Faixa válida: valor mínimo de 1. Valor máximo de 50.

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200  
Content-type: application/json  
  
{  
  "LayerVersions": [  
    {
```

```
    "CompatibleRuntimes": [ "string" ],
    "CreatedDate": "string",
    "Description": "string",
    "LayerVersionArn": "string",
    "LicenseInfo": "string",
    "Version": number
}
],
"NextMarker": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[LayerVersions \(p. 903\)](#)

Uma lista de versões.

Tipo: matriz de [LayerVersionsListItem \(p. 1030\)](#) objetos

[NextMarker \(p. 903\)](#)

Um token de paginação retornado quando a resposta não contém todas as versões.

Tipo: sequência

Errors

[InvalidOperationException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ResourceNotFoundException](#)

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

[TooManyRequestsException](#)

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)

- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListProvisionedConcurrencyConfigs

Recupera uma lista de configurações de simultaneidade provisionadas para uma função.

Sintaxe da solicitação

```
GET /2019-09-30/functions/FunctionName/provisioned-concurrency?  
List=ALL&Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 906)

O nome da função Lambda.

Formatos de nome

- Nome da função - `my-function`.
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- ARN parcial - `123456789012:function:my-function`.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (`arn:(aws[a-zA-Z-]*)?:lambda:`)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[Marker](#) (p. 906)

Especifique o token de paginação retornado por uma solicitação anterior para recuperar a próxima página de resultados.

[MaxItems](#) (p. 906)

Especifique um número para limitar o número de configurações retornadas.

Faixa válida: valor mínimo de 1. Valor máximo de 50.

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200  
Content-type: application/json  
  
{  
  "NextMarker": "string",  
  "ProvisionedConcurrencyConfigs": [  
    {
```

```
        "AllocatedProvisionedConcurrentExecutions": number,
        "AvailableProvisionedConcurrentExecutions": number,
        "FunctionArn": "string",
        "LastModified": "string",
        "RequestedProvisionedConcurrentExecutions": number,
        "Status": "string",
        "StatusReason": "string"
    }
]
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[NextMarker](#) (p. 906)

O token de paginação incluído se houver mais resultados disponíveis.

Tipo: sequência

[ProvisionedConcurrencyConfigs](#) (p. 906)

Uma lista de configurações de simultaneidade provisionada.

Tipo: matriz de [ProvisionedConcurrencyConfigListItem](#) (p. 1034) objetos

Errors

[InvalidParameterValueException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ResourceNotFoundException](#)

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

[TooManyRequestsException](#)

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)

- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListTags

Retorna as [etiquetas](#) de uma função. Também é possível visualizar etiquetas com [GetFunction](#) (p. 842).

Sintaxe da solicitação

```
GET /2017-03-31/tags/ARN HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[ARN](#) (p. 909)

O nome do recurso da Amazon (ARN) da função. Observação: o Lambda não é compatível com a adição de etiquetas a aliases ou versões.

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
  "Tags": {
    "string" : "string"
  }
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[Tags](#) (p. 909)

As etiquetas da função.

Tipo:: mapa de string para string

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400
ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404
ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500
TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListVersionsByFunction

Retorna uma lista de [versões](#), com a configuração específica da versão de cada uma. O Lambda retorna até 50 versões por chamada.

Sintaxe da solicitação

```
GET /2015-03-31/functions/FunctionName/versions?Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 911)

O nome da função Lambda.

Formatos de nome

- Nome da função - MyFunction.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- ARN parcial - 123456789012:function:MyFunction.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 170.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[Marker](#) (p. 911)

Especifique o token de paginação retornado por uma solicitação anterior para recuperar a próxima página de resultados.

[MaxItems](#) (p. 911)

O número máximo de versões a serem retornadas. Observe que [ListVersionsByFunction](#) retorna um máximo de 50 itens em cada resposta, mesmo que você defina um número maior.

Faixa válida: valor mínimo de 1. Valor máximo de 10000.

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
    "NextMarker": "string",
    "Versions": [
```

```
{  
    "CodeSha256": "string",  
    "CodeSize": number,  
    "DeadLetterConfig": {  
        "TargetArn": "string"  
    },  
    "Description": "string",  
    "Environment": {  
        "Error": {  
            "ErrorCode": "string",  
            "Message": "string"  
        },  
        "Variables": {  
            "string" : "string"  
        }  
    },  
    "FileSystemConfigs": [  
        {  
            "Arn": "string",  
            "LocalMountPath": "string"  
        }  
    ],  
    "FunctionArn": "string",  
    "FunctionName": "string",  
    "Handler": "string",  
    "ImageConfigResponse": {  
        "Error": {  
            "ErrorCode": "string",  
            "Message": "string"  
        },  
        "ImageConfig": {  
            "Command": [ "string" ],  
            "EntryPoint": [ "string" ],  
            "WorkingDirectory": "string"  
        }  
    },  
    "KMSKeyArn": "string",  
    "LastModified": "string",  
    "LastUpdateStatus": "string",  
    "LastUpdateStatusReason": "string",  
    "LastUpdateStatusReasonCode": "string",  
    "Layers": [  
        {  
            "Arn": "string",  
            "CodeSize": number,  
            "SigningJobArn": "string",  
            "SigningProfileVersionArn": "string"  
        }  
    ],  
    "MasterArn": "string",  
    "MemorySize": number,  
    "PackageType": "string",  
    "RevisionId": "string",  
    "Role": "string",  
    "Runtime": "string",  
    "SigningJobArn": "string",  
    "SigningProfileVersionArn": "string",  
    "State": "string",  
    "StateReason": "string",  
    "StateReasonCode": "string",  
    "Timeout": number,  
    "TracingConfig": {  
        "Mode": "string"  
    },  
    "Version": "string",  
    "VpcConfig": {  
        "SubnetIds": [  
            "string"  
        ],  
        "SecurityGroupIds": [  
            "string"  
        ]  
    }  
}
```

```
        "SecurityGroupIds": [ "string" ],
        "SubnetIds": [ "string" ],
        "VpcId": "string"
    }
}
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[NextMarker](#) (p. 911)

O token de paginação incluído se houver mais resultados disponíveis.

Tipo: sequência

[Versions](#) (p. 911)

Uma lista das versões da função do Lambda.

Tipo: matriz de [FunctionConfiguration](#) (p. 1015) objetos

Errors

[InvalidParameterValueException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ResourceNotFoundException](#)

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

[TooManyRequestsException](#)

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)

- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

PublishLayerVersion

Cria uma [camada do AWS Lambda](#) de um arquivo ZIP. Cada vez que você chama `PublishLayerVersion` com o mesmo nome de camada, uma nova versão é criada.

Adicione camadas à sua função com [CreateFunction \(p. 796\)](#) ou [UpdateFunctionConfiguration \(p. 974\)](#).

Sintaxe da solicitação

```
POST /2018-10-31/layers/LayerName/versions HTTP/1.1
Content-type: application/json

{
  "CompatibleRuntimes": [ "string" ],
  "Content": {
    "S3Bucket": "string",
    "S3Key": "string",
    "S3ObjectVersion": "string",
    "ZipFile": blob
  },
  "Description": "string",
  "LicenseInfo": "string"
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[LayerName \(p. 915\)](#)

O nome ou o nome de recurso da Amazon (ARN) da camada.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_+])|[a-zA-Z0-9-_+]

Obrigatório: sim

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[CompatibleRuntimes \(p. 915\)](#)

Uma lista de [tempos de execução da função](#) compatíveis. Usado para filtragem com [ListLayers \(p. 900\)](#) e [ListLayerVersions \(p. 903\)](#).

Tipo: matriz de strings

Membros da matriz: número máximo de 15 itens.

Valores válidos: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | nodejs10.x | nodejs12.x | nodejs14.x | java8 | java8.al2 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | python3.9 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | dotnetcore3.1 | nodejs4.3-edge | go1.x | ruby2.5 | ruby2.7 | provided | provided.al2

Exigido: Não

[Content \(p. 915\)](#)

O arquivamento de camadas de função.

Tipo: objeto [LayerVersionContentInput \(p. 1028\)](#)

Obrigatório: sim

[Description \(p. 915\)](#)

A descrição da versão.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

Exigido: Não

[LicenseInfo \(p. 915\)](#)

A licença de software da camada. Pode ser qualquer um dos seguintes:

- Um [identificador de licença SPDX](#). Por exemplo, MIT.
- O URL de uma licença hospedada na Internet. Por exemplo, <https://opensource.org/licenses/MIT>.
- O texto completo da licença.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 512.

Exigido: Não

Sintaxe da resposta

```
HTTP/1.1 201
Content-type: application/json

{
  "CompatibleRuntimes": [ "string" ],
  "Content": {
    "CodeSha256": "string",
    "CodeSize": number,
    "Location": "string",
    "SigningJobArn": "string",
    "SigningProfileVersionArn": "string"
  },
  "CreatedDate": "string",
  "Description": "string",
  "LayerArn": "string",
  "LayerVersionArn": "string",
  "LicenseInfo": "string",
  "Version": number
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço retornará uma resposta HTTP 201.

Os seguintes dados são retornados no formato JSON pelo serviço.

[CompatibleRuntimes \(p. 916\)](#)

Os tempos de execução compatíveis da camada.

Tipo: matriz de strings

Membros da matriz: número máximo de 15 itens.

Valores válidos: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | nodejs10.x | nodejs12.x | nodejs14.x | java8 | java8.al2 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | python3.9 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | dotnetcore3.1 | nodejs4.3-edge | go1.x | ruby2.5 | ruby2.7 | provided | provided.al2

[Content \(p. 916\)](#)

Detalhes sobre a versão da camada.

Tipo: objeto [LayerVersionContentOutput \(p. 1029\)](#)

[CreatedDate \(p. 916\)](#)

A data em que a versão da camada foi criada, em [formato ISO-8601](#) (AAAA-MM-DDThh:mm:ss.sTZD).

Tipo: sequência

[Description \(p. 916\)](#)

A descrição da versão.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

[LayerArn \(p. 916\)](#)

O ARN da camada.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+

[LayerVersionArn \(p. 916\)](#)

O ARN da versão da camada.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

[LicenseInfo \(p. 916\)](#)

A licença de software da camada.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 512.

[Version \(p. 916\)](#)

O número da versão.

Type: longo

Errors

CodeStorageExceededException

Você excedeu seu tamanho máximo total de código por conta. [Saiba mais](#)

Código de status HTTP: 400

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

PublishVersion

Cria uma [versão](#) do código atual e a configuração de uma função. Use versões para criar um snapshot de seu código de função e a configuração não será alterada.

O AWS Lambda não publica uma versão se a configuração e o código da função não tiverem sido alterados desde a última versão. Use [UpdateFunctionCode \(p. 965\)](#) ou [UpdateFunctionConfiguration \(p. 974\)](#) para atualizar a função antes de publicar uma versão.

Os clientes podem invocar versões diretamente ou com um alias. Para criar um alias, use [CreateAlias \(p. 779\)](#).

Sintaxe da solicitação

```
POST /2015-03-31/functions/FunctionName/versions HTTP/1.1
Content-type: application/json

{
  "CodeSha256": "string",
  "Description": "string",
  "RevisionId": "string"
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName \(p. 919\)](#)

O nome da função Lambda.

Formatos de nome

- Nome da função - MyFunction.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- ARN parcial - 123456789012:function:MyFunction.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[CodeSha256 \(p. 919\)](#)

Publique uma versão somente se o valor de hash corresponder ao valor especificado. Use essa opção para evitar a publicação de uma versão se o código da função tiver sido alterado desde a última atualização. Você pode obter o hash para a versão que você enviou por upload a partir da saída de [UpdateFunctionCode \(p. 965\)](#).

Tipo: string

Exigido: Não

[Description \(p. 919\)](#)

Uma descrição da versão para substituir a descrição na configuração da função.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

Exigido: Não

[RevisionId \(p. 919\)](#)

Atualize a função somente se o ID da revisão corresponder ao ID especificado. Use essa opção para evitar a publicação de uma versão se a configuração da função tiver sido alterada desde a última atualização.

Tipo: string

Exigido: Não

Sintaxe da resposta

```
HTTP/1.1 201
Content-type: application/json

{
    "CodeSha256": "string",
    "CodeSize": number,
    "DeadLetterConfig": {
        "TargetArn": "string"
    },
    "Description": "string",
    "Environment": {
        "Error": {
            "ErrorCode": "string",
            "Message": "string"
        },
        "Variables": {
            "string" : "string"
        }
    },
    "FileSystemConfigs": [
        {
            "Arn": "string",
            "LocalMountPath": "string"
        }
    ],
    "FunctionArn": "string",
    "FunctionName": "string",
    "Handler": "string",
    "ImageConfigResponse": {
        "Error": {
            "ErrorCode": "string",
            "Message": "string"
        },
        "ImageConfig": {
            "Command": [ "string" ],
            "EntryPoint": [ "string" ],
            "WorkingDirectory": "string"
        }
    }
}
```

```
        },
        "KMSKeyArn": "string",
        "LastModified": "string",
        "LastUpdateStatus": "string",
        "LastUpdateStatusReason": "string",
        "LastUpdateStatusReasonCode": "string",
        "Layers": [
            {
                "Arn": "string",
                "CodeSize": number,
                "SigningJobArn": "string",
                "SigningProfileVersionArn": "string"
            }
        ],
        "MasterArn": "string",
        "MemorySize": number,
        "PackageType": "string",
        "RevisionId": "string",
        "Role": "string",
        "Runtime": "string",
        "SigningJobArn": "string",
        "SigningProfileVersionArn": "string",
        "State": "string",
        "StateReason": "string",
        "StateReasonCode": "string",
        "Timeout": number,
        "TracingConfig": {
            "Mode": "string"
        },
        "Version": "string",
        "VpcConfig": {
            "SecurityGroupIds": [ "string" ],
            "SubnetIds": [ "string" ],
            "VpcId": "string"
        }
    }
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço retornará uma resposta HTTP 201.

Os seguintes dados são retornados no formato JSON pelo serviço.

[CodeSha256 \(p. 920\)](#)

O hash SHA256 do pacote de implantação da função.

Tipo: sequência

[CodeSize \(p. 920\)](#)

O tamanho do pacote de implantação da função em bytes.

Type: longo

[DeadLetterConfig \(p. 920\)](#)

A fila de mensagens mortas da função.

Tipo: objeto [DeadLetterConfig \(p. 1001\)](#)

[Description \(p. 920\)](#)

A descrição da função.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.
[Environment \(p. 920\)](#)

As [variáveis de ambiente](#) da função.

Tipo: objeto [EnvironmentResponse \(p. 1005\)](#)
[FileSystemConfigs \(p. 920\)](#)

Configurações de conexão para um [sistema de arquivos do Amazon EFS](#).

Tipo: matriz de [FileSystemConfig \(p. 1011\)](#)objetos

Membros da matriz: número máximo de 1 item.
[FunctionArn \(p. 920\)](#)

O nome do recurso da Amazon (ARN) da função.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`
[FunctionName \(p. 920\)](#)

Nome da função.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 170.
Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?)`
[Handler \(p. 920\)](#)

A função que o Lambda chama para começar a executar a função.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 128.

Pattern: `[^\s]+`
[ImageConfigResponse \(p. 920\)](#)

Os valores de configuração da imagem da função.

Tipo: objeto [ImageConfigResponse \(p. 1025\)](#)
[KMSKeyArn \(p. 920\)](#)

A chave KMS usada para criptografar as variáveis do ambiente da função. Esta chave é retornada somente se você tiver configurado uma CMK gerenciada pelo cliente.

Tipo: sequência
Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-_\.]+\.:*)|()`
[LastModified \(p. 920\)](#)

A data e a hora em que a função foi atualizada, no [formato ISO-8601](#) (AAAA-MM-DDThh:mm:ss.sTZD).

Tipo: sequência

[LastUpdateStatus \(p. 920\)](#)

O status da última atualização que foi executada na função. Ele é definido pela primeira vez como `Successful` após a conclusão da criação da função.

Tipo: sequência

Valores válidos: `Successful` | `Failed` | `InProgress`

[LastUpdateStatusReason \(p. 920\)](#)

O motivo pelo qual foi realizada a última atualização na função.

Tipo: sequência

[LastUpdateStatusReasonCode \(p. 920\)](#)

O código do motivo pelo qual foi realizada a última atualização na função.

Tipo: sequência

Valores válidos: `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup` | `ImageDeleted` | `ImageAccessDenied` | `InvalidImage`

[Layers \(p. 920\)](#)

As [camadas](#) da função.

Tipo: matriz de [Layer \(p. 1026\)](#) objetos

[MasterArn \(p. 920\)](#)

Para funções do Lambda@Edge, o ARN da função mestre.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*):lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\${LATEST}|[a-zA-Z0-9-_+]))?`

[MemorySize \(p. 920\)](#)

A quantidade de memória disponível para a função no tempo de execução.

Type: inteiro

Intervalo válido: valor mínimo de 128. Valor máximo de 10240.

[PackageType \(p. 920\)](#)

O tipo de pacote de implantação. Defina como `Image` para imagem de contêiner e defina `Zip` para arquivo de documento `.zip`.

Tipo: sequência

Valores válidos: `zip` | `Image`

[RevisionId \(p. 920\)](#)

A última revisão atualizada da função ou do alias.

Tipo: sequência

[Role \(p. 920\)](#)

A função de execução da função.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@/-/_]+
[Runtime \(p. 920\)](#)

O ambiente de execução da função do Lambda.

Tipo: sequência

Valores válidos: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | nodejs10.x | nodejs12.x | nodejs14.x | java8 | java8.al2 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | python3.9 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | dotnetcore3.1 | nodejs4.3-edge | go1.x | ruby2.5 | ruby2.7 | provided | provided.al2

[SigningJobArn \(p. 920\)](#)

O ARN do trabalho de assinatura.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*)([a-zA-Z0-9-]+)([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12})?:(.*)

[SigningProfileVersionArn \(p. 920\)](#)

O ARN da versão do perfil de assinatura.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*)([a-zA-Z0-9-]+)([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12})?:(.*)

[State \(p. 920\)](#)

O estado atual da função. Quando o estado é `Inactive`, você pode invocar a função para reativá-la.

Tipo: sequência

Valores válidos: Pending | Active | Inactive | Failed

[StateReason \(p. 920\)](#)

O motivo para o estado atual da função.

Tipo: sequência

[StateReasonCode \(p. 920\)](#)

O código do motivo para o estado atual da função. Quando o código for `Creating`, não será possível invocar ou modificar a função.

Tipo: sequência

Valores válidos: Idle | Creating | Restoring | EniLimitExceeded | InsufficientRolePermissions | InvalidConfiguration | InternalError | SubnetOutOfIPAddresses | InvalidSubnet | InvalidSecurityGroup | ImageDeleted | ImageAccessDenied | InvalidImage

[Timeout \(p. 920\)](#)

A quantidade de tempo, em segundos, que o Lambda permite que uma função seja executada antes de encerrá-la.

Type: inteiro

Faixa válida: valor mínimo de 1.

[TracingConfig \(p. 920\)](#)

A configuração de rastreamento de AWS X-Ray da função.

Tipo: objeto [TracingConfigResponse \(p. 1040\)](#)

[Version \(p. 920\)](#)

A versão da função do Lambda.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Pattern: (\\$LATEST | [0-9]+)

[VpcConfig \(p. 920\)](#)

A configuração de rede da função.

Tipo: objeto [VpcConfigResponse \(p. 1042\)](#)

Errors

[CodeStorageExceededException](#)

Você excedeu seu tamanho máximo total de código por conta. [Saiba mais](#)

Código de status HTTP: 400

[InvalidParameterValueException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[PreconditionFailedException](#)

O RevisionId fornecido não corresponde ao RevisionId mais recente da função ou do alias do Lambda. Chame a API `GetFunction` ou `GetAlias` para recuperar o RevisionId mais recente para o seu recurso.

Código de status HTTP: 412

[ResourceConflictException](#)

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

[ResourceNotFoundException](#)

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

[TooManyRequestsException](#)

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

PutFunctionCodeSigningConfig

Atualize a configuração de assinatura de código da função especificada. As alterações na configuração de assinatura de código entrarão em vigor na próxima vez que um usuário tentar implantar um pacote de código para a função.

Sintaxe da solicitação

```
PUT /2020-06-30/functions/FunctionName/code-signing-config HTTP/1.1
Content-type: application/json

{
    "CodeSigningConfigArn": "string"
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 927)

O nome da função Lambda.

Formatos de nome

- Nome da função - `MyFunction`.
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- ARN parcial - `123456789012:function:MyFunction`.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Obrigatório: sim

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[CodeSigningConfigArn](#) (p. 927)

O nome do recurso da Amazon (ARN) da configuração de assinatura de código.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 200.

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}((-gov)|(-iso(b?)))?-[a-z]+\d{1}:\d{12}:code-signing-config:csc-[a-zA-Z0-9]{17}`

Obrigatório: sim

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
    "CodeSigningConfigArn": "string",
    "FunctionName": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[CodeSigningConfigArn](#) (p. 928)

O nome do recurso da Amazon (ARN) da configuração de assinatura de código.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 200.

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}((-gov)|(-iso(b?)))?- [a-z]+-\d{1}:\d{12}:code-signing-config:csc-[a-zA-Z0-9]{17}

[FunctionName](#) (p. 928)

O nome da função Lambda.

Formatos de nome

- Nome da função - MyFunction.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- ARN parcial - 123456789012:function:MyFunction.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Errors

[CodeSigningConfigNotFoundException](#)

A configuração de assinatura de código especificada não existe.

Código de status HTTP: 404

[InvalidParameterValueException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceConflictException

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

PutFunctionConcurrency

Define o número máximo de execuções simultâneas para uma função e reserva capacidade para esse nível de simultaneidade.

As configurações de simultaneidade aplicam-se à função como um todo, incluindo todas as versões publicadas e a versão não publicada. Reservar simultaneidade garante que sua função tenha capacidade para processar o número especificado de eventos simultaneamente e impede que ela seja dimensionada além desse nível. Use [GetFunction \(p. 842\)](#) para ver a configuração atual de uma função.

Use [GetAccountSettings \(p. 830\)](#) para ver o limite de simultaneidade regional. Você pode reservar simultaneidade para quantas funções quiser, desde que você deixe pelo menos 100 execuções simultâneas sem reservas para funções que não estão configuradas com um limite por função. Para obter mais informações, consulte [Gerenciamento de simultaneidade](#).

Sintaxe da solicitação

```
PUT /2017-10-31/functions/FunctionName/concurrency HTTP/1.1
Content-type: application/json

{
    "ReservedConcurrentExecutions": number
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName \(p. 930\)](#)

O nome da função Lambda.

Formatos de nome

- Nome da função - my-function.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- ARN parcial - 123456789012:function:my-function.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[ReservedConcurrentExecutions \(p. 930\)](#)

O número de execuções simultâneas a serem reservadas para a função.

Type: inteiro

Faixa válida: valor mínimo de 0.

Obrigatório: sim

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
    "ReservedConcurrentExecutions": number
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[ReservedConcurrentExecutions \(p. 931\)](#)

O número de execuções simultâneas que estão reservadas para essa função. Para obter mais informações, consulte [Gerenciamento de simultaneidade](#).

Type: inteiro

Faixa válida: valor mínimo de 0.

Errors

[InvalidParameterValueException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ResourceConflictException](#)

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

[ResourceNotFoundException](#)

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

[TooManyRequestsException](#)

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

PutFunctionEventInvokeConfig

Configura opções para [invocação assíncrona](#) em uma função, uma versão ou um alias. Se já existe uma configuração para uma função, uma versão ou um alias, essa operação a substituirá. Se você excluir quaisquer configurações, elas serão removidas. Para definir uma opção sem afetar as configurações existentes de outras opções, use [UpdateFunctionEventInvokeConfig](#) (p. 985).

Por padrão, o Lambda tentará novamente uma invocação assíncrona duas vezes se a função retornar um erro. Ele retém eventos em uma fila por até seis horas. Quando um evento falha em todas as tentativas de processamento ou permanece na fila de invocação assíncrona por muito tempo, o Lambda o descarta. Configure uma fila de mensagens mortas com [UpdateFunctionConfiguration](#) (p. 974) para reter eventos descartados.

Para enviar um registro de chamada para uma fila, um tópico, uma função ou um barramento de eventos, especifique um [destino](#). Você pode configurar destinos separados para chamadas bem-sucedidas (on-success) e eventos que falharam em todas as tentativas de processamento (on-failure). Você pode configurar destinos além de uma fila de mensagens mortas ou no lugar dela.

Sintaxe da solicitação

```
PUT /2019-09-25/functions/FunctionName/event-invoke-config?Qualifier=Qualifier HTTP/1.1
Content-type: application/json

{
  "DestinationConfig": {
    "OnFailure": {
      "Destination": "string"
    },
    "OnSuccess": {
      "Destination": "string"
    }
  },
  "MaximumEventAgeInSeconds": number,
  "MaximumRetryAttempts": number
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

FunctionName (p. 933)

O nome da função, versão ou alias do Lambda.

Formatos de nome

- Function name - `my-function` (somente nome), `my-function:v1` (com alias).
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- ARN parcial - `123456789012:function:my-function`.

Você pode anexar um número de versão ou alias a qualquer um dos formatos. A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (`arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}):\?`
`(\d{12}:\)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Obrigatório: sim

[Qualifier \(p. 933\)](#)

Um número de versão ou nome de alias.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: (| [a-zA-Z0-9\$_-]+)

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[DestinationConfig \(p. 933\)](#)

Um destino para eventos depois que eles foram enviados a uma função para processamento.

Destinations

- Function (Função) – o nome de recurso da Amazon (ARN) da função do Lambda.
- Queue (Fila) – o ARN de uma fila do SQS.
- Topic (Tópico) – o ARN de um tópico do SNS.
- Event Bus (Barramento de eventos) – o ARN de um barramento de eventos do Amazon EventBridge.

Tipo: objeto [DestinationConfig \(p. 1002\)](#)

Exigido: Não

[MaximumEventAgeInSeconds \(p. 933\)](#)

A idade máxima de uma solicitação que o Lambda envia a uma função para processamento.

Type: inteiro

Intervalo válido: valor mínimo de 60. Valor máximo de 21600.

Exigido: Não

[MaximumRetryAttempts \(p. 933\)](#)

O número máximo de vezes para tentar novamente quando a função retorna um erro.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 2.

Exigido: Não

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
  "DestinationConfig": {
    "OnFailure": {
```

```
        "Destination": "string"
    },
    "OnSuccess": {
        "Destination": "string"
    }
},
"FunctionArn": "string",
"LastModified": number,
"MaximumEventAgeInSeconds": number,
"MaximumRetryAttempts": number
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[DestinationConfig \(p. 934\)](#)

Um destino para eventos depois que eles foram enviados a uma função para processamento.

Destinations

- Function (Função) – o nome de recurso da Amazon (ARN) da função do Lambda.
- Queue (Fila) – o ARN de uma fila do SQS.
- Topic (Tópico) – o ARN de um tópico do SNS.
- Event Bus (Barramento de eventos) – o ARN de um barramento de eventos do Amazon EventBridge.

Tipo: objeto [DestinationConfig \(p. 1002\)](#)

[FunctionArn \(p. 934\)](#)

O nome de recurso da Amazon (ARN) da função.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?

[LastModified \(p. 934\)](#)

A data e a hora em que a configuração foi atualizada pela última vez, em segundos no tempo do Unix.

Type: timestamp

[MaximumEventAgeInSeconds \(p. 934\)](#)

A idade máxima de uma solicitação que o Lambda envia a uma função para processamento.

Type: inteiro

Intervalo válido: valor mínimo de 60. Valor máximo de 21600.

[MaximumRetryAttempts \(p. 934\)](#)

O número máximo de vezes para tentar novamente quando a função retorna um erro.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 2.

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceConflictException

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

PutProvisionedConcurrencyConfig

Adiciona uma configuração de simultaneidade provisionada ao alias ou à versão de uma função.

Sintaxe da solicitação

```
PUT /2019-09-30/functions/FunctionName/provisioned-concurrency?Qualifier=Qualifier HTTP/1.1
Content-type: application/json

{
    "ProvisionedConcurrentExecutions": number
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName \(p. 937\)](#)

O nome da função Lambda.

Formatos de nome

- Nome da função - my-function.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- ARN parcial - 123456789012:function:my-function.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[Qualifier \(p. 937\)](#)

O número de versão ou nome de alias.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Obrigatório: sim

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[ProvisionedConcurrentExecutions \(p. 937\)](#)

A quantidade de simultaneidade provisionada a ser alocada para a versão ou o alias.

Type: inteiro

Faixa válida: valor mínimo de 1.

Obrigatório: sim

Sintaxe da resposta

```
HTTP/1.1 202
Content-type: application/json

{
    "AllocatedProvisionedConcurrentExecutions": number,
    "AvailableProvisionedConcurrentExecutions": number,
    "LastModified": "string",
    "RequestedProvisionedConcurrentExecutions": number,
    "Status": "string",
    "StatusReason": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço retornará uma resposta HTTP 202.

Os seguintes dados são retornados no formato JSON pelo serviço.

[AllocatedProvisionedConcurrentExecutions \(p. 938\)](#)

A quantidade de simultaneidade provisionada alocada.

Type: inteiro

Faixa válida: valor mínimo de 0.

[AvailableProvisionedConcurrentExecutions \(p. 938\)](#)

A quantidade de simultaneidade provisionada disponível.

Type: inteiro

Faixa válida: valor mínimo de 0.

[LastModified \(p. 938\)](#)

A data e hora em que um usuário atualizou a configuração pela última vez, no [formato ISO 8601](#).

Tipo: sequência

[RequestedProvisionedConcurrentExecutions \(p. 938\)](#)

A quantidade de simultaneidade provisionada solicitada.

Type: inteiro

Faixa válida: valor mínimo de 1.

[Status \(p. 938\)](#)

O status do processo de alocação.

Tipo: sequência

Valores válidos: IN_PROGRESS | READY | FAILED

[StatusReason \(p. 938\)](#)

Para alocações com falha, o motivo pelo qual a simultaneidade provisionada não pôde ser alocada.

Tipo: sequência

Errors

InvalidArgumentException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceConflictException

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

RemoveLayerVersionPermission

Remove uma instrução da política de permissões de uma versão de uma camada do AWS Lambda. Para obter mais informações, consulte [AddLayerVersionPermission \(p. 771\)](#).

Sintaxe da solicitação

```
DELETE /2018-10-31/layers/LayerName/versions/VersionNumber/policy/StatementId?  
RevisionId=RevisionId HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[LayerName \(p. 940\)](#)

O nome ou o nome de recurso da Amazon (ARN) da camada.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (`arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\:\d{12}:layer:[a-zA-Z0-9-_+]|[a-zA-Z0-9-_]+`)

Obrigatório: sim

[RevisionId \(p. 940\)](#)

Atualize a política somente se o ID da revisão corresponder ao ID especificado. Use essa opção para evitar a modificação de uma política que foi alterada desde a última leitura.

[StatementId \(p. 940\)](#)

O identificador que foi especificado quando a instrução foi adicionada.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 100.

Pattern: (`[a-zA-Z0-9-_+]`)

Obrigatório: sim

[VersionNumber \(p. 940\)](#)

O número da versão.

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 204
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 204 com um corpo HTTP vazio.

Errors

InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

PreconditionFailedException

O RevisionId fornecido não corresponde ao RevisionId mais recente da função ou do alias do Lambda. Chame a API `GetFunction` ou `GetAlias` para recuperar o RevisionId mais recente para o seu recurso.

Código de status HTTP: 412

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

RemovePermission

Revoga a permissão de uso da função de um serviço da AWS ou de outra conta. É possível obter o ID da instrução da saída de [GetPolicy](#) (p. 869).

Sintaxe da solicitação

```
DELETE /2015-03-31/functions/FunctionName/policy/StatementId?  
Qualifier=Qualifier&RevisionId=RevisionId HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 942)

O nome da função, versão ou alias do Lambda.

Formatos de nome

- Function name - my-function (somente nome), my-function:v1 (com alias).
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- ARN parcial - 123456789012:function:my-function.

Você pode anexar um número de versão ou alias a qualquer um dos formatos. A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*):lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[Qualifier](#) (p. 942)

Especifique uma versão ou alias para remover permissões de uma versão publicada da função.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: ([a-zA-Z0-9\$-_]+)

[RevisionId](#) (p. 942)

Atualize a política somente se o ID da revisão corresponder ao ID especificado. Use essa opção para evitar a modificação de uma política que foi alterada desde a última leitura.

[StatementId](#) (p. 942)

ID da instrução da permissão a ser removida.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 100.

Pattern: ([a-zA-Z0-9-_]+)

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 204
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 204 com um corpo HTTP vazio.

Errors

InvalidArgumentException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

PreconditionFailedException

O RevisionId fornecido não corresponde ao RevisionId mais recente da função ou do alias do Lambda. Chame a API GetFunction ou GetAlias para recuperar o RevisionId mais recente para o seu recurso.

Código de status HTTP: 412

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)

- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

TagResource

Adiciona [etiquetas](#) a uma função.

Sintaxe da solicitação

```
POST /2017-03-31/tags/ARN HTTP/1.1
Content-type: application/json

{
    "Tags": {
        "string" : "string"
    }
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[ARN](#) (p. 945)

O nome do recurso da Amazon (ARN) da função.

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\${LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[Tags](#) (p. 945)

Uma lista de tags para aplicar à função.

Tipo:: mapa de string para string

Obrigatório: sim

Sintaxe da resposta

```
HTTP/1.1 204
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 204 com um corpo HTTP vazio.

Errors

[InvalidParameterValueException](#)

Um dos parâmetros na solicitação é inválido.

- Código de status HTTP: 400
`ResourceConflictException`
 - O recurso já existe ou outra operação está em andamento.
- Código de status HTTP: 409
`ResourceNotFoundException`
 - O recurso especificado na solicitação não existe.
- Código de status HTTP: 404
`ServiceException`
 - O serviço AWS Lambda encontrou um erro interno.
- Código de status HTTP: 500
`TooManyRequestsException`
 - O limite de taxa de transferência da solicitação foi excedido.
- Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

UntagResource

Remove [etiquetas](#) de uma função.

Sintaxe da solicitação

```
DELETE /2017-03-31/tags/ARN?tagKeys=TagKeys HTTP/1.1
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[ARN](#) (p. 947)

O nome do recurso da Amazon (ARN) da função.

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[TagKeys](#) (p. 947)

Uma lista de chaves de etiqueta a serem removidas da função.

Obrigatório: sim

Corpo da solicitação

Essa solicitação não tem o corpo da solicitação.

Sintaxe da resposta

```
HTTP/1.1 204
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 204 com um corpo HTTP vazio.

Errors

[InvalidParameterValueException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ResourceConflictException](#)

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

[ResourceNotFoundException](#)

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

UpdateAlias

Atualiza a configuração do [alias](#) de uma função do Lambda.

Sintaxe da solicitação

```
PUT /2015-03-31/functions/FunctionName/aliases/Name HTTP/1.1
Content-type: application/json

{
    "Description": "string",
    "FunctionVersion": "string",
    "RevisionId": "string",
    "RoutingConfig": {
        "AdditionalVersionWeights": {
            "string": number
        }
    }
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 949)

O nome da função Lambda.

Formatos de nome

- Nome da função - `MyFunction`.
- ARN da função - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- ARN parcial - `123456789012:function:MyFunction`.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (`arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:[)?(\d{12}:[)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`)

Obrigatório: sim

[Name](#) (p. 949)

O nome do alias.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: (?![0-9]+\$)([a-zA-Z0-9-_]+)

Obrigatório: sim

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[Description \(p. 949\)](#)

Uma descrição do alias.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

Exigido: Não

[FunctionVersion \(p. 949\)](#)

A versão da função que o alias invoca.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Pattern: (\\$LATEST | [0-9]+)

Exigido: Não

[RevisionId \(p. 949\)](#)

Atualize o alias somente se o ID da revisão corresponder ao ID especificado. Use essa opção para evitar a modificação de um alias que foi alterado desde a última leitura.

Tipo: string

Exigido: Não

[RoutingConfig \(p. 949\)](#)

A configuração de roteamento do alias.

Tipo: objeto [AliasRoutingConfiguration \(p. 995\)](#)

Exigido: Não

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
    "AliasArn": "string",
    "Description": "string",
    "FunctionVersion": "string",
    "Name": "string",
    "RevisionId": "string",
    "RoutingConfig": {
        "AdditionalVersionWeights": {
            "string" : number
        }
    }
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[AliasArn \(p. 950\)](#)

O nome do recurso da Amazon (ARN) do alarme do alias.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+\-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[Description \(p. 950\)](#)

Uma descrição do alias.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

[FunctionVersion \(p. 950\)](#)

A versão da função que o alias invoca.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Pattern: (\\$LATEST|[0-9]+)

[Name \(p. 950\)](#)

O nome do alias.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: (?![0-9]+\\$)([a-zA-Z0-9-_]+)

[RevisionId \(p. 950\)](#)

Um identificador exclusivo que muda ao atualizar o alias.

Tipo: sequência

[RoutingConfig \(p. 950\)](#)

A [configuração de roteamento](#) do alias.

Tipo: objeto [AliasRoutingConfiguration \(p. 995\)](#)

Errors

[InvalidParameterValueException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[PreconditionFailedException](#)

O RevisionId fornecido não corresponde ao RevisionId mais recente da função ou do alias do Lambda. Chame a API `GetFunction` ou `GetAlias` para recuperar o RevisionId mais recente para o seu recurso.

Código de status HTTP: 412

ResourceConflictException

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

UpdateCodeSigningConfig

Atualize uma configuração de assinatura de código. As alterações na configuração de assinatura de código entrarão em vigor na próxima vez que um usuário tentar implantar um pacote de código para a função.

Sintaxe da solicitação

```
PUT /2020-04-22/code-signing-configs/CodeSigningConfigArn HTTP/1.1
Content-type: application/json

{
  "AllowedPublishers": {
    "SigningProfileVersionArns": [ "string" ]
  },
  "CodeSigningPolicies": {
    "UntrustedArtifactOnDeployment": "string"
  },
  "Description": "string"
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[CodeSigningConfigArn](#) (p. 953)

O nome do recurso da Amazon (ARN) da configuração de assinatura de código.

Restrições de tamanho: tamanho máximo de 200.

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}((-gov)|(-iso(b?)))?-([a-z]+-\d{1}):\d{12}:code-signing-config:csc-[a-z0-9]{17}

Obrigatório: sim

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[AllowedPublishers](#) (p. 953)

Assinatura de perfis para esta configuração de assinatura de código.

Tipo: objeto [AllowedPublishers](#) (p. 996)

Exigido: Não

[CodeSigningPolicies](#) (p. 953)

A política de assinatura de código.

Tipo: objeto [CodeSigningPolicies](#) (p. 999)

Exigido: Não

[Description](#) (p. 953)

Nome descritivo para essa configuração de assinatura de código.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

Exigido: Não

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
    "CodeSigningConfig": {
        "AllowedPublishers": {
            "SigningProfileVersionArns": [ "string" ]
        },
        "CodeSigningConfigArn": "string",
        "CodeSigningConfigId": "string",
        "CodeSigningPolicies": {
            "UntrustedArtifactOnDeployment": "string"
        },
        "Description": "string",
        "LastModified": "string"
    }
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[CodeSigningConfig \(p. 954\)](#)

A configuração de assinatura de código

Tipo: objeto [CodeSigningConfig \(p. 997\)](#)

Errors

[InvalidOperationException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ResourceNotFoundException](#)

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- AWS Interface da linha de comando
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK para JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UpdateEventSourceMapping

Atualiza um mapeamento de origem do evento. Não é possível alterar a função que AWS Lambda invoca nem pausar a invocação e retomar mais tarde a partir do mesmo local.

As seguintes opções de tratamento de erros estão disponíveis apenas para fontes de stream (DynamoDB e Kinesis):

- `BisectBatchOnFunctionError`: se a função retornar um erro, divide o lote em dois e tente novamente.
- `DestinationConfig`: envie registros descartados para uma fila do Amazon SQS ou para um tópico do Amazon SNS.
- `MaximumRecordAgeInSeconds`: descarta registros mais antigos que a idade especificada. O valor padrão é infinito (-1). Quando definido como infinito (-1), são feitas novas tentativas para os registros com falha até o registro expirar
- `MaximumRetryAttempts`: descarta registros após o número especificado de tentativas. O valor padrão é infinito (-1). Quando definido como infinito (-1), são feitas novas tentativas para os registros com falha até o registro expirar.
- `ParallelizationFactor`: processe vários lotes de cada fragmento simultaneamente.

Sintaxe da solicitação

```
PUT /2015-03-31/event-source-mappings/UUID HTTP/1.1
Content-type: application/json

{
    "BatchSize": number,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "Enabled": boolean,
    "FunctionName": "string",
    "FunctionResponseTypes": [ "string" ],
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "SourceAccessConfigurations": [
        {
            "Type": "string",
            "URI": "string"
        }
    ],
    "TumblingWindowInSeconds": number
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[UUID \(p. 956\)](#)

O identificador do mapeamento da origem do evento.

Obrigatório: sim

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[BatchSize \(p. 956\)](#)

O número máximo de registros em cada batch que o Lambda extrai da sua transmissão ou fila e envia para sua função. O Lambda transmite todos os registros no batch para a função em uma única chamada até o limite de carga útil para invocação síncrona (6 MB).

- Amazon Kinesis - padrão 100. No máximo 10.000.
- Amazon DynamoDB Streams - padrão 100. No máximo 1.000.
- Amazon Simple Queue Service - padrão 10. Para filas padrão, o máximo é 10.000. Para filas FIFO, o máximo é 10.
- Amazon Managed Streaming for Apache Kafka – padrão 100. No máximo 10.000.
- Apache Kafka autogerenciado - padrão 100. No máximo 10.000.

Type: inteiro

Faixa válida: valor mínimo de 1. Valor máximo de 10000.

Exigido: Não

[BisectBatchOnFunctionError \(p. 956\)](#)

(Somente streams) Se a função retornar um erro, divida o lote em dois e tente novamente.

Type: booliano

Exigido: Não

[DestinationConfig \(p. 956\)](#)

(Somente streams) Uma fila do Amazon SQS ou um destino de tópico do Amazon SNS para registros descartados.

Tipo: objeto [DestinationConfig \(p. 1002\)](#)

Exigido: Não

[Enabled \(p. 956\)](#)

Quando verdadeiro, o mapeamento da fonte do evento estará ativo. Quando falso, o Lambda pausará a sondagem e a invocação.

Padrão: True

Type: booliano

Exigido: Não

[FunctionName \(p. 956\)](#)

O nome da função Lambda.

Formatos de nome

- Nome da função - MyFunction.

- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- ARN da versão ou alias - arn:aws:lambda:us-west-2:123456789012:function:MyFunction:PROD.
- ARN parcial - 123456789012:function:MyFunction.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Exigido: Não

[FunctionResponseTypes \(p. 956\)](#)

(Somente transmissões) Uma lista de enums de tipos de resposta atuais aplicados ao mapeamento de fontes de eventos.

Tipo: matriz de strings

Membros da matriz: número mínimo de 0 itens. Número máximo de 1 item.

Valores válidos: ReportBatchItemFailures

Exigido: Não

[MaximumBatchingWindowInSeconds \(p. 956\)](#)

(Transmissões e filas padrão do Amazon SQS) O tempo máximo usado pelo Lambda, em segundos, para reunir os registros antes de invocar a função.

Padrão: 0

Configuração relacionada: quando você define BatchSize como um valor maior que 10, deve definir MaximumBatchingWindowInSeconds como pelo menos 1.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 300.

Exigido: Não

[MaximumRecordAgeInSeconds \(p. 956\)](#)

(Somente streams) Descarta registros mais antigos que a idade especificada. O valor padrão é infinito (-1).

Type: inteiro

Intervalo válido: valor mínimo de -1. Valor máximo de 604800.

Exigido: Não

[MaximumRetryAttempts \(p. 956\)](#)

(Somente streams) Descarta registros após o número especificado de novas tentativas. O valor padrão é infinito (-1). Quando definido como infinito (-1), serão feitas novas tentativas para os registros com falha até o registro expirar.

Type: inteiro

Intervalo válido: valor mínimo de -1. Valor máximo de 10000.

Exigido: Não

[ParallelizationFactor \(p. 956\)](#)

(Somente streams) O número de lotes a serem processados de cada fragmento simultaneamente.

Type: inteiro

Faixa válida: valor mínimo de 1. Valor máximo de 10.

Exigido: Não

[SourceAccessConfigurations \(p. 956\)](#)

Uma matriz de protocolos de autenticação ou componentes da VPC necessária para proteger a origem do evento.

Tipo: matriz de [SourceAccessConfiguration \(p. 1037\)](#) objetos

Membros da matriz: número mínimo de 0 itens. Número máximo de 22 itens.

Exigido: Não

[TumblingWindowInSeconds \(p. 956\)](#)

(Somente streams) A duração, em segundos, de uma janela de processamento. O intervalo é entre 1 segundo até 900 minutos.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 900.

Exigido: Não

Sintaxe da resposta

```
HTTP/1.1 202
Content-type: application/json

{
    "BatchSize": number,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "FunctionResponseTypes": [ "string" ],
    "LastModified": number,
    "LastProcessingResult": "string",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "Queues": [ "string" ],
    "SelfManagedEventSource": {
        "ARN": "string",
        "EventSourceArn": "string",
        "FunctionArn": "string",
        "FunctionResponseTypes": [ "string" ],
        "LastModified": number,
        "LastProcessingResult": "string",
        "MaximumBatchingWindowInSeconds": number,
        "MaximumRecordAgeInSeconds": number,
        "MaximumRetryAttempts": number,
        "ParallelizationFactor": number
    }
}
```

```
    "Endpoints": {
        "string" : [ "string" ]
    }
},
"SourceAccessConfigurations": [
{
    "Type": "string",
    "URI": "string"
}
],
"StartingPosition": "string",
"StartingPositionTimestamp": number,
"State": "string",
"StateTransitionReason": "string",
"Topics": [ "string" ],
"TumblingWindowInSeconds": number,
"UUID": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço retornará uma resposta HTTP 202.

Os seguintes dados são retornados no formato JSON pelo serviço.

[BatchSize \(p. 959\)](#)

O número máximo de registros em cada batch que o Lambda extrai da sua transmissão ou fila e envia para sua função. O Lambda transmite todos os registros no batch para a função em uma única chamada até o limite de carga útil para invocação síncrona (6 MB).

Valor padrão: varia de acordo com o serviço. Para o Amazon SQS, o padrão é 10. Para todos os outros serviços, o padrão é 100.

Configuração relacionada: quando você define `BatchSize` como um valor maior que 10, deve definir `MaximumBatchingWindowInSeconds` como pelo menos 1.

Type: inteiro

Faixa válida: valor mínimo de 1. Valor máximo de 10000.

[BisectBatchOnFunctionError \(p. 959\)](#)

(Somente streams) Se a função retornar um erro, divida o lote em dois e tente novamente. O valor padrão é falso.

Type: booliano

[DestinationConfig \(p. 959\)](#)

(Somente streams) Uma fila do Amazon SQS ou um destino de tópico do Amazon SNS para registros descartados.

Tipo: objeto [DestinationConfig \(p. 1002\)](#)

[EventSourceArn \(p. 959\)](#)

O nome de recurso da Amazon (ARN) da origem do evento.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+(:([a-z]{2}(-gov)?-[a-z]+\-\d{1}))?:(\d{12})?:(.*?)`

[FunctionArn \(p. 959\)](#)

O ARN da função Lambda.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[FunctionResponseTypes \(p. 959\)](#)

(Somente transmissões) Uma lista de enums de tipos de resposta atuais aplicados ao mapeamento de fontes de eventos.

Tipo: matriz de strings

Membros da matriz: número mínimo de 0 itens. Número máximo de 1 item.

Valores válidos: ReportBatchItemFailures

[LastModified \(p. 959\)](#)

A data em que o mapeamento de fontes de eventos foi atualizado pela última vez ou seu estado mudou, em segundos no horário do Unix.

Type: timestamp

[LastProcessingResult \(p. 959\)](#)

O resultado da última invocação do Lambda da sua função.

Tipo: sequência

[MaximumBatchingWindowInSeconds \(p. 959\)](#)

(Transmissões e filas padrão do Amazon SQS) O tempo máximo usado pelo Lambda, em segundos, para reunir os registros antes de invocar a função.

Padrão: 0

Configuração relacionada: quando você define BatchSize como um valor maior que 10, deve definir MaximumBatchingWindowInSeconds como pelo menos 1.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 300.

[MaximumRecordAgeInSeconds \(p. 959\)](#)

(Somente streams) Descarta registros mais antigos que a idade especificada. O valor padrão é -1, o que define a idade máxima como infinito. Quando o valor é definido como infinito, o Lambda nunca descarta registros antigos.

Type: inteiro

Intervalo válido: valor mínimo de -1. Valor máximo de 604800.

[MaximumRetryAttempts \(p. 959\)](#)

(Somente streams) Descarta registros após o número especificado de novas tentativas.

O valor padrão é -1, o que define o número máximo de tentativas como infinito. Quando

MaximumRetryAttempts é infinito, o Lambda tenta executar novamente os registros com falha até que o registro expire na fonte de eventos.

Type: inteiro

Intervalo válido: valor mínimo de -1. Valor máximo de 10000.

[ParallelizationFactor \(p. 959\)](#)

(Somente transmissões) O número de lotes a serem processados de cada fragmento simultaneamente. O valor padrão é 1.

Type: inteiro

Faixa válida: valor mínimo de 1. Valor máximo de 10.

[Queues \(p. 959\)](#)

(Amazon MQ) O nome da fila de destino do agente do Amazon MQ a ser consumido.

Tipo: matriz de strings

Membros da matriz: número fixo de 1 item.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1000.

Pattern: [\s\S]*

[SelfManagedEventSource \(p. 959\)](#)

O cluster autogerenciado do Apache Kafka para sua fonte de eventos.

Tipo: objeto [SelfManagedEventSource \(p. 1036\)](#)

[SourceAccessConfigurations \(p. 959\)](#)

Uma matriz do protocolo de autenticação, os componentes da VPC ou o host virtual para proteger e definir a fonte de eventos.

Tipo: matriz de [SourceAccessConfiguration \(p. 1037\)](#) objetos

Membros da matriz: número mínimo de 0 itens. Número máximo de 22 itens.

[StartingPosition \(p. 959\)](#)

A posição em um fluxo da qual você deseja iniciar a leitura. Obrigatório para origens de fluxo do Amazon Kinesis, Amazon DynamoDB e Amazon MSK. AT_TIMESTAMP só é compatível com o Amazon Kinesis Streams.

Tipo: sequência

Valores válidos: TRIM_HORIZON | LATEST | AT_TIMESTAMP

[StartingPositionTimestamp \(p. 959\)](#)

Com StartingPosition definido como AT_TIMESTAMP, o tempo a partir do qual iniciar a leitura em segundos no horário do Unix.

Type: timestamp

[State \(p. 959\)](#)

O estado do mapeamento da fonte de eventos. Pode ser um destes: Creating, Enabling, Enabled, Disabling, Disabled, Updating ou Deleting.

Tipo: sequência

[StateTransitionReason \(p. 959\)](#)

Indica se um usuário ou o Lambda fez a última alteração no mapeamento de fontes de eventos.

Tipo: sequência

[Topics \(p. 959\)](#)

O nome do tópico do Kafka.

Tipo: matriz de strings

Membros da matriz: número fixo de 1 item.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 249.

Pattern: ^[^\n.]([a-zA-Z0-9\\-_\\.]+)

[TumblingWindowInSeconds \(p. 959\)](#)

(Somente transmissões) A duração, em segundos, de uma janela de processamento. O intervalo é 1 a 900 segundos.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 900.

[UUID \(p. 959\)](#)

O identificador do mapeamento de fontes de eventos.

Tipo: sequência

Errors

[InvalidOperationException](#)

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

[ResourceConflictException](#)

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

[ResourceInUseException](#)

A operação entra em conflito com a disponibilidade do recurso. Por exemplo, você tentou atualizar um mapeamento EventSource no estado “CREATING” ou tentou excluir um mapeamento EventSource atualmente no estado “UPDATING”.

Código de status HTTP: 400

[ResourceNotFoundException](#)

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

[ServiceException](#)

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

[TooManyRequestsException](#)

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

UpdateFunctionCode

Atualiza o código de uma função do Lambda. Se a assinatura de código estiver habilitada para a função, o pacote de código deve ser assinado por um editor confiável. Para obter mais informações, consulte [Configurar a assinatura de código](#).

O código da função é bloqueado quando você publica uma versão. Não é possível modificar o código de uma versão publicada, somente o da versão não publicada.

Note

Para uma função definida como uma imagem de contêiner, o Lambda resolve a etiqueta de imagem para um resumo de imagem. No Amazon ECR, se você atualizar a etiqueta de imagem para uma nova imagem, o Lambda não atualizará automaticamente a função.

Sintaxe da solicitação

```
PUT /2015-03-31/functions/FunctionName/code HTTP/1.1
Content-type: application/json

{
  "DryRun": boolean,
  "ImageUri": "string",
  "Publish": boolean,
  "RevisionId": "string",
  "S3Bucket": "string",
  "S3Key": "string",
  "S3ObjectVersion": "string",
  "ZipFile": blob
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

FunctionName [\(p. 965\)](#)

O nome da função Lambda.

Formatos de nome

- Nome da função - my-function.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- ARN parcial - 123456789012:function:my-function.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_+]):(\$LATEST|[a-zA-Z0-9-_+]))?

Obrigatório: sim

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[DryRun \(p. 965\)](#)

Defina como “true” para validar os parâmetros de solicitação e as permissões de acesso sem modificar o código da função.

Type: booliano

Exigido: Não

[ImageUri \(p. 965\)](#)

URI de uma imagem de contêiner no registro do Amazon ECR.

Tipo: string

Exigido: Não

[Publish \(p. 965\)](#)

Defina como “true” para publicar uma nova versão da função após a atualização do código. Isso tem o mesmo resultado que chamar [PublishVersion \(p. 919\)](#) separadamente.

Type: booliano

Exigido: Não

[RevisionId \(p. 965\)](#)

Atualize a função somente se o ID da revisão corresponder ao ID especificado. Use essa opção para evitar a modificação de uma função que foi alterada desde a última leitura.

Tipo: string

Exigido: Não

[S3Bucket \(p. 965\)](#)

Um bucket do Amazon S3 na mesma região da AWS que sua função. O bucket pode estar em uma outra conta da AWS.

Tipo: sequência

Restrições de tamanho: comprimento mínimo de 3. Tamanho máximo de 63.

Pattern: ^[0-9A-Za-z\.\-_]*(\?!\.)\$

Exigido: Não

[S3Key \(p. 965\)](#)

A chave do Amazon S3 do pacote de implantação.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Exigido: Não

[S3ObjectVersion \(p. 965\)](#)

Para objetos com controle de versão, a versão do objeto do pacote de implantação a ser usada.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Exigido: Não

[ZipFile \(p. 965\)](#)

O conteúdo codificado em base64 do pacote de implantação. AWS Os clientes do SDK e da AWS CLI lidam com a codificação para você.

Tipo: Objeto de dados binários codificado pelo Base64

Exigido: Não

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
  "CodeSha256": "string",
  "CodeSize": number,
  "DeadLetterConfig": {
    "TargetArn": "string"
  },
  "Description": "string",
  "Environment": {
    "Error": {
      "ErrorCode": "string",
      "Message": "string"
    },
    "Variables": {
      "string" : "string"
    }
  },
  "FileSystemConfigs": [
    {
      "Arn": "string",
      "LocalMountPath": "string"
    }
  ],
  "FunctionArn": "string",
  "FunctionName": "string",
  "Handler": "string",
  "ImageConfigResponse": {
    "Error": {
      "ErrorCode": "string",
      "Message": "string"
    },
    "ImageConfig": {
      "Command": [ "string" ],
      "EntryPoint": [ "string" ],
      "WorkingDirectory": "string"
    }
  },
  "KMSKeyArn": "string",
  "LastModified": "string",
  "LastUpdateStatus": "string",
  "LastUpdateStatusReason": "string",
  "LastUpdateStatusReasonCode": "string",
  "Layers": [
    {
      "Arn": "string",
      "CodeSize": number,
      "SigningJobArn": "string",
      "SigningProfileVersionArn": "string"
    }
  ],
}
```

```
"MasterArn": "string",
"MemorySize": number,
"PackageType": "string",
"RevisionId": "string",
"Role": "string",
"Runtime": "string",
"SigningJobArn": "string",
"SigningProfileVersionArn": "string",
"State": "string",
"StateReason": "string",
"StateReasonCode": "string",
"Timeout": number,
"TracingConfig": {
    "Mode": "string"
},
"Version": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[CodeSha256](#) (p. 967)

O hash SHA256 do pacote de implantação da função.

Tipo: sequência

[CodeSize](#) (p. 967)

O tamanho do pacote de implantação da função em bytes.

Tipo: longo

[DeadLetterConfig](#) (p. 967)

A fila de mensagens mortas da função.

Tipo: objeto [DeadLetterConfig](#) (p. 1001)

[Description](#) (p. 967)

A descrição da função.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

[Environment](#) (p. 967)

As [variáveis de ambiente](#) da função.

Tipo: objeto [EnvironmentResponse](#) (p. 1005)

[FileSystemConfigs](#) (p. 967)

Configurações de conexão para um [sistema de arquivos do Amazon EFS](#).

Tipo: matriz de [FileSystemConfig](#) (p. 1011) objetos

Membros da matriz: número máximo de 1 item.

[FunctionArn \(p. 967\)](#)

O nome do recurso da Amazon (ARN) da função.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$\$LATEST|[a-zA-Z0-9-_]+))?`

[FunctionName \(p. 967\)](#)

Nome da função.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 170.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+\d{1}:\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+(:(\$\$LATEST|[a-zA-Z0-9-_]+))?)`

[Handler \(p. 967\)](#)

A função que o Lambda chama para começar a executar a função.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 128.

Pattern: `[^\s]+`

[ImageConfigResponse \(p. 967\)](#)

Os valores de configuração da imagem da função.

Tipo: objeto [ImageConfigResponse \(p. 1025\)](#)

[KMSKeyArn \(p. 967\)](#)

A chave KMS usada para criptografar as variáveis do ambiente da função. Esta chave é retornada somente se você tiver configurado uma CMK gerenciada pelo cliente.

Tipo: sequência

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:.*|()`

[LastModified \(p. 967\)](#)

A data e a hora em que a função foi atualizada, no [formato ISO-8601](#) (AAAA-MM-DDThh:mm:ss.sTZD).

Tipo: sequência

[LastUpdateStatus \(p. 967\)](#)

O status da última atualização que foi executada na função. Ele é definido pela primeira vez como `Successful` após a conclusão da criação da função.

Tipo: sequência

Valores válidos: `Successful | Failed | InProgress`

[LastUpdateStatusReason \(p. 967\)](#)

O motivo pelo qual foi realizada a última atualização na função.

Tipo: sequência

[LastUpdateStatusReasonCode \(p. 967\)](#)

O código do motivo pelo qual foi realizada a última atualização na função.

Tipo: sequência

Valores válidos: `EniLimitExceeded | InsufficientRolePermissions | InvalidConfiguration | InternalError | SubnetOutOfIPAddresses | InvalidSubnet | InvalidSecurityGroup | ImageDeleted | ImageAccessDenied | InvalidImage`

[Layers \(p. 967\)](#)

As [camadas](#) da função.

Tipo: matriz de [Layer \(p. 1026\)](#) objetos

[MasterArn \(p. 967\)](#)

Para funções do Lambda@Edge, o ARN da função mestre.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\${LATEST|[a-zA-Z0-9-_]+))?`

[MemorySize \(p. 967\)](#)

A quantidade de memória disponível para a função no tempo de execução.

Type: inteiro

Intervalo válido: valor mínimo de 128. Valor máximo de 10240.

[PackageType \(p. 967\)](#)

O tipo de pacote de implantação. Defina como `Image` para imagem de contêiner e defina `Zip` para arquivo de documento `.zip`.

Tipo: sequência

Valores válidos: `Zip | Image`

[RevisionId \(p. 967\)](#)

A última revisão atualizada da função ou do alias.

Tipo: sequência

[Role \(p. 967\)](#)

A função de execução da função.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/.+]`

[Runtime \(p. 967\)](#)

O ambiente de execução da função do Lambda.

Tipo: sequência

Valores válidos: `nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | nodejs10.x | nodejs12.x | nodejs14.x | java8 | java8.al2 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | python3.9 | dotnetcore1.0 | dotnetcore2.0 |`

dotnetcore2.1 | dotnetcore3.1 | nodejs4.3-edge | go1.x | ruby2.5 | ruby2.7 | provided | provided.al2

[SigningJobArn](#) (p. 967)

O ARN do trabalho de assinatura.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*)([a-zA-Z0-9\-])+([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12})?:(.*)

[SigningProfileVersionArn](#) (p. 967)

O ARN da versão do perfil de assinatura.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*)([a-zA-Z0-9\-])+([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12})?:(.*)

[State](#) (p. 967)

O estado atual da função. Quando o estado é `Inactive`, você pode invocar a função para reativá-la.

Tipo: sequência

Valores válidos: `Pending` | `Active` | `Inactive` | `Failed`

[StateReason](#) (p. 967)

O motivo para o estado atual da função.

Tipo: sequência

[StateReasonCode](#) (p. 967)

O código do motivo para o estado atual da função. Quando o código for `Creating`, não será possível invocar ou modificar a função.

Tipo: sequência

Valores válidos: `Idle` | `Creating` | `Restoring` | `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup` | `ImageDeleted` | `ImageAccessDenied` | `InvalidImage`

[Timeout](#) (p. 967)

A quantidade de tempo, em segundos, que o Lambda permite que uma função seja executada antes de encerrá-la.

Type: inteiro

Faixa válida: valor mínimo de 1.

[TracingConfig](#) (p. 967)

A configuração de rastreamento de AWS X-Ray da função.

Tipo: objeto [TracingConfigResponse](#) (p. 1040)

[Version](#) (p. 967)

A versão da função do Lambda.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Pattern: (\\$LATEST | [0-9]+)

[VpcConfig \(p. 967\)](#)

A configuração de rede da função.

Tipo: objeto [VpcConfigResponse \(p. 1042\)](#)

Errors

`CodeSigningConfigNotFoundException`

A configuração de assinatura de código especificada não existe.

Código de status HTTP: 404

`CodeStorageExceededException`

Você excedeu seu tamanho máximo total de código por conta. [Saiba mais](#)

Código de status HTTP: 400

`CodeVerificationFailedException`

A assinatura de código falhou em uma ou mais verificações de validação em relação à incompatibilidade ou expiração de assinatura, e a política de assinatura de código está definida como ENFORCE. O Lambda bloqueia a implantação.

Código de status HTTP: 400

`InvalidCodeSignatureException`

A assinatura de código falhou na verificação de integridade. O Lambda sempre bloqueia a implantação se a verificação de integridade falhar, mesmo que a diretiva de assinatura de código esteja definida como WARN.

Código de status HTTP: 400

`InvalidParameterValueException`

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

`PreconditionFailedException`

O RevisionId fornecido não corresponde ao RevisionId mais recente da função ou do alias do Lambda. Chame a API `GetFunction` ou `GetAlias` para recuperar o RevisionId mais recente para o seu recurso.

Código de status HTTP: 412

`ResourceConflictException`

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

`ResourceNotFoundException`

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

UpdateFunctionConfiguration

Modifique as configurações específicas da versão de uma função do Lambda.

Quando você atualiza uma função, o Lambda provisiona uma instância da função e seus recursos de suporte. Se sua função se conecta a uma VPC, esse processo pode demorar um minuto. Durante esse período, não será possível modificar a função, mas ainda será possível invocá-la. Os campos `LastUpdateStatus`, `LastUpdateStatusReason` e `LastUpdateStatusReasonCode` na resposta de [GetFunctionConfiguration \(p. 851\)](#) indicam quando a atualização está concluída e a função está processando eventos com a nova configuração. Para obter mais informações, consulte [Estados da função](#).

Essas configurações podem variar entre as versões de uma função e são bloqueadas quando você publica uma versão. Não é possível modificar a configuração de uma versão publicada, somente a da versão não publicada.

Para configurar a simultaneidade da função, use [PutFunctionConcurrency \(p. 930\)](#). Para conceder permissões para uma conta ou serviço da AWS, use [AddPermission \(p. 775\)](#).

Sintaxe da solicitação

```
PUT /2015-03-31/functions/FunctionName/configuration HTTP/1.1
Content-type: application/json

{
  "DeadLetterConfig": {
    "TargetArn": "string"
  },
  "Description": "string",
  "Environment": {
    "Variables": {
      "string": "string"
    }
  },
  "FileSystemConfigs": [
    {
      "Arn": "string",
      "LocalMountPath": "string"
    }
  ],
  "Handler": "string",
  "ImageConfig": {
    "Command": [ "string" ],
    "EntryPoint": [ "string" ],
    "WorkingDirectory": "string"
  },
  "KMSKeyArn": "string",
  "Layers": [ "string" ],
  "MemorySize": number,
  "RevisionId": "string",
  "Role": "string",
  "Runtime": "string",
  "Timeout": number,
  "TracingConfig": {
    "Mode": "string"
  },
  "VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ]
  }
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 974)

O nome da função Lambda.

Formatos de nome

- Nome da função - my-function.
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- ARN parcial - 123456789012:function:my-function.

A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[DeadLetterConfig](#) (p. 974)

Uma configuração de dead letter queue que especifica a fila ou tópico em que o Lambda envia eventos assíncronos quando eles falham no processamento. Para obter mais informações, consulte [Filas de mensagens mortas](#).

Tipo: objeto [DeadLetterConfig](#) (p. 1001)

Exigido: Não

[Description](#) (p. 974)

Uma descrição da função.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

Exigido: Não

[Environment](#) (p. 974)

As variáveis de ambiente que são acessíveis pelo código de função durante a execução.

Tipo: objeto [Environment](#) (p. 1003)

Exigido: Não

[FileSystemConfigs](#) (p. 974)

Configurações de conexão para um sistema de arquivos do Amazon EFS.

Tipo: matriz de [FileSystemConfig](#) (p. 1011) objetos

Membros da matriz: número máximo de 1 item.

Exigido: Não

[Handler \(p. 974\)](#)

O nome do método em seu código que o Lambda chama para executar a função. O formato inclui o nome do arquivo. Ele também pode incluir namespaces e outros qualificadores, dependendo do tempo de execução. Para obter mais informações, consulte [Modelo de programação](#).

Tipo: sequência

Restrições de tamanho: tamanho máximo de 128.

Pattern: [^\s]+

Exigido: Não

[ImageConfig \(p. 974\)](#)

[Valores de configuração da imagem do contêiner](#) que substituem os valores no arquivo Dockerfile da imagem do contêiner.

Tipo: objeto [ImageConfig \(p. 1023\)](#)

Exigido: Não

[KMSKeyArn \(p. 974\)](#)

O ARN da chave do AWS Key Management Service (AWS KMS) usada para criptografar variáveis de ambiente de sua função. Se não for fornecido, o AWS Lambda usa uma chave de serviço padrão.

Tipo: sequência

Pattern: (arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:[.]*)|()

Exigido: Não

[Layers \(p. 974\)](#)

Uma lista de [camadas de função](#) para adicionar ao ambiente de execução da função. Especifique cada camada por seu ARN, incluindo a versão.

Tipo: matriz de strings

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

Exigido: Não

[MemorySize \(p. 974\)](#)

A quantidade de [memória disponível para a função](#) no tempo de execução. Aumentar a memória de função também aumenta sua alocação de CPU. O valor padrão é 128 MB. O valor pode ser qualquer múltiplo de 1 MB.

Type: inteiro

Intervalo válido: valor mínimo de 128. Valor máximo de 10240.

Exigido: Não

[RevisionId \(p. 974\)](#)

Atualize a função somente se o ID da revisão corresponder ao ID especificado. Use essa opção para evitar a modificação de uma função que foi alterada desde a última leitura.

Tipo: string

Exigido: Não

[Role \(p. 974\)](#)

O nome de recurso da Amazon (ARN) da função de execução da função.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\\-_/+]

Exigido: Não

[Runtime \(p. 974\)](#)

O identificador do tempo de execução da função.

Tipo: sequência

Valores válidos: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | nodejs10.x | nodejs12.x | nodejs14.x | java8 | java8.al2 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | python3.9 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | dotnetcore3.1 | nodejs4.3-edge | go1.x | ruby2.5 | ruby2.7 | provided | provided.al2

Exigido: Não

[Timeout \(p. 974\)](#)

O valor do tempo que o Lambda permite que uma função seja executada antes de encerrá-la. O padrão é 3 segundos. O valor máximo permitido é de 900 segundos. Para obter informações adicionais, consulte [Ambiente de execução do Lambda](#).

Type: inteiro

Faixa válida: valor mínimo de 1.

Exigido: Não

[TracingConfig \(p. 974\)](#)

Defina Mode como Active para criar uma amostra e rastrear um subconjunto de solicitações de entrada com o [x-Ray](#).

Tipo: objeto [TracingConfig \(p. 1039\)](#)

Exigido: Não

[VpcConfig \(p. 974\)](#)

Para a conectividade de rede para os recursos da AWS em uma VPC, especifique uma lista de grupos de segurança e sub-redes na VPC. Quando você conecta uma função a uma VPC, ela só pode acessar recursos e a Internet por meio dessa VPC. Para obter mais informações, consulte [Configurações da VPC](#).

Tipo: objeto [VpcConfig \(p. 1041\)](#)

Exigido: Não

Sintaxe da resposta

HTTP/1.1 200

```
Content-type: application/json

{
    "CodeSha256": "string",
    "CodeSize": number,
    "DeadLetterConfig": {
        "TargetArn": "string"
    },
    "Description": "string",
    "Environment": {
        "Error": {
            "ErrorCode": "string",
            "Message": "string"
        },
        "Variables": {
            "string" : "string"
        }
    },
    "FileSystemConfigs": [
        {
            "Arn": "string",
            "LocalMountPath": "string"
        }
    ],
    "FunctionArn": "string",
    "FunctionName": "string",
    "Handler": "string",
    "ImageConfigResponse": {
        "Error": {
            "ErrorCode": "string",
            "Message": "string"
        },
        "ImageConfig": {
            "Command": [ "string" ],
            "EntryPoint": [ "string" ],
            "WorkingDirectory": "string"
        }
    },
    "KMSKeyArn": "string",
    "LastModified": "string",
    "LastUpdateStatus": "string",
    "LastUpdateStatusReason": "string",
    "LastUpdateStatusReasonCode": "string",
    "Layers": [
        {
            "Arn": "string",
            "CodeSize": number,
            "SigningJobArn": "string",
            "SigningProfileVersionArn": "string"
        }
    ],
    "MasterArn": "string",
    "MemorySize": number,
    "PackageType": "string",
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "string",
    "SigningJobArn": "string",
    "SigningProfileVersionArn": "string",
    "State": "string",
    "StateReason": "string",
    "StateReasonCode": "string",
    "Timeout": number,
    "TracingConfig": {
        "Mode": "string"
    }
},
```

```
"Version": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[CodeSha256 \(p. 977\)](#)

O hash SHA256 do pacote de implantação da função.

Tipo: sequência

[CodeSize \(p. 977\)](#)

O tamanho do pacote de implantação da função em bytes.

Type: longo

[DeadLetterConfig \(p. 977\)](#)

A fila de mensagens mortas da função.

Tipo: objeto [DeadLetterConfig \(p. 1001\)](#)

[Description \(p. 977\)](#)

A descrição da função.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

[Environment \(p. 977\)](#)

As [variáveis de ambiente](#) da função.

Tipo: objeto [EnvironmentResponse \(p. 1005\)](#)

[FileSystemConfigs \(p. 977\)](#)

Configurações de conexão para um [sistema de arquivos do Amazon EFS](#).

Tipo: matriz de [FileSystemConfig \(p. 1011\)](#) objetos

Membros da matriz: número máximo de 1 item.

[FunctionArn \(p. 977\)](#)

O nome do recurso da Amazon (ARN) da função.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$\$LATEST|[a-zA-Z0-9-_]+))?`

[FunctionName \(p. 977\)](#)

Nome da função.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 170.

Pattern: (arn:(aws[a-zA-Z-]*):lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[Handler \(p. 977\)](#)

A função que o Lambda chama para começar a executar a função.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 128.

Pattern: [^\s]+

[ImageConfigResponse \(p. 977\)](#)

Os valores de configuração da imagem da função.

Tipo: objeto [ImageConfigResponse \(p. 1025\)](#)

[KMSKeyArn \(p. 977\)](#)

A chave KMS usada para criptografar as variáveis do ambiente da função. Esta chave é retornada somente se você tiver configurado uma CMK gerenciada pelo cliente.

Tipo: sequência

Pattern: (arn:(aws[a-zA-Z-]*):[a-zA-Z0-9-.]+:[^.]+|()

[LastModified \(p. 977\)](#)

A data e a hora em que a função foi atualizada, no [formato ISO-8601](#) (AAAA-MM-DDThh:mm:ss.sTZD).

Tipo: sequência

[LastUpdateStatus \(p. 977\)](#)

O status da última atualização que foi executada na função. Ele é definido pela primeira vez como `Successful` após a conclusão da criação da função.

Tipo: sequência

Valores válidos: `Successful` | `Failed` | `InProgress`

[LastUpdateStatusReason \(p. 977\)](#)

O motivo pelo qual foi realizada a última atualização na função.

Tipo: sequência

[LastUpdateStatusReasonCode \(p. 977\)](#)

O código do motivo pelo qual foi realizada a última atualização na função.

Tipo: sequência

Valores válidos: `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup` | `ImageDeleted` | `ImageAccessDenied` | `InvalidImage`

[Layers \(p. 977\)](#)

As [camadas](#) da função.

Tipo: matriz de [Layer](#) (p. 1026) objetos

[MasterArn](#) (p. 977)

Para funções do Lambda@Edge, o ARN da função mestre.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_+]))?

[MemorySize](#) (p. 977)

A quantidade de memória disponível para a função no tempo de execução.

Type: inteiro

Intervalo válido: valor mínimo de 128. Valor máximo de 10240.

[PackageType](#) (p. 977)

O tipo de pacote de implantação. Defina como `Image` para imagem de contêiner e defina `Zip` para arquivo de documento .zip.

Tipo: sequência

Valores válidos: `Zip` | `Image`

[RevisionId](#) (p. 977)

A última revisão atualizada da função ou do alias.

Tipo: sequência

[Role](#) (p. 977)

A função de execução da função.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-/]+

[Runtime](#) (p. 977)

O ambiente de execução da função do Lambda.

Tipo: sequência

Valores válidos: `nodejs` | `nodejs4.3` | `nodejs6.10` | `nodejs8.10` | `nodejs10.x` | `nodejs12.x` | `nodejs14.x` | `java8` | `java8.al2` | `java11` | `python2.7` | `python3.6` | `python3.7` | `python3.8` | `python3.9` | `dotnetcore1.0` | `dotnetcore2.0` | `dotnetcore2.1` | `dotnetcore3.1` | `nodejs4.3-edge` | `go1.x` | `ruby2.5` | `ruby2.7` | `provided` | `provided.al2`

[SigningJobArn](#) (p. 977)

O ARN do trabalho de assinatura.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+:([a-z]{2}(-gov)?-[a-z]+\d{1})?:(\d{12})?:(.*)

[SigningProfileVersionArn](#) (p. 977)

O ARN da versão do perfil de assinatura.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12})?:(.*)

[State \(p. 977\)](#)

O estado atual da função. Quando o estado é `Inactive`, você pode invocar a função para reativá-la.

Tipo: sequência

Valores válidos: `Pending` | `Active` | `Inactive` | `Failed`

[StateReason \(p. 977\)](#)

O motivo para o estado atual da função.

Tipo: sequência

[StateReasonCode \(p. 977\)](#)

O código do motivo para o estado atual da função. Quando o código for `Creating`, não será possível invocar ou modificar a função.

Tipo: sequência

Valores válidos: `Idle` | `Creating` | `Restoring` | `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfRange` | `InvalidSubnet` | `InvalidSecurityGroup` | `ImageDeleted` | `ImageAccessDenied` | `InvalidImage`

[Timeout \(p. 977\)](#)

A quantidade de tempo, em segundos, que o Lambda permite que uma função seja executada antes de encerrá-la.

Type: inteiro

Faixa válida: valor mínimo de 1.

[TracingConfig \(p. 977\)](#)

A configuração de rastreamento de AWS X-Ray da função.

Tipo: objeto [TracingConfigResponse \(p. 1040\)](#)

[Version \(p. 977\)](#)

A versão da função do Lambda.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 977\)](#)

A configuração de rede da função.

Tipo: objeto [VpcConfigResponse \(p. 1042\)](#)

Errors

CodeSigningConfigNotFoundException

A configuração de assinatura de código especificada não existe.

Código de status HTTP: 404
CodeVerificationFailedException

A assinatura de código falhou em uma ou mais verificações de validação em relação à incompatibilidade ou expiração de assinatura, e a política de assinatura de código está definida como ENFORCE. O Lambda bloqueia a implantação.

Código de status HTTP: 400
InvalidCodeSignatureException

A assinatura de código falhou na verificação de integridade. O Lambda sempre bloqueia a implantação se a verificação de integridade falhar, mesmo que a diretiva de assinatura de código esteja definida como WARN.

Código de status HTTP: 400
InvalidParameterValueException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400
PreconditionFailedException

O RevisionId fornecido não corresponde ao RevisionId mais recente da função ou do alias do Lambda. Chame a API GetFunction ou GetAlias para recuperar o RevisionId mais recente para o seu recurso.

Código de status HTTP: 412
ResourceConflictException

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409
ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404
ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500
TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)

- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

UpdateFunctionEventInvokeConfig

Atualiza a configuração para invocação assíncrona de uma função, uma versão ou um alias.

Para configurar opções de invocação assíncrona, use [PutFunctionEventInvokeConfig](#) (p. 933).

Sintaxe da solicitação

```
POST /2019-09-25/functions/FunctionName/event-invoke-config?Qualifier=Qualifier HTTP/1.1
Content-type: application/json

{
  "DestinationConfig": {
    "OnFailure": {
      "Destination": "string"
    },
    "OnSuccess": {
      "Destination": "string"
    }
  },
  "MaximumEventAgeInSeconds": number,
  "MaximumRetryAttempts": number
}
```

Parâmetros da solicitação de URI

A solicitação usa os parâmetros de URI a seguir.

[FunctionName](#) (p. 985)

O nome da função, versão ou alias do Lambda.

Formatos de nome

- Function name - my-function (somente nome), my-function:v1 (com alias).
- ARN da função - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- ARN parcial - 123456789012:function:my-function.

Você pode anexar um número de versão ou alias a qualquer um dos formatos. A restrição de comprimento se aplica apenas ao ARN completo. Se você especificar apenas o nome da função, ele será limitado a 64 caracteres.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:(aws[a-zA-Z-]*):lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

Obrigatório: sim

[Qualifier](#) (p. 985)

Um número de versão ou nome de alias.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Corpo da solicitação

A solicitação aceita os dados a seguir no formato JSON.

[DestinationConfig \(p. 985\)](#)

Um destino para eventos depois que eles foram enviados a uma função para processamento.

Destinations

- Function (Função) – o nome de recurso da Amazon (ARN) da função do Lambda.
- Queue (Fila) – o ARN de uma fila do SQS.
- Topic (Tópico) – o ARN de um tópico do SNS.
- Event Bus (Barramento de eventos) – o ARN de um barramento de eventos do Amazon EventBridge.

Tipo: objeto [DestinationConfig \(p. 1002\)](#)

Exigido: Não

[MaximumEventAgeInSeconds \(p. 985\)](#)

A idade máxima de uma solicitação que o Lambda envia a uma função para processamento.

Type: inteiro

Intervalo válido: valor mínimo de 60. Valor máximo de 21600.

Exigido: Não

[MaximumRetryAttempts \(p. 985\)](#)

O número máximo de vezes para tentar novamente quando a função retorna um erro.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 2.

Exigido: Não

Sintaxe da resposta

```
HTTP/1.1 200
Content-type: application/json

{
  "DestinationConfig": {
    "OnFailure": {
      "Destination": "string"
    },
    "OnSuccess": {
      "Destination": "string"
    }
  },
  "FunctionArn": "string",
  "LastModified": number,
  "MaximumEventAgeInSeconds": number,
  "MaximumRetryAttempts": number
}
```

Elementos de resposta

Se a ação for bem-sucedida, o serviço reenviará uma resposta HTTP 200.

Os seguintes dados são retornados no formato JSON pelo serviço.

[DestinationConfig \(p. 986\)](#)

Um destino para eventos depois que eles foram enviados a uma função para processamento.

Destinations

- Function (Função) – o nome de recurso da Amazon (ARN) da função do Lambda.
- Queue (Fila) – o ARN de uma fila do SQS.
- Topic (Tópico) – o ARN de um tópico do SNS.
- Event Bus (Barramento de eventos) – o ARN de um barramento de eventos do Amazon EventBridge.

Tipo: objeto [DestinationConfig \(p. 1002\)](#)

[FunctionArn \(p. 986\)](#)

O nome de recurso da Amazon (ARN) da função.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[LastModified \(p. 986\)](#)

A data e a hora em que a configuração foi atualizada pela última vez, em segundos no tempo do Unix.

Type: timestamp

[MaximumEventAgeInSeconds \(p. 986\)](#)

A idade máxima de uma solicitação que o Lambda envia a uma função para processamento.

Type: inteiro

Intervalo válido: valor mínimo de 60. Valor máximo de 21600.

[MaximumRetryAttempts \(p. 986\)](#)

O número máximo de vezes para tentar novamente quando a função retorna um erro.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 2.

Errors

InvalidArgumentException

Um dos parâmetros na solicitação é inválido.

Código de status HTTP: 400

ResourceConflictException

O recurso já existe ou outra operação está em andamento.

Código de status HTTP: 409

ResourceNotFoundException

O recurso especificado na solicitação não existe.

Código de status HTTP: 404

ServiceException

O serviço AWS Lambda encontrou um erro interno.

Código de status HTTP: 500

TooManyRequestsException

O limite de taxa de transferência da solicitação foi excedido.

Código de status HTTP: 429

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS Interface da linha de comando](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK para JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

Tipos de dados

Os seguintes tipos de dados são compatíveis:

- [AccountLimit \(p. 990\)](#)
- [AccountUsage \(p. 992\)](#)
- [AliasConfiguration \(p. 993\)](#)
- [AliasRoutingConfiguration \(p. 995\)](#)
- [AllowedPublishers \(p. 996\)](#)
- [CodeSigningConfig \(p. 997\)](#)
- [CodeSigningPolicies \(p. 999\)](#)
- [Concurrency \(p. 1000\)](#)
- [DeadLetterConfig \(p. 1001\)](#)
- [DestinationConfig \(p. 1002\)](#)
- [Environment \(p. 1003\)](#)
- [EnvironmentError \(p. 1004\)](#)
- [EnvironmentResponse \(p. 1005\)](#)
- [EventSourceMappingConfiguration \(p. 1006\)](#)
- [FileSystemConfig \(p. 1011\)](#)
- [FunctionCode \(p. 1012\)](#)
- [FunctionCodeLocation \(p. 1014\)](#)
- [FunctionConfiguration \(p. 1015\)](#)

- [FunctionEventInvokeConfig \(p. 1021\)](#)
- [ImageConfig \(p. 1023\)](#)
- [ImageConfigError \(p. 1024\)](#)
- [ImageConfigResponse \(p. 1025\)](#)
- [Layer \(p. 1026\)](#)
- [LayersListItem \(p. 1027\)](#)
- [LayerVersionContentInput \(p. 1028\)](#)
- [LayerVersionContentOutput \(p. 1029\)](#)
- [LayerVersionsListItem \(p. 1030\)](#)
- [OnFailure \(p. 1032\)](#)
- [OnSuccess \(p. 1033\)](#)
- [ProvisionedConcurrencyConfigListItem \(p. 1034\)](#)
- [SelfManagedEventSource \(p. 1036\)](#)
- [SourceAccessConfiguration \(p. 1037\)](#)
- [TracingConfig \(p. 1039\)](#)
- [TracingConfigResponse \(p. 1040\)](#)
- [VpcConfig \(p. 1041\)](#)
- [VpcConfigResponse \(p. 1042\)](#)

AccountLimit

Limites relacionados à simultaneidade e ao armazenamento. Todos os tamanhos de arquivo e armazenamento estão em bytes.

Contents

CodeSizeUnzipped

O tamanho máximo do pacote e das camadas de implantação de uma função quando eles são extraídos.

Type: longo

Exigido: Não

CodeSizeZipped

O tamanho máximo de um pacote de implantação quando ele é enviado por upload diretamente para o Lambda. Use o Amazon S3 para arquivos maiores.

Type: longo

Exigido: Não

ConcurrentExecutions

O número máximo de execuções simultâneas da função.

Type: inteiro

Exigido: Não

TotalCodeSize

A quantidade de espaço de armazenamento que você pode usar para todos os pacotes de implantação e arquivos de camada.

Type: longo

Exigido: Não

UnreservedConcurrentExecutions

O número máximo de execuções simultâneas da função, menos a capacidade reservada para funções individuais com [PutFunctionConcurrency \(p. 930\)](#).

Type: inteiro

Faixa válida: valor mínimo de 0.

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)

- [AWS SDK for Ruby V3](#)

AccountUsage

O número de funções e a quantidade de armazenamento em uso.

Contents

FunctionCount

O número de funções do Lambda.

Type: longo

Exigido: Não

TotalCodeSize

A quantidade de armazenamento, em bytes, em uso por pacotes de implantação e arquivos de camada.

Type: longo

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

AliasConfiguration

Fornece informações de configuração sobre o [alias](#) de uma função do Lambda.

Contents

AliasArn

O nome do recurso da Amazon (ARN) do alarme do alias.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?

Exigido: Não

Description

Uma descrição do alias.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

Exigido: Não

FunctionVersion

A versão da função que o alias invoca.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Pattern: (\$LATEST|[0-9]+)

Exigido: Não

Name

O nome do alias.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 128.

Pattern: (?![0-9]+\$)([a-zA-Z0-9-_]+)

Exigido: Não

RevisionId

Um identificador exclusivo que muda ao atualizar o alias.

Tipo: string

Exigido: Não

RoutingConfig

A [configuração de roteamento](#) do alias.

Tipo: objeto [AliasRoutingConfiguration](#) (p. 995)

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

AliasRoutingConfiguration

A configuração de [mudança de tráfego](#) de um alias da função do Lambda.

Contents

AdditionalVersionWeights

A segunda versão e a porcentagem de tráfego que é roteada para ela.

Tipo: mapa de string para double

Restrições de tamanho de chave: tamanho mínimo de 1. Tamanho máximo de 1024.

Padrão da chave: [0–9]⁺

Faixa válida: valor mínimo de 0.0. Valor máximo de 1.0.

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

AllowedPublishers

Lista de perfis de assinatura que podem assinar um pacote de código.

Contents

SigningProfileVersionArns

O nome de recurso da Amazon (ARN) para cada um dos perfis de assinatura. Um perfil de assinatura define um usuário confiável que pode assinar um pacote de código.

Tipo: matriz de strings

Membros da matriz: número mínimo de 1 item. Número máximo de 20 itens.

Pattern: arn:(aws[a-zA-Z0-9-]*)([a-zA-Z0-9\-])+([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12})?:(.*?)

Obrigatório: sim

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

CodeSigningConfig

Detalhes sobre uma [Configuração de assinatura de código](#).

Contents

AllowedPublishers

Lista de editores permitidos.

Tipo: objeto [AllowedPublishers](#) (p. 996)

Obrigatório: sim

CodeSigningConfigArn

O nome do recurso da Amazon (ARN) da configuração de assinatura de código.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 200.

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}((-gov)|(-iso(b?)))?- [a-z]+-\d{1}:\d{12}:code-signing-config:csc-[a-zA-Z0-9]{17}

Obrigatório: sim

CodeSigningConfigId

Identificador exclusivo para a configuração de assinatura de código.

Tipo: sequência

Pattern: csc-[a-zA-Z0-9-_\.]{17}

Obrigatório: sim

CodeSigningPolicies

A política de assinatura de código controla a ação de falha de validação para incompatibilidade de assinatura ou expiração.

Tipo: objeto [CodeSigningPolicies](#) (p. 999)

Obrigatório: sim

Description

Descrição da configuração de assinatura de código.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

Exigido: Não

LastModified

A data e a hora em que a configuração de assinatura de código foi modificada pela última vez, no formato ISO-8601 (AAAA-MM-DDThh:mm:ss.sTZD).

Tipo: sequência

Obrigatório: sim

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

CodeSigningPolicies

As [políticas](#) de configuração de assinatura de código especificam a ação de falha de validação para incompatibilidade de assinatura ou expiração.

Contents

UntrustedArtifactOnDeployment

Política de configuração de assinatura de código para falha de validação de implantação. Se você definir a política como `Enforce`, o Lambda bloqueará a solicitação de implantação se as verificações de validação de assinatura apresentarem falha. Se você definir a política como `Warn`, o Lambda permitirá a implantação e criará um log do CloudWatch.

Valor padrão: `Warn`

Tipo: sequência

Valores válidos: `Warn` | `Enforce`

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

Concurrency

Contents

ReservedConcurrentExecutions

O número de execuções simultâneas que estão reservadas para essa função. Para obter mais informações, consulte [Gerenciamento de simultaneidade](#).

Type: inteiro

Faixa válida: valor mínimo de 0.

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

DeadLetterConfig

A fila de mensagens mortas de invocações assíncronas com falha.

Contents

TargetArn

O nome de recurso da Amazon (ARN) de uma fila do Amazon SQS ou um tópico do Amazon SNS.

Tipo: sequência

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:[.*])|()`

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

DestinationConfig

Um objeto de configuração que especifica o destino de um evento depois que o Lambda processá-lo.

Contents

OnFailure

A configuração de destino para invocações com falha.

Tipo: objeto [OnFailure \(p. 1032\)](#)

Exigido: Não

OnSuccess

A configuração de destino para invocações com êxito.

Tipo: objeto [OnSuccess \(p. 1033\)](#)

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

Environment

As configurações da variável de ambiente da função. É possível usar variáveis de ambiente para ajustar o comportamento da função sem atualizar o código. Uma variável de ambiente é um par de strings armazenadas na configuração específica da versão de uma função.

Contents

Variables

Pares de chave-valor da variável de ambiente. Para obter mais informações, consulte [Usar variáveis de ambiente do Lambda](#).

Tipo:: mapa de string para string

Padrão da chave: [a-zA-Z]([a-zA-Z0-9_])+

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

EnvironmentError

As mensagens de erro para variáveis do ambiente que não puderam ser aplicadas.

Contents

ErrorCode

O código do erro.

Tipo: string

Exigido: Não

Message

A mensagem de erro.

Tipo: string

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

EnvironmentResponse

Os resultados de uma operação para atualizar ou ler variáveis de ambiente. Se a operação for bem-sucedida, a resposta conterá as variáveis de ambiente. Se falhar, a resposta conterá detalhes sobre o erro.

Contents

Error

As mensagens de erro para variáveis do ambiente que não puderam ser aplicadas.

Tipo: objeto [EnvironmentError](#) (p. 1004)

Exigido: Não

Variables

Pares de chave-valor da variável de ambiente.

Tipo:: mapa de string para string

Padrão da chave: [a-zA-Z]([a-zA-Z0-9_])+

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

EventSourceMappingConfiguration

Um mapeamento entre um recurso da AWS e uma função do Lambda. Para obter mais detalhes, consulte [CreateEventSourceMapping \(p. 786\)](#).

Contents

BatchSize

O número máximo de registros em cada batch que o Lambda extrai da sua transmissão ou fila e envia para sua função. O Lambda transmite todos os registros no batch para a função em uma única chamada até o limite de carga útil para invocação síncrona (6 MB).

Valor padrão: varia de acordo com o serviço. Para o Amazon SQS, o padrão é 10. Para todos os outros serviços, o padrão é 100.

Configuração relacionada: quando você define `BatchSize` como um valor maior que 10, deve definir `MaximumBatchingWindowInSeconds` como pelo menos 1.

Type: inteiro

Faixa válida: valor mínimo de 1. Valor máximo de 10000.

Exigido: Não

BisectBatchOnFunctionError

(Somente streams) Se a função retornar um erro, divida o lote em dois e tente novamente. O valor padrão é falso.

Type: booleano

Exigido: Não

DestinationConfig

(Somente streams) Uma fila do Amazon SQS ou um destino de tópico do Amazon SNS para registros descartados.

Tipo: objeto [DestinationConfig \(p. 1002\)](#)

Exigido: Não

EventSourceArn

O nome de recurso da Amazon (ARN) da origem do evento.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-\-])+([a-z]{2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.*?)`

Exigido: Não

FunctionArn

O ARN da função Lambda.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*):lambda:[a-z]{2}(-gov)?-[a-z]+\-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Exigido: Não

FunctionResponseTypes

(Somente transmissões) Uma lista de enums de tipos de resposta atuais aplicados ao mapeamento de fontes de eventos.

Tipo: matriz de strings

Membros da matriz: número mínimo de 0 itens. Número máximo de 1 item.

Valores válidos: ReportBatchItemFailures

Exigido: Não

LastModified

A data em que o mapeamento de fontes de eventos foi atualizado pela última vez ou seu estado mudou, em segundos no horário do Unix.

Tipo: timestamp

Exigido: Não

LastProcessingResult

O resultado da última invocação do Lambda da sua função.

Tipo: string

Exigido: Não

MaximumBatchingWindowInSeconds

(Transmissões e filas padrão do Amazon SQS) O tempo máximo usado pelo Lambda, em segundos, para reunir os registros antes de invocar a função.

Padrão: 0

Configuração relacionada: quando você define BatchSize como um valor maior que 10, deve definir MaximumBatchingWindowInSeconds como pelo menos 1.

Tipo: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 300.

Exigido: Não

MaximumRecordAgeInSeconds

(Somente streams) Descarta registros mais antigos que a idade especificada. O valor padrão é -1, o que define a idade máxima como infinito. Quando o valor é definido como infinito, o Lambda nunca descarta registros antigos.

Tipo: inteiro

Intervalo válido: valor mínimo de -1. Valor máximo de 604800.

Exigido: Não

MaximumRetryAttempts

(Somente streams) Descarta registros após o número especificado de novas tentativas.

O valor padrão é -1, o que define o número máximo de tentativas como infinito. Quando

MaximumRetryAttempts é infinito, o Lambda tenta executar novamente os registros com falha até que o registro expire na fonte de eventos.

Type: inteiro

Intervalo válido: valor mínimo de -1. Valor máximo de 10000.

Exigido: Não

ParallelizationFactor

(Somente transmissões) O número de lotes a serem processados de cada fragmento simultaneamente. O valor padrão é 1.

Type: inteiro

Faixa válida: valor mínimo de 1. Valor máximo de 10.

Exigido: Não

Queues

(Amazon MQ) O nome da fila de destino do agente do Amazon MQ a ser consumido.

Tipo: matriz de strings

Membros da matriz: número fixo de 1 item.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1000.

Pattern: [\s\S]*

Exigido: Não

SelfManagedEventSource

O cluster autogerenciado do Apache Kafka para sua fonte de eventos.

Tipo: objeto [SelfManagedEventSource \(p. 1036\)](#)

Exigido: Não

SourceAccessConfigurations

Uma matriz do protocolo de autenticação, os componentes da VPC ou o host virtual para proteger e definir a fonte de eventos.

Tipo: matriz de [SourceAccessConfiguration \(p. 1037\)](#) objetos

Membros da matriz: número mínimo de 0 itens. Número máximo de 22 itens.

Exigido: Não

StartingPosition

A posição em um fluxo da qual você deseja iniciar a leitura. Obrigatório para origens de fluxo do Amazon Kinesis, Amazon DynamoDB e Amazon MSK. AT_TIMESTAMP só é compatível com o Amazon Kinesis Streams.

Tipo: sequência

Valores válidos: TRIM_HORIZON | LATEST | AT_TIMESTAMP

Exigido: Não

StartingPositionTimestamp

Com StartingPosition definido como AT_TIMESTAMP, o tempo a partir do qual iniciar a leitura em segundos no horário do Unix.

Type: timestamp

Exigido: Não

State

O estado do mapeamento da fonte de eventos. Pode ser um destes: Creating, Enabling, Enabled, Disabling, Disabled, Updating ou Deleting.

Tipo: string

Exigido: Não

StateTransitionReason

Indica se um usuário ou o Lambda fez a última alteração no mapeamento de fontes de eventos.

Tipo: string

Exigido: Não

Topics

O nome do tópico do Kafka.

Tipo: matriz de strings

Membros da matriz: número fixo de 1 item.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 249.

Pattern: ^[^ .]([a-zA-Z0-9\-_ .]+)

Exigido: Não

TumblingWindowInSeconds

(Somente transmissões) A duração, em segundos, de uma janela de processamento. O intervalo é 1 a 900 segundos.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 900.

Exigido: Não

UUID

O identificador do mapeamento de fontes de eventos.

Tipo: string

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

FileSystemConfig

Detalhes sobre a conexão entre uma função do Lambda e um [sistema de arquivos do Amazon EFS](#).

Contents

Arn

O nome de recurso da Amazon (ARN) do ponto de acesso do Amazon EFS que fornece acesso ao sistema de arquivos.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 200.

Pattern: `arn:aws[a-zA-Z-]*:elasticfilesystem:[a-z]{2}((-gov)|(-iso(b?)))?-[_a-zA-Z+-]\d{1}:\d{12}:access-point/fsap-[a-f0-9]{17}`

Obrigatório: sim

LocalMountPath

O caminho onde a função pode acessar o sistema de arquivos, começando com `/mnt/`.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 160.

Pattern: `^/mnt/[a-zA-Z0-9-_\.]+\$`

Obrigatório: sim

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

FunctionCode

O código da função do Lambda. É possível especificar um objeto no Amazon S3, fazer upload de um pacote de implantação de arquivo .zip diretamente, ou especificar o URI de uma imagem de contêiner.

Contents

ImageUri

URI de uma [imagem de contêiner](#) no registro do Amazon ECR.

Tipo: string

Exigido: Não

S3Bucket

Um bucket do Amazon S3 na mesma região da AWS que sua função. O bucket pode estar em uma outra conta da AWS.

Tipo: sequência

Restrições de tamanho: comprimento mínimo de 3. Tamanho máximo de 63.

Pattern: ^[0-9A-Za-z\.\-_]*(\?!\\.)\$

Exigido: Não

S3Key

A chave do Amazon S3 do pacote de implantação.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Exigido: Não

S3ObjectVersion

Para objetos com controle de versão, a versão do objeto do pacote de implantação a ser usada.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Exigido: Não

ZipFile

O conteúdo codificado em base64 do pacote de implantação. AWS Os clientes do SDK e da AWS CLI lidam com a codificação para você.

Tipo: Objeto de dados binários codificado pelo Base64

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

FunctionCodeLocation

Detalhes sobre o pacote de implantação de uma função.

Contents

ImageUri

URI de uma imagem de contêiner no registro do Amazon ECR.

Tipo: string

Exigido: Não

Location

Um URL pré-assinado que você pode usar para fazer download do pacote de implantação.

Tipo: string

Exigido: Não

RepositoryType

O serviço que está hospedando o arquivo.

Tipo: string

Exigido: Não

ResolvedImageUri

O URI resolvido da imagem.

Tipo: string

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

FunctionConfiguration

Detalhes sobre a configuração de uma função.

Contents

CodeSha256

O hash SHA256 do pacote de implantação da função.

Tipo: string

Exigido: Não

CodeSize

O tamanho do pacote de implantação da função em bytes.

Tipo: longo

Exigido: Não

DeadLetterConfig

A fila de mensagens mortas da função.

Tipo: objeto [DeadLetterConfig \(p. 1001\)](#)

Exigido: Não

Description

A descrição da função.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

Exigido: Não

Environment

As [variáveis de ambiente](#) da função.

Tipo: objeto [EnvironmentResponse \(p. 1005\)](#)

Exigido: Não

FileSystemConfigs

Configurações de conexão para um [sistema de arquivos do Amazon EFS](#).

Tipo: matriz de [FileSystemConfig \(p. 1011\)](#) objetos

Membros da matriz: número máximo de 1 item.

Exigido: Não

FunctionArn

O nome do recurso da Amazon (ARN) da função.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$\$LATEST|[a-zA-Z0-9-_]+))?`

Exigido: Não
FunctionName

Nome da função.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 170.

Pattern: `(arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\$\$LATEST|[a-zA-Z0-9-_]+))?`

Exigido: Não
Handler

A função que o Lambda chama para começar a executar a função.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 128.

Pattern: `[^\s]+`

Exigido: Não
ImageConfigResponse

Os valores de configuração da imagem da função.

Tipo: objeto [ImageConfigResponse \(p. 1025\)](#)

Exigido: Não
KMSKeyArn

A chave KMS usada para criptografar as variáveis do ambiente da função. Esta chave é retornada somente se você tiver configurado uma CMK gerenciada pelo cliente.

Tipo: sequência

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-_\.]+\.*|()`

Exigido: Não
LastModified

A data e a hora em que a função foi atualizada, no [formato ISO-8601](#) (AAAA-MM-DDThh:mm:ss.sTZD).

Tipo: string

Exigido: Não
LastUpdateStatus

O status da última atualização que foi executada na função. Ele é definido pela primeira vez como `Successful` após a conclusão da criação da função.

Tipo: sequência

Valores válidos: `Successful | Failed | InProgress`

Exigido: Não

LastUpdateStatusReason

O motivo pelo qual foi realizada a última atualização na função.

Tipo: string

Exigido: Não

LastUpdateStatusReasonCode

O código do motivo pelo qual foi realizada a última atualização na função.

Tipo: sequência

Valores válidos: `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup` | `ImageDeleted` | `ImageAccessDenied` | `InvalidImage`

Exigido: Não

Layers

As [camadas](#) da função.

Tipo: matriz de [Layer \(p. 1026\)](#) objetos

Exigido: Não

MasterArn

Para funções do Lambda@Edge, o ARN da função mestre.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Exigido: Não

MemorySize

A quantidade de memória disponível para a função no tempo de execução.

Type: inteiro

Intervalo válido: valor mínimo de 128. Valor máximo de 10240.

Exigido: Não

PackageType

O tipo de pacote de implantação. Defina como `Image` para imagem de contêiner e defina `Zip` para arquivo de documento `.zip`.

Tipo: sequência

Valores válidos: `Zip` | `Image`

Exigido: Não

RevisionId

A última revisão atualizada da função ou do alias.

Tipo: string

Exigido: Não

Role

A função de execução da função.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@/-/_]+

Exigido: Não

Runtime

O ambiente de execução da função do Lambda.

Tipo: sequência

Valores válidos: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | nodejs10.x | nodejs12.x | nodejs14.x | java8 | java8.al2 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | python3.9 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | dotnetcore3.1 | nodejs4.3-edge | go1.x | ruby2.5 | ruby2.7 | provided | provided.al2

Exigido: Não

SigningJobArn

O ARN do trabalho de assinatura.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-\-])+:(a-z){2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.*)

Exigido: Não

SigningProfileVersionArn

O ARN da versão do perfil de assinatura.

Tipo: sequência

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-\-])+:(a-z){2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.*)

Exigido: Não

State

O estado atual da função. Quando o estado é Inactive, você pode invocar a função para reativá-la.

Tipo: sequência

Valores válidos: Pending | Active | Inactive | Failed

Exigido: Não

StateReason

O motivo para o estado atual da função.

Tipo: string

Exigido: Não
StateReasonCode

O código do motivo para o estado atual da função. Quando o código for `Creating`, não será possível invocar ou modificar a função.

Tipo: sequência

Valores válidos: `Idle` | `Creating` | `Restoring` | `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfRangeAddresses` | `InvalidSubnet` | `InvalidSecurityGroup` | `ImageDeleted` | `ImageAccessDenied` | `InvalidImage`

Exigido: Não
Timeout

A quantidade de tempo, em segundos, que o Lambda permite que uma função seja executada antes de encerrá-la.

Type: inteiro

Faixa válida: valor mínimo de 1.

Exigido: Não
TracingConfig

A configuração de rastreamento de AWS X-Ray da função.

Tipo: objeto [TracingConfigResponse \(p. 1040\)](#)

Exigido: Não
Version

A versão da função do Lambda.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Pattern: `(\$LATEST|[0-9]+)`

Exigido: Não
VpcConfig

A configuração de rede da função.

Tipo: objeto [VpcConfigResponse \(p. 1042\)](#)

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)

- [AWS SDK for Ruby V3](#)

FunctionEventInvokeConfig

Contents

DestinationConfig

Um destino para eventos depois que eles foram enviados a uma função para processamento.

Destinations

- Function (Função) – o nome de recurso da Amazon (ARN) da função do Lambda.
- Queue (Fila) – o ARN de uma fila do SQS.
- Topic (Tópico) – o ARN de um tópico do SNS.
- Event Bus (Barramento de eventos) – o ARN de um barramento de eventos do Amazon EventBridge.

Tipo: objeto [DestinationConfig \(p. 1002\)](#)

Exigido: Não

FunctionArn

O nome de recurso da Amazon (ARN) da função.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+\-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Exigido: Não

LastModified

A data e a hora em que a configuração foi atualizada pela última vez, em segundos no tempo do Unix.

Type: timestamp

Exigido: Não

MaximumEventAgeInSeconds

A idade máxima de uma solicitação que o Lambda envia a uma função para processamento.

Type: inteiro

Intervalo válido: valor mínimo de 60. Valor máximo de 21600.

Exigido: Não

MaximumRetryAttempts

O número máximo de vezes para tentar novamente quando a função retorna um erro.

Type: inteiro

Faixa válida: valor mínimo de 0. Valor máximo de 2.

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for Ruby V3

ImageConfig

Valores de configuração que substituem as configurações do Dockerfile da imagem do contêiner. Consulte [Configurações de contêiner](#).

Contents

Command

Especifica parâmetros que você deseja transmitir com ENTRYPPOINT.

Tipo: matriz de strings

Membros da matriz: número máximo de 1500 itens.

Exigido: Não

EntryPoint

Especifica o ponto de entrada para a aplicação, que normalmente é o local do executável durante a execução.

Tipo: matriz de strings

Membros da matriz: número máximo de 1500 itens.

Exigido: Não

WorkingDirectory

Especifica o diretório de trabalho.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 1000.

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

ImageConfigError

Resposta de erro para GetFunctionConfiguration

Contents

ErrorCode

Código de erro.

Tipo: string

Exigido: Não

Message

A mensagem de erro.

Tipo: string

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

ImageConfigResponse

Resposta à solicitação de GetFunctionConfiguration.

Contents

Error

Resposta de erro para GetFunctionConfiguration.

Tipo: objeto [ImageConfigError \(p. 1024\)](#)

Exigido: Não

ImageConfig

Valores de configuração que substituem o Dockerfile da imagem do contêiner.

Tipo: objeto [ImageConfig \(p. 1023\)](#)

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

Layer

Uma [camada do AWS Lambda](#).

Contents

Arn

O nome de recurso da Amazon (ARN) da camada da função.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: `arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\:\d{12}:layer:[a-zA-Z0-9-_+]:[0-9]+`

Exigido: Não

CodeSize

O tamanho do arquivamento de camada em bytes.

Tipo: longo

Exigido: Não

SigningJobArn

O nome do recurso da Amazon (ARN) de um trabalho de assinatura.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z0-9-]*)([a-zA-Z0-9\-\-])+([a-z]{2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.**)`

Exigido: Não

SignedProfileVersionArn

O nome do recurso da Amazon (ARN) para um perfil de assinatura.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z0-9-]*)([a-zA-Z0-9\-\-])+([a-z]{2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.**)`

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

LayersListItem

Detalhes sobre uma [camada do AWS Lambda](#).

Contents

LatestMatchingVersion

A versão mais recente da camada.

Tipo: objeto [LayerVersionsListItem](#) (p. 1030)

Exigido: Não

LayerArn

O nome de recurso da Amazon (ARN) da camada da função.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\:\d{12}:layer:[a-zA-Z0-9-_]+

Exigido: Não

LayerName

O nome da camada.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\:\d{12}:layer:[a-zA-Z0-9-_]+)|[a-zA-Z0-9-_]+

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

LayerVersionContentInput

Um arquivo ZIP que contém os conteúdos de uma [camada do AWS Lambda](#). Você pode especificar um local do Amazon S3 ou fazer upload de um arquivo de camada diretamente.

Contents

S3Bucket

O bucket do Amazon S3 do arquivamento de camada.

Tipo: sequência

Restrições de tamanho: comprimento mínimo de 3. Tamanho máximo de 63.

Pattern: ^[0-9A-Za-z\.\-_]*(<!\.)\$

Exigido: Não

S3Key

A chave do Amazon S3 do arquivamento de camada.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Exigido: Não

S3ObjectVersion

Para objetos com controle de versão, a versão do objeto de arquivamento de camada a ser usada.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 1024.

Exigido: Não

ZipFile

O conteúdo codificado em base64 do arquivo de camada. Os clientes do SDK e da AWS CLI lidam com a codificação para você.

Tipo: Objeto de dados binários codificado pelo Base64

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

LayerVersionContentOutput

Detalhes sobre uma versão de uma [camada do AWS Lambda](#).

Contents

CodeSha256

O hash SHA-256 do arquivo de camada.

Tipo: string

Exigido: Não

CodeSize

O tamanho do arquivamento de camada em bytes.

Tipo: longo

Exigido: Não

Location

Um link para o arquivo de camada no Amazon S3 que é válido por 10 minutos.

Tipo: string

Exigido: Não

SigningJobArn

O nome do recurso da Amazon (ARN) de um trabalho de assinatura.

Tipo: string

Exigido: Não

SigningProfileVersionArn

O nome do recurso da Amazon (ARN) para um perfil de assinatura.

Tipo: string

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

LayerVersionsListItem

Detalhes sobre uma versão de uma [camada do AWS Lambda](#).

Contents

CompatibleRuntimes

Os tempos de execução compatíveis da camada.

Tipo: matriz de strings

Membros da matriz: número máximo de 15 itens.

Valores válidos: nodejs | nodejs4.3 | nodejs6.10 | nodejs8.10 | nodejs10.x | nodejs12.x | nodejs14.x | java8 | java8.al2 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | python3.9 | dotnetcore1.0 | dotnetcore2.0 | dotnetcore2.1 | dotnetcore3.1 | nodejs4.3-edge | go1.x | ruby2.5 | ruby2.7 | provided | provided.al2

Exigido: Não

CreatedDate

A data em que a versão foi criada, no formato ISO 8601. Por exemplo,
2018-11-27T15:10:45.123+0000.

Tipo: string

Exigido: Não

Description

A descrição da versão.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 256.

Exigido: Não

LayerVersionArn

O ARN da versão da camada.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

Exigido: Não

LicensesInfo

A licença de código aberto da camada.

Tipo: sequência

Restrições de tamanho: tamanho máximo de 512.

Exigido: Não

Version

O número da versão.

Type: longo

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

OnFailure

Um destino para eventos que tiveram falha no processamento.

Contents

Destination

O nome de recurso da Amazon (ARN) do recurso de destino.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 350.

Pattern: ^\$|arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-.])+:([a-z]{2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.*)

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

OnSuccess

Um destino para eventos que foram processados com êxito.

Contents

Destination

O nome de recurso da Amazon (ARN) do recurso de destino.

Tipo: sequência

Restrições de comprimento: comprimento mínimo de 0. Tamanho máximo de 350.

Pattern: ^\$|arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-.])+:([a-z]{2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.*)

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

ProvisionedConcurrencyConfigListItem

Detalhes sobre a configuração de simultaneidade provisionada para o alias ou a versão de uma função.

Contents

AllocatedProvisionedConcurrentExecutions

A quantidade de simultaneidade provisionada alocada.

Type: inteiro

Faixa válida: valor mínimo de 0.

Exigido: Não

AvailableProvisionedConcurrentExecutions

A quantidade de simultaneidade provisionada disponível.

Type: inteiro

Faixa válida: valor mínimo de 0.

Exigido: Não

FunctionArn

O nome do recurso da Amazon (ARN) do alias ou da versão.

Tipo: sequência

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-z]{2}(-gov)?-[a-z]+-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\${LATEST|[a-zA-Z0-9-_]+))?`

Exigido: Não

LastModified

A data e hora em que um usuário atualizou a configuração pela última vez, no [formato ISO 8601](#).

Tipo: string

Exigido: Não

RequestedProvisionedConcurrentExecutions

A quantidade de simultaneidade provisionada solicitada.

Type: inteiro

Faixa válida: valor mínimo de 1.

Exigido: Não

Status

O status do processo de alocação.

Tipo: sequência

Valores válidos: `IN_PROGRESS` | `READY` | `FAILED`

Exigido: Não

StatusReason

Para alocações com falha, o motivo pelo qual a simultaneidade provisionada não pôde ser alocada.

Tipo: string

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

SelfManagedEventSource

O cluster autogerenciado do Apache Kafka para sua fonte de eventos.

Contents

Endpoints

A lista de servidores de bootstrap para seus corretores Kafka no seguinte formato:
"KAFKA_BOOTSTRAP_SERVERS": ["abc.xyz.com:xxxx", "abc2.xyz.com:xxxx"].

Tipo: string para a matriz do mapa de strings

Entradas do mapa: número máximo de 2 itens.

Chaves válidas: KAFKA_BOOTSTRAP_SERVERS

Membros da matriz: número mínimo de 1 item. Número máximo de 10 itens.

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 300.

Pattern: ^(([a-zA-Z0-9] | [a-zA-Z0-9][a-zA-Z0-9\-\-]*[a-zA-Z0-9])\.)*([A-Za-z0-9] | [A-Za-z0-9][A-Za-z0-9\-\-]*[A-Za-z0-9]):[0-9]{1,5}

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

SourceAccessConfiguration

Para proteger e definir o acesso à sua fonte de eventos, é possível especificar o protocolo de autenticação, os componentes da VPC ou o host virtual.

Contents

Type

O tipo do protocolo de autenticação, os componentes da VPC ou o host virtual da fonte de eventos.
Por exemplo: "Type" : "SASL_SCRAM_512_AUTH".

- **BASIC_AUTH** - (Amazon MQ) O segredo do AWS Secrets Manager que armazena as credenciais do agente.
- **BASIC_AUTH** - (Apache Kafka autogerenciado) O ARN do Secrets Manager de sua chave secreta usado para a autenticação SASL/PLAIN de seus agentes do Apache Kafka.
- **VPC_SUBNET** - As sub-redes associadas à sua VPC. O Lambda conecta essas sub-redes para buscar dados do cluster autogerenciado do Apache Kafka.
- **VPC_SECURITY_GROUP** - O grupo de segurança da VPC usado para gerenciar o acesso a seus agentes autogerenciados do Apache Kafka.
- **SASL_SCRAM_256_AUTH** - O ARN do Secrets Manager de sua chave secreta usado para a autenticação SASL SCRAM-256 de seus agentes autogerenciados do Apache Kafka.
- **SASL_SCRAM_512_AUTH** - O ARN do Secrets Manager de sua chave secreta usado para a autenticação SASL SCRAM-512 de seus agentes autogerenciados do Apache Kafka.
- **VIRTUAL_HOST** - (Amazon MQ) O nome do host virtual em seu agente do RabbitMQ. O Lambda usa esse host RabbitMQ como fonte de eventos.

Tipo: sequência

Valores válidos: `BASIC_AUTH | VPC_SUBNET | VPC_SECURITY_GROUP | SASL_SCRAM_512_AUTH | SASL_SCRAM_256_AUTH | VIRTUAL_HOST`

Exigido: Não

URI

O valor para a configuração escolhida em Type. Por exemplo: "URI" :
`"arn:aws:secretsmanager:us-east-1:01234567890:secret:MyBrokerSecretName"`.

Tipo: sequência

Restrições de tamanho: tamanho mínimo 1. Tamanho máximo de 200.

Pattern: `[a-zA-Z0-9-\/*:_+=.@-]*`

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

TracingConfig

A configuração de rastreamento do [AWS X-Ray](#) da função. Defina o Mode como Active para fazer amostragens e registrar as solicitações de entrada.

Contents

Mode

O modo de rastreamento.

Tipo: sequência

Valores válidos: Active | PassThrough

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

TracingConfigResponse

A configuração de rastreamento de AWS X-Ray da função.

Contents

Mode

O modo de rastreamento.

Tipo: sequência

Valores válidos: Active | PassThrough

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

VpcConfig

Os grupos de segurança da VPC e as sub-redes que estão anexadas a uma função do Lambda. Para obter mais informações, consulte [Configurações da VPC](#).

Contents

SecurityGroupIds

Uma lista de IDs dos grupos de segurança da VPC.

Tipo: matriz de strings

Membros da matriz: número máximo de 5 itens.

Exigido: Não

SubnetIds

Uma lista de IDs de sub-rede da VPC.

Tipo: matriz de strings

Membros da matriz: número máximo de 16 itens.

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

VpcConfigResponse

Os grupos de segurança da VPC e as sub-redes que estão anexadas a uma função do Lambda.

Contents

SecurityGroupIds

Uma lista de IDs dos grupos de segurança da VPC.

Tipo: matriz de strings

Membros da matriz: número máximo de 5 itens.

Exigido: Não

SubnetIds

Uma lista de IDs de sub-rede da VPC.

Tipo: matriz de strings

Membros da matriz: número máximo de 16 itens.

Exigido: Não

VpcId

O ID da VPC.

Tipo: string

Exigido: Não

Consulte também

Para obter mais informações sobre como usar essa API em um dos AWS SDKs específicos de linguagem, consulte o seguinte:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java V2](#)
- [AWS SDK for Ruby V3](#)

Erros de certificado ao usar um SDK

Como os AWS SDKs usam os certificados de CA do computador, as alterações feitas nos certificados nos servidores da AWS podem provocar falhas de conexão quando você tenta usar um SDK. Você pode impedir essas falhas mantendo os certificados de CA do computador e o sistema operacional atualizados. Se encontrar esse problema em um ambiente corporativo e não gerenciar seu computador, talvez tenha de pedir a um administrador para auxiliá-lo no processo de atualização. A lista a seguir mostra as versões mínimas de sistema operacional e Java:

- As versões do Microsoft Windows que têm atualizações de janeiro de 2005 ou posteriores instaladas contêm pelo menos uma das CAs necessárias em sua lista de confiança.

- O Mac OS X 10.4 com Java para Mac OS X 10.4 Release 5 (fevereiro de 2007), Mac OS X 10.5 (outubro de 2007) e versões posteriores contêm pelo menos uma das CAs necessárias em sua lista de confiança.
- O Red Hat Enterprise Linux 5 (março de 2007), 6 e 7 e o CentOS 5, 6 e 7 contêm pelo menos uma das CAs necessárias em sua lista de CAs confiáveis padrão.
- Java 1.4.2_12 (maio de 2006), 5 Update 2 (março de 2005) e todas as versões mais recentes, como Java 6 (dezembro de 2006), 7 e 8, contêm pelo menos uma das CAs necessárias em sua lista de CAs confiáveis padrão.

Ao acessar o console de gerenciamento do AWS Lambda ou os endpoints de API do AWS Lambda, seja por meio dos navegadores ou de maneira programática, é necessário verificar se suas máquinas clientes suportam algum destes CAs:

- Amazon Root CA 1
- Starfield Services Root Certificate Authority – G2
- Starfield Class 2 Certification Authority

Os certificados raiz das duas primeiras autoridades estão disponíveis em [Amazon Trust Services](#), mas manter o computador atualizado é a solução mais simples. Para saber mais sobre os certificados fornecidos pelo ACM, consulte [Perguntas frequentes sobre o AWS Certificate Manager](#).

AWSGlossário da

Para obter a terminologia mais recente da AWS, consulte o [glossário da AWS](#) na Referência geral da AWS.