
AWS Lambda

开发人员指南



AWS Lambda: 开发人员指南

Copyright © 2020 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

什么是 AWS Lambda ?	1
应在何时使用 AWS Lambda ?	1
您是 AWS Lambda 的新用户吗 ?	1
入门	3
创建函数	3
使用 Designer	3
调用 Lambda 函数	4
代码编辑器	5
处理文件和文件夹	6
使用代码	7
在全屏模式下工作	10
使用首选项	10
AWS CLI	11
先决条件	11
创建执行角色	11
创建函数	12
列出您的账户中的 Lambda 函数	14
检索 Lambda 函数	15
清除	15
概念	16
功能	16
Runtime	16
Event	16
并发	17
Trigger	17
功能	17
编程模型	17
部署程序包	19
层	19
扩展	19
并发控制	20
异步调用	22
事件源映射	23
目标	24
函数蓝图	25
应用程序模板	25
工具	26
AWS Command Line Interface	26
AWS 无服务器应用程序模型	26
SAM CLI	26
代码编写工具	27
限制	27
权限	30
执行角色	30
在 IAM 控制台中创建执行角色	31
使用 IAM API 管理角色	32
Lambda 功能的托管策略	33
基于资源的策略	33
向 AWS 服务授予函数访问权	35
向其他账户授予函数访问权	35
向其他账户授予层访问权	36
清除基于资源的策略	37
用户策略	37
函数开发	38

层开发和使用	40
跨账户角色	41
资源和条件	41
函数	43
事件源映射	44
层	44
权限边界	45
管理函数	47
配置控制台	47
环境变量	49
运行时环境变量	51
保护环境变量	52
使用 Lambda API 配置环境变量	53
示例代码和模板	54
并发	54
配置预留并发	55
配置预配置并发	57
使用 Lambda API 配置并发	60
版本	63
使用 Lambda API 管理版本	64
使用版本	64
资源策略	65
别名	65
使用 Lambda API 管理别名	66
使用别名	66
资源策略	67
别名路由配置	67
层	68
将函数配置为使用层	69
管理层	69
在层中包括库依赖项	71
层权限	72
Network	72
执行角色和用户权限	73
使用 Lambda API 配置 Amazon VPC 访问权限	74
对 VPC 连接函数的 Internet 和服务访问	74
VPC 配置示例	74
Database	75
示例应用程序	75
标签	77
通过 AWS CLI 使用标签	79
标签键和值要求	80
调用函数	81
同步调用	81
异步调用	83
配置异步调用的错误处理	85
配置异步调用目标	85
异步调用配置 API	87
死信队列	88
事件源映射	90
函数状态	92
函数扩展	94
错误处理	98
适用于 Android 的 移动软件开发工具包	99
教程	100
示例代码	105
Lambda 运行时	108

执行关联	109
运行时支持策略	110
自定义运行时	111
使用自定义运行时	111
构建自定义运行时	111
运行时接口	112
下一个调用	113
调用响应	114
调用错误	114
初始化错误	114
教程 – 自定义运行时	115
先决条件	115
创建函数	115
创建层	117
更新函数	117
更新运行时	118
共享层	119
清除	119
Lambda 应用程序	120
管理应用程序	120
监控应用程序	121
自定义监控控制面板	121
教程 – 创建应用程序	122
先决条件	123
创建应用程序	123
调用函数	124
添加 AWS 资源	124
更新权限边界	125
更新函数代码	126
后续步骤	127
故障排除	127
清除	128
示例 – 错误处理器	128
架构和事件结构	129
使用 AWS X-Ray 进行检测	130
AWS CloudFormation 模板和其他资源	131
滚动部署	132
示例 AWS SAM Lambda 模板	132
使用案例	133
示例 1：Amazon S3 推送事件并调用 Lambda 函数	133
示例 2：AWS Lambda 从 Kinesis 流中拉取事件并调用 Lambda 函数	134
最佳实践	135
函数代码	135
函数配置	136
警报与指标	136
流事件调用	136
使用其他服务	138
应用程序负载均衡器	139
Alexa	141
API 网关	141
权限	144
利用 API 网关 API 处理错误	145
选择 API 类型	146
示例应用程序	147
教程	147
示例代码	155
微服务蓝图	158

示例模板	159
AWS CloudTrail	159
教程	161
示例代码	165
CloudWatch Events	167
教程	168
示例模板	170
计划表达式	171
Amazon CloudWatch Logs	172
AWS CloudFormation	172
CloudFront	174
AWS CodeCommit	176
CodePipeline	176
权限	178
教程	178
Amazon Cognito	183
AWS Config	184
Amazon DynamoDB	184
执行角色权限	186
将流配置为事件源	186
事件源映射 API	187
错误处理	189
Amazon CloudWatch 指标	190
教程	190
示例代码	194
示例模板	197
Amazon ElastiCache	198
先决条件	198
创建执行角色	198
创建 ElastiCache 集群	199
创建部署程序包	199
创建 Lambda 函数	200
测试 Lambda 函数。	200
Amazon EC2	200
权限	201
教程 – Spot 实例	202
AWS IoT	210
AWS IoT Events	211
Kinesis	212
配置您的数据流和函数	213
执行角色权限	214
将流配置为事件源	214
事件源映射 API	215
错误处理	217
Amazon CloudWatch 指标	218
教程	218
示例代码	221
示例模板	224
Kinesis Data Firehose	226
Amazon Lex	226
角色和权限	228
Amazon RDS	229
教程 : Amazon RDS	230
Amazon S3 事件	233
教程	234
示例代码	240
示例模板	246

Amazon S3 批处理操作	246
从 Amazon S3 批处理操作调用 Lambda 函数	247
Amazon SES	248
Amazon SNS	250
教程	251
示例代码	253
Amazon SQS	255
扩展和处理	256
配置队列以便与 Lambda 一起使用	257
执行角色权限	257
将队列配置为事件源	257
事件源映射 API	187
教程	258
示例代码	261
示例模板	264
使用 Node.js	265
处理程序	266
异步处理程序	267
非异步处理程序	267
部署程序包	268
更新没有依赖项的函数	268
更新具有额外依赖项的函数	269
上下文	270
日志记录	271
在 AWS 管理控制台中查看日志	272
使用 AWS CLI	272
删除日志	274
错误	274
跟踪	275
使用 Python	277
处理程序	278
部署程序包	279
先决条件	280
更新没有依赖项的函数	280
更新具有额外依赖项的函数	280
使用虚拟环境	281
上下文	283
日志记录	284
在 AWS 管理控制台中查看日志	285
使用 AWS CLI	285
删除日志	287
日志记录库	287
错误	287
跟踪	288
使用 Ruby	291
处理程序	292
部署程序包	293
更新没有依赖项的函数	293
更新具有额外依赖项的函数	294
上下文	295
日志记录	295
在 AWS 管理控制台中查看日志	285
使用 AWS CLI	285
删除日志	287
错误	299
使用 Java	301
部署程序包	302

使用 Gradle 构建部署程序包	303
使用 Maven 构建部署程序包	304
使用 Lambda API 上传部署程序包	305
使用 AWS SAM 上传部署程序包	306
处理程序	307
选择输入和输出类型	309
处理程序接口	309
示例处理程序代码	310
上下文	311
示例应用程序中的上下文	313
日志记录	313
在 AWS 管理控制台中查看日志	315
使用 AWS CLI	315
删除日志	316
使用 Log4j 2 和 SLF4J 进行高级日志记录	316
日志记录代码示例	318
错误	319
查看错误输出	320
了解错误类型和来源	321
客户端中的错误处理	322
其他 AWS 服务中的错误处理	323
示例应用程序中的错误处理	323
跟踪	323
使用 Lambda API 启用主动跟踪	325
使用 AWS CloudFormation 启用主动跟踪	325
示例应用程序中的跟踪	326
教程 - Eclipse IDE	327
先决条件	327
创建并构建项目	327
使用 Go	330
部署程序包	330
在 Windows 上创建部署程序包	331
处理程序	331
使用结构化类型的 Lambda 函数处理程序	332
使用全局状态	333
上下文	334
访问调用上下文信息	335
日志记录	336
在 AWS 管理控制台中查看日志	337
使用 AWS CLI	337
删除日志	339
错误	339
跟踪	339
安装适用于 Go 的 X-Ray 开发工具包	339
配置适用于 Go 的 X-Ray 开发工具包	340
创建子分段	340
Capture	340
跟踪 HTTP 请求	340
环境变量	341
使用 C#	342
部署程序包	342
.NET Core CLI	343
AWS Toolkit for Visual Studio	346
处理程序	347
处理流	348
处理标准数据类型	348
处理程序签名	349

序列化 Lambda 函数	350
Lambda 函数处理程序限制	350
在使用 C# 编写的 AWS Lambda 函数中应用 Async	351
上下文	352
日志记录	352
在 AWS 管理控制台中查看日志	354
使用 AWS CLI	354
删除日志	355
错误	355
使用 PowerShell	359
开发环境	359
部署程序包	360
处理程序	361
返回数据	362
上下文	362
日志记录	362
在 AWS 管理控制台中查看日志	364
使用 AWS CLI	364
删除日志	365
错误	366
监控	367
监控控制台	367
函数指标	368
使用调用指标	368
使用性能指标	369
使用并发指标	369
CloudWatch 日志	370
使用 AWS X-Ray	371
利用 AWS X-Ray 跟踪基于 Lambda 的应用程序	371
从 Lambda 函数发送跟踪分段	372
Lambda 环境中的 AWS X-Ray 守护程序	373
使用环境变量与 AWS X-Ray 通信	373
在 AWS X-Ray 控制台中跟踪 Lambda：示例	373
CloudTrail	374
CloudTrail 中的 AWS Lambda 信息	375
了解 AWS Lambda 日志文件条目	376
使用 CloudTrail 跟踪函数调用	377
安全性	378
数据保护	378
传输中加密	379
静态加密	379
Identity and Access Management	379
受众	379
使用身份进行身份验证	380
使用策略管理访问	381
AWS Lambda 如何与 IAM 协同工作	382
基于身份的策略示例	382
故障排除	384
合规性验证	385
恢复功能	386
基础设施安全	386
配置和漏洞分析	386
故障排除	388
部署	388
调用	390
执行	391
联网	392

版本	393
早期更新	398
API 参考	402
Actions	402
AddLayerVersionPermission	404
AddPermission	407
CreateAlias	411
CreateEventSourceMapping	415
CreateFunction	421
DeleteAlias	430
DeleteEventSourceMapping	432
DeleteFunction	436
DeleteFunctionConcurrency	438
DeleteFunctionEventInvokeConfig	440
DeleteLayerVersion	442
DeleteProvisionedConcurrencyConfig	444
GetAccountSettings	446
GetAlias	448
GetEventSourceMapping	451
GetFunction	455
GetFunctionConcurrency	458
GetFunctionConfiguration	460
GetFunctionEventInvokeConfig	466
GetLayerVersion	469
GetLayerVersionByArn	472
GetLayerVersionPolicy	475
GetPolicy	477
GetProvisionedConcurrencyConfig	479
Invoke	482
InvokeAsync	487
ListAliases	489
ListEventSourceMappings	492
ListFunctionEventInvokeConfigs	495
ListFunctions	498
ListLayers	501
ListLayerVersions	503
ListProvisionedConcurrencyConfigs	506
ListTags	509
ListVersionsByFunction	511
PublishLayerVersion	514
PublishVersion	518
PutFunctionConcurrency	525
PutFunctionEventInvokeConfig	528
PutProvisionedConcurrencyConfig	532
RemoveLayerVersionPermission	535
RemovePermission	537
TagResource	539
UntagResource	541
UpdateAlias	543
UpdateEventSourceMapping	547
UpdateFunctionCode	553
UpdateFunctionConfiguration	560
UpdateFunctionEventInvokeConfig	569
Data Types	572
AccountLimit	574
AccountUsage	575
AliasConfiguration	576

AliasRoutingConfiguration	578
Concurrency	579
DeadLetterConfig	580
DestinationConfig	581
Environment	582
EnvironmentError	583
EnvironmentResponse	584
EventSourceMappingConfiguration	585
FunctionCode	588
FunctionCodeLocation	589
FunctionConfiguration	590
FunctionEventInvokeConfig	595
Layer	597
LayersListItem	598
LayerVersionContentInput	599
LayerVersionContentOutput	600
LayerVersionsListItem	601
OnFailure	603
OnSuccess	604
ProvisionedConcurrencyConfigListItem	605
TracingConfig	607
TracingConfigResponse	608
VpcConfig	609
VpcConfigResponse	610
在使用 SDK 时出现证书错误	610
AWS 词汇表	612

什么是 AWS Lambda ?

AWS Lambda 是一项计算服务，可使您无需预配置或管理服务器即可运行代码。AWS Lambda 只在需要时执行您的代码并自动缩放，从每天几个请求到每秒数千个请求。您只需按消耗的计算时间付费 – 代码未运行时不产生费用。借助 AWS Lambda，您几乎可以为任何类型的应用程序或后端服务运行代码，并且不必进行任何管理。AWS Lambda 在可用性高的计算基础设施上运行您的代码，执行计算资源的所有管理工作，其中包括服务器和操作系统维护、容量预置和自动扩展、代码监控和记录。您只需要以 [AWS Lambda 支持的一种语言 \(p. 108\)](#) 提供您的代码。

您可以使用 AWS Lambda 运行代码以响应事件，例如更改 Amazon S3 存储桶或 Amazon DynamoDB 表中的数据；以及使用 Amazon API Gateway 运行代码以响应 HTTP 请求；或者使用通过 AWS SDK 完成的 API 调用来调用您的代码。借助这些功能，您可以使用 Lambda 轻松地为 Amazon S3 和 Amazon DynamoDB 等 AWS 服务构建数据处理触发程序，处理 Kinesis 中存储的流数据，或创建您自己的按 AWS 规模、性能和安全性运行的后端。

您也可以构建由事件触发的函数组成的无服务器应用程序，并使用 CodePipeline 和 AWS CodeBuild 自动部署这些应用程序。有关更多信息，请参阅[AWS Lambda 应用程序 \(p. 120\)](#)。

应在何时使用 AWS Lambda ?

AWS Lambda 是很多应用程序场景的理想计算平台，只要您可以用 AWS Lambda 支持的语言编写应用程序代码并在 AWS Lambda 标准运行时环境和 Lambda 提供的资源中运行。

在使用 AWS Lambda 时，您只需负责自己的代码。AWS Lambda 管理提供内存、CPU、网络和其他资源均衡的计算机群。这是以灵活性为代价的，这意味着您不能登录计算实例，或针对提供的运行时自定义操作系统。通过这些约束，AWS Lambda 可以代表您执行操作和管理活动，包括预置容量、监控机群运行状况、应用安全补丁、部署您的代码以及监控和记录您的 Lambda 函数日志。

如果您需要管理自己的计算资源，Amazon Web Services 还提供了其他计算服务以满足您的需求。

- Amazon Elastic Compute Cloud (Amazon EC2) 服务提供灵活性和各种 EC2 实例类型供您选择。它允许您选择自定义操作系统、网络和安全性设置以及整个软件堆栈，但您负责预置容量、监控机群运行状况和性能以及使用可用区来实现容错。
- Elastic Beanstalk 提供易用的服务，您可将应用程序部署和扩展到 Amazon EC2 上，在其中您保留对底层 EC2 实例的所有权和完整控制权。

Lambda 是一项高度可用的服务。有关更多信息，请参阅 [AWS Lambda 服务等级协议](#)。

您是 AWS Lambda 的新用户吗？

如果您是首次接触 AWS Lambda 的用户，我们建议您按顺序阅读以下内容：

1. 阅读产品概述并观看宣传视频，以了解示例使用案例。这些资源可在 [AWS Lambda 网页](#) 上找到。
2. 尝试基于控制台的入门练习。此练习提供了使用控制台创建和测试您的第一个 Lambda 函数的说明。您还将了解编程模型和其他 Lambda 概念。有关更多信息，请参阅[开始使用 AWS Lambda \(p. 3\)](#)。
3. 阅读本指南的[使用 AWS Lambda 部署应用程序 \(p. 120\)](#)部分。本部分介绍了您可以用来打造端到端体验的各种 AWS Lambda 组件。

除了入门练习之外，您还可浏览各种使用案例，每个使用案例都随附有指导您完成示例方案的教程。根据您的应用程序需求（例如，无论您需要事件驱动型 Lambda 函数调用还是按需调用），您可按照满足您特定需求的特定教程进行操作。有关更多信息，请参阅[将 AWS Lambda 与其他服务结合使用 \(p. 138\)](#)。

开始使用 AWS Lambda

要开始使用 AWS Lambda，请使用 Lambda 控制台创建函数。在几分钟的时间内，您可以创建一个函数，调用它，并查看日志、指标和跟踪数据。

Note

要使用 Lambda 和其他 AWS 服务，您需要 AWS 账户。如果您没有账户，请访问 aws.amazon.com，然后选择创建 AWS 账户。有关详细说明，请参阅[创建和激活 AWS 账户](#)。作为最佳实践，您还应创建一个具有管理员权限的 AWS Identity and Access Management (IAM) 用户，并在不需要根凭证的所有工作中使用该用户。创建密码以用于访问控制台，并创建访问密钥以使用命令行工具。有关说明，请参阅 IAM 用户指南 中的[创建您的第一个 IAM 管理员用户和组](#)。

您可以在 Lambda 控制台中编写函数，也可以使用 IDE 工具包、命令行工具或软件开发工具包编写函数。Lambda 控制台为非编译语言提供了[代码编辑器 \(p. 5\)](#)，使您可以快速修改和测试代码。[AWS CLI \(p. 11\)](#) 使您可以直接访问 Lambda API 以获取高级配置和自动化使用案例。

主题

- [使用控制台创建 Lambda 函数 \(p. 3\)](#)
- [使用 AWS Lambda 控制台编辑器创建函数 \(p. 5\)](#)
- [将 AWS Lambda 与 AWS Command Line Interface 结合使用 \(p. 11\)](#)
- [AWS Lambda 概念 \(p. 16\)](#)
- [AWS Lambda 功能 \(p. 17\)](#)
- [与 AWS Lambda 一起使用的工具 \(p. 26\)](#)
- [AWS Lambda 限制 \(p. 27\)](#)

使用控制台创建 Lambda 函数

在本入门练习中，您将使用 AWS Lambda 控制台创建一个 Lambda 函数。接下来，您将使用示例事件数据手动调用 Lambda 函数。AWS Lambda 将执行 Lambda 函数并返回结果。然后，您将验证执行结果，包括您的 Lambda 函数已创建的日志和各种 CloudWatch 指标。

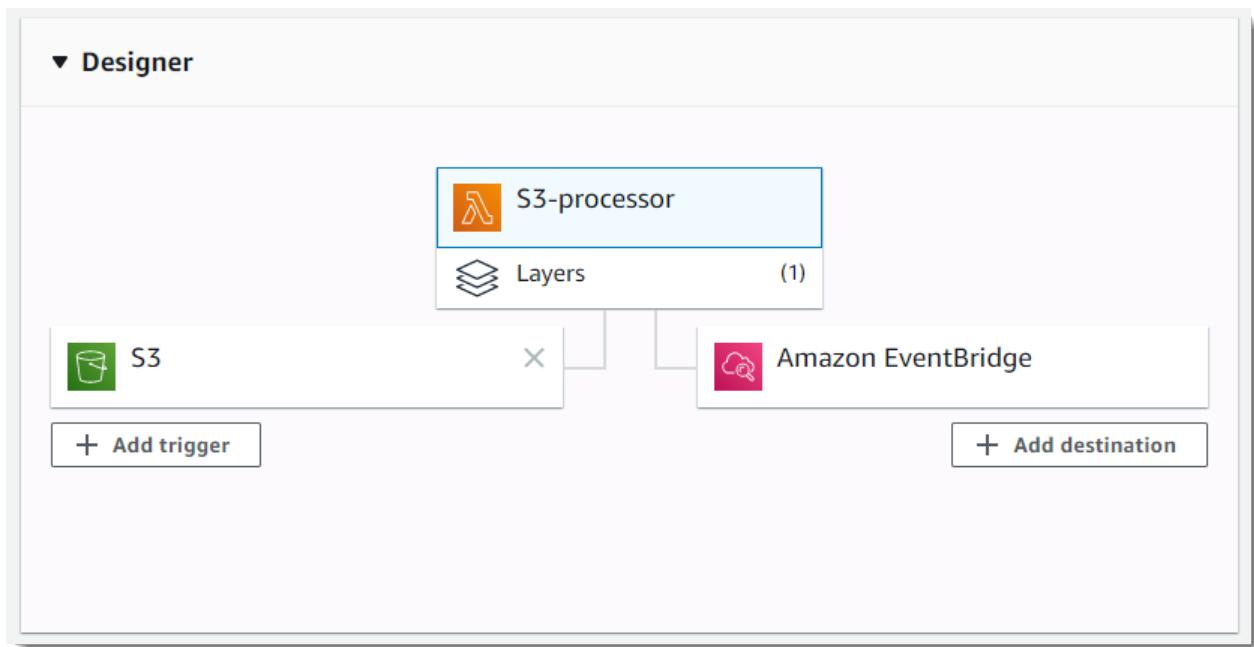
创建 Lambda 函数

1. 打开 [AWS Lambda 控制台](#)。
2. 选择 Create a function。
3. 在函数名称中，输入 **my-function**。
4. 选择 Create function。

Lambda 创建一个 Node.js 函数和授予该函数上传日志的权限的执行角色。在您调用函数时，Lambda 代入执行角色，并使用它为 AWS 开发工具包创建凭证和从事件源读取数据。

使用 Designer

设计器显示您的函数及其上游和下游资源的概述。您可以使用它来配置触发器、层和目标。



在 Designer 中选择 my-function 返回到函数的代码和配置。对于脚本语言，Lambda 包含返回成功响应的示例代码。只要源代码未超过 3 MB 的限制，您就可以使用嵌入式 AWS Cloud9 编辑器编辑函数代码。

调用 Lambda 函数

使用控制台中提供的示例事件数据调用 Lambda 函数。

调用函数

1. 在右上角，选择测试。
2. 在 Configure test event 页面中，选择 Create new test event，并且在 Event template 中，保留默认的 Hello World 选项。输入 Event name 并记录以下示例事件模板：

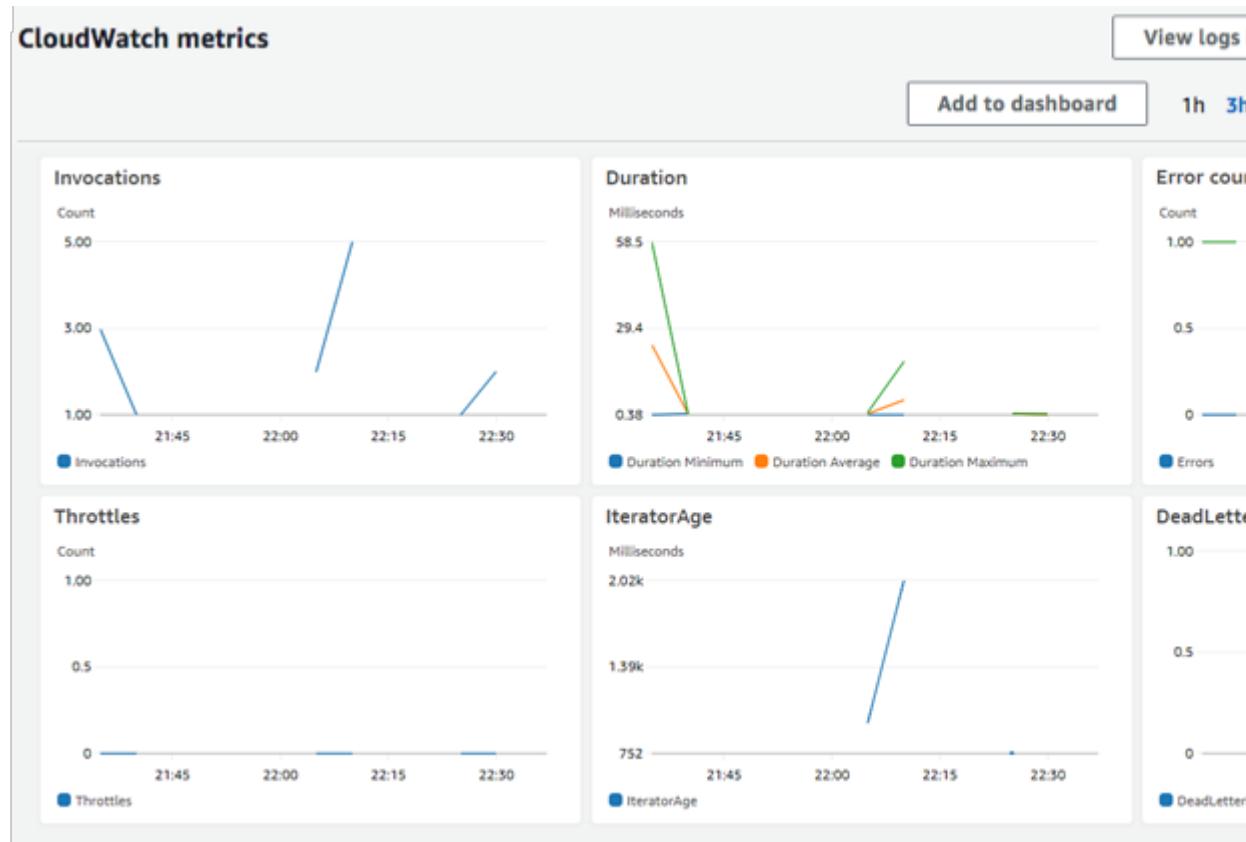
```
{  
    "key3": "value3",  
    "key2": "value2",  
    "key1": "value1"  
}
```

可以更改示例 JSON 中的键和值，但不要更改事件结构。如果您更改任何键和值，则必须相应更新示例代码。

3. 选择 Create (创建)，然后选择 Test (测试)。每个用户每个函数可以创建最多 10 个测试事件。这些测试事件不适用于其他用户。
4. AWS Lambda 代表您执行您的函数。您的 Lambda 函数中的 handler 接收并处理示例事件。
5. 成功执行后，在控制台中查看结果。
 - Execution result 部分将执行状态显示为 succeeded，还将显示由 return 语句返回的函数执行结果。
 - Summary 部分显示在 Log output 部分中报告的密钥信息（执行日志中的 REPORT 行）。
 - Log output 部分显示 AWS Lambda 针对每次执行生成的日志。这些是由 Lambda 函数写入到 CloudWatch 的日志。为方便起见，AWS Lambda 控制台为您显示了这些日志。

注意：Click here 链接在 CloudWatch 控制台中显示日志。然后，该函数在与 Lambda 函数对应的日志组中向 Amazon CloudWatch 添加日志。

6. 运行 Lambda 函数几次以收集您可在下一个步骤中查看的一些指标。
7. 选择 Monitoring。此页面显示了 Lambda 发送到 CloudWatch 的指标的图表。

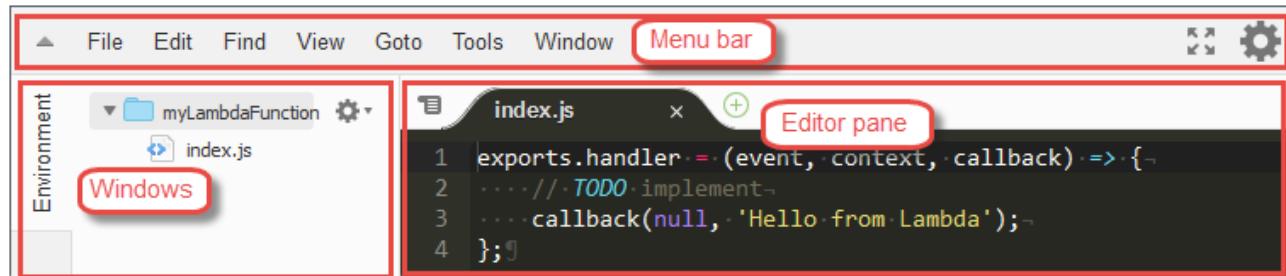


有关这些图表的更多信息，请参阅[在 AWS Lambda 控制台中监控函数 \(p. 367\)](#)。

使用 AWS Lambda 控制台编辑器创建函数

使用 AWS Lambda 控制台中的代码编辑器，您可以编写和测试您的 Lambda 函数代码并查看其执行结果。

该代码编辑器包含菜单栏、窗口和编辑器窗格。



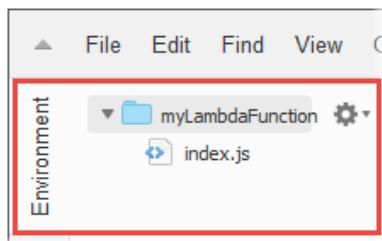
有关命令可执行的操作的列表，请参阅 AWS Cloud9 用户指南 中的[菜单命令参考](#)。请注意，该参考中列出的一些命令在代码编辑器中不可用。

主题

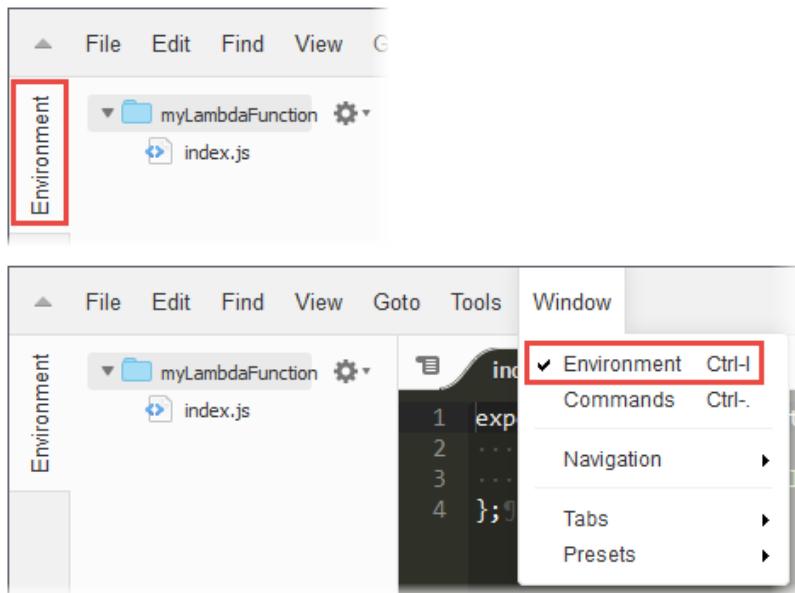
- [处理文件和文件夹 \(p. 6\)](#)
- [使用代码 \(p. 7\)](#)
- [在全屏模式下工作 \(p. 10\)](#)
- [使用首选项 \(p. 10\)](#)

处理文件和文件夹

您可以在代码编辑器中使用 Environment 窗口为您的函数创建、打开和管理文件。



要显示或隐藏“Environment”窗口，请选择 Environment 按钮。如果 Environment 按钮不可见，请选择菜单栏上的 Window、Environment。



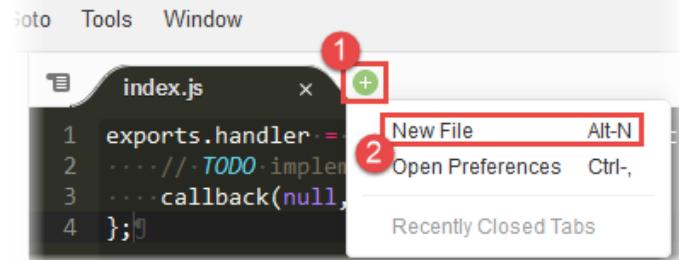
要打开单个文件并在编辑器窗格中显示其内容，请在 Environment 窗口中双击该文件。

要打开多个文件并在编辑器窗格中显示其内容，请在 Environment 窗口中选择这些文件。右键单击选定内容，然后选择 Open。

要创建新文件，请执行以下操作之一：

- 在 Environment 窗口中，右键单击您希望将新文件放入的文件夹，然后选择 New File。键入文件的名称和扩展名，然后按 Enter。
- 在菜单栏上选择 File、New File。当您准备好保存文件时，在菜单栏上选择 File、Save 或 File、Save As。然后，使用显示的 Save As 对话框命名文件并选择保存该文件的位置。

- 在编辑器窗格的选项卡按钮栏中，选择 + 按钮，然后选择 New File。当您准备好保存文件时，在菜单栏上选择 File、Save 或 File、Save As。然后，使用显示的 Save As 对话框命名文件并选择保存该文件的位置。



要创建新文件夹，请在 Environment 窗口中右键单击您希望将新文件夹放入的文件夹，然后选择 New Folder。键入文件夹名称，然后按 Enter。

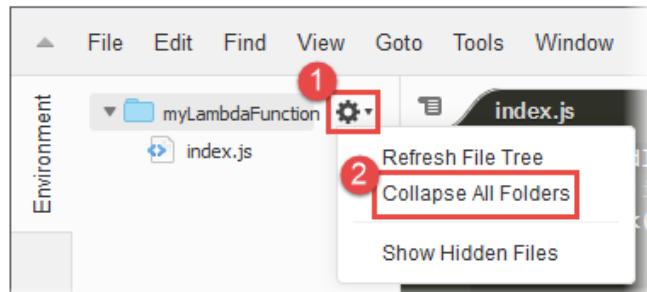
要保存某个文件，请在编辑器窗格中打开该文件并显示其内容，然后在菜单栏上选择 File、Save。

要重命名某个文件或文件夹，请在 Environment 窗口中右键单击该文件或文件夹。键入替换名称，然后按 Enter。

要删除文件或文件夹，请在 Environment 窗口中选择文件或文件夹。右键单击选定内容，然后选择 Delete。然后，通过选择 Yes (对于单个选定项) 或 Yes to All 来确认删除。

要剪切、拷贝、粘贴或复制文件或文件夹，请在 Environment 窗口中选择文件或文件夹。右键单击选定内容，然后相应地选择 Cut、Copy、Paste 或 Duplicate。

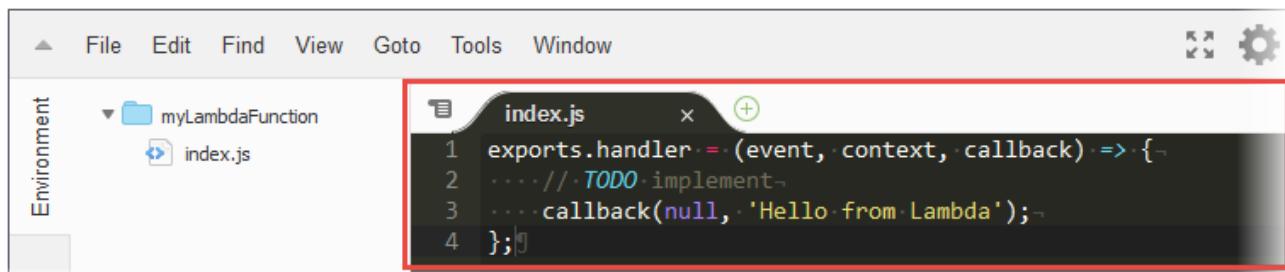
要折叠文件夹，请选择 Environment 窗口中的齿轮图标，然后选择 Collapse All Folders。



要显示已隐藏的文件，请选择 Environment 窗口中的齿轮图标，然后选择 Show Hidden Files。

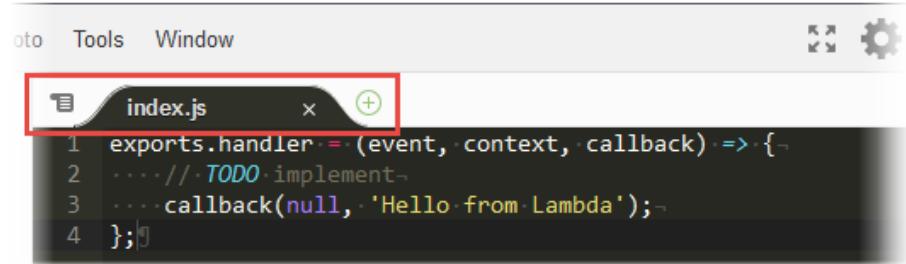
使用代码

使用代码编辑器中的编辑器窗格可以查看和编写代码。



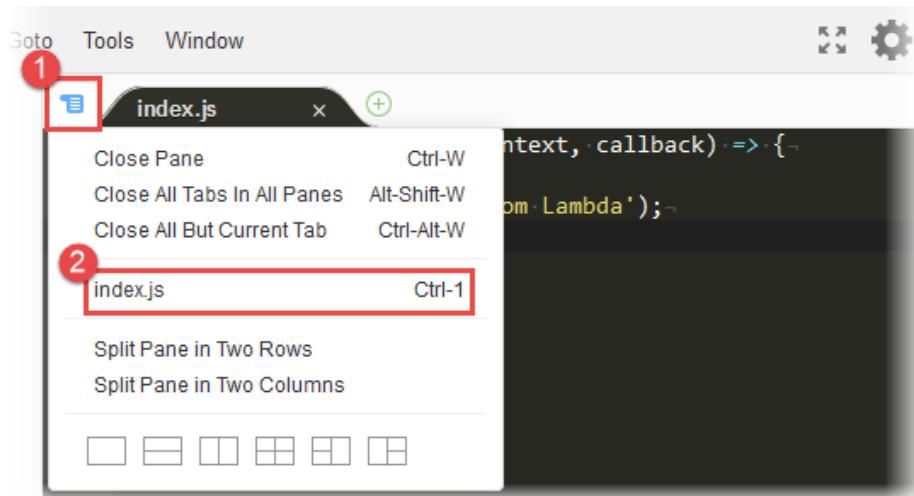
使用选项卡按钮

使用选项卡按钮栏可以选择、查看和创建文件。



要显示某个已打开文件的内容，请执行以下操作之一：

- 选择该文件的选项卡。
- 选择选项卡按钮栏中的下拉菜单按钮，然后选择文件的名称。



要关闭某个已打开文件，请执行下列操作之一：

- 选择该文件的选项卡中的 X 图标。
- 选择该文件的选项卡。接下来，选择选项卡按钮栏中的下拉菜单按钮，然后选择 Close Pane。

要关闭多个已打开的文件，请选择选项卡按钮栏中的下拉菜单，然后根据需要选择 Close All Tabs in All Panes 或 Close All But Current Tab。

要创建新文件，请选择选项卡按钮栏中的 + 按钮，然后选择 New File。当您准备好保存文件时，在菜单栏上选择 File、Save 或 File、Save As。然后，使用显示的 Save As 对话框命名文件并选择保存该文件的位置。

使用状态栏

使用状态栏可以快速移动到活动文件中的某一行并更改代码的显示方式。

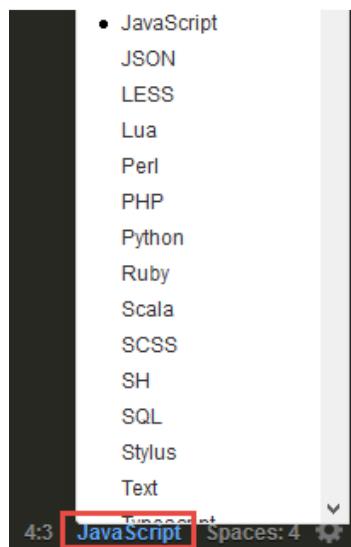
```
index.js x +  
1 exports.handler = (event, context, callback) => {  
2     // TODO implement  
3     callback(null, 'Hello from Lambda')  
4 };
```

4:3 JavaScript Spaces: 4

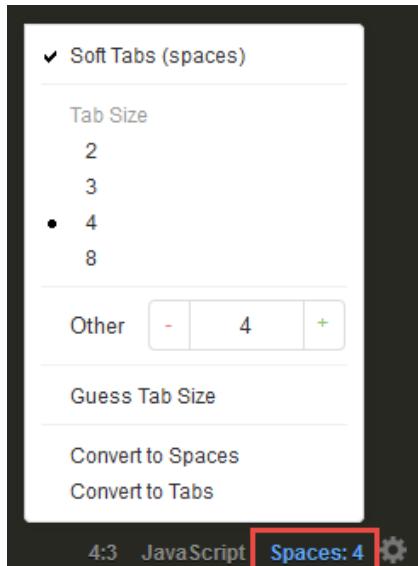
要快速移动到活动文件中的某一行，请选择行选择器，键入要转到的行号，然后按 Enter。

4:3 JavaScript Spaces: 4

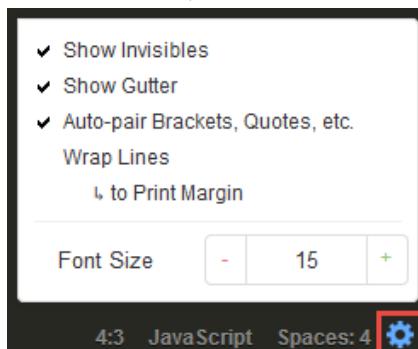
要更改活动文件中的代码颜色方案，请选择代码颜色方案选择器，然后选择新的代码颜色方案。



要更改在活动文件中是否使用软制表符或空格，或者是否转换为空格或制表符，请选择空格和制表符选择器，然后选择新设置。



要为所有文件更改显示还是隐藏不可见字符或间距、自动配对括号或引号、换行符，或者更改字体大小，请选择齿轮图标，然后选择新设置。



在全屏模式下工作

您可以展开代码编辑器，以获得更多的空间来处理您的代码。

要将代码编辑器展开到 Web 浏览器窗口的边缘，请在菜单栏中选择 Toggle fullscreen 按钮。



要将代码编辑器缩小到其原始大小，请再次选择 Toggle fullscreen 按钮。

在全屏模式下，将在菜单栏上显示额外的选项：Save (保存) 和 Test (测试)。选择 Save 可以保存函数代码。选择 Test 或 Configure Events 可以创建或编辑函数的测试事件。

使用首选项

您可以更改各种代码编辑器设置，例如显示哪些编码提示和警告，代码折叠行为，代码自动完成行为以及其他功能。

要更改代码编辑器设置，请在菜单栏中选择 Preferences 齿轮图标。



有关这些设置的作用的列表，请参阅 AWS Cloud9 用户指南中的以下参考。

- 您可以执行的项目设置更改
- 您可以执行的用户设置更改

请注意，这些参考中列出的一些设置在代码编辑器中不可用。

将 AWS Lambda 与 AWS Command Line Interface 结合使用

您可以使用 AWS Command Line Interface 管理函数及其他 AWS Lambda 资源。AWS CLI 使用 AWS SDK for Python (Boto) 与 Lambda API 进行交互。您可以使用它来了解 API，并在构建将 Lambda 与 AWS 开发工具包结合使用的应用程序时运用所学的知识。

在本教程中，您将使用 AWS CLI 管理和调用 Lambda 函数。

先决条件

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command  
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以[安装 Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

本教程使用 AWS Command Line Interface (AWS CLI) 调用服务 API 操作。要安装 AWS CLI，请参阅 AWS Command Line Interface 用户指南 中的[安装 AWS CLI](#)。

创建执行角色

[创建执行角色 \(p. 30\)](#)，向您的函数授予访问 AWS 资源的权限。要使用 AWS CLI 创建执行角色，请使用 `create-role` 命令。

```
$ aws iam create-role --role-name lambda-ex --assume-role-policy-document file://trust-policy.json  
{  
    "Role": {  
        "Path": "/",  
        "RoleName": "lambda-ex",  
        "RoleId": "AROAQFOXMP6TZ6ITKWND",  
        "Arn": "arn:aws:iam::123456789012:role/lambda-ex",  
        "CreateDate": "2020-01-17T23:19:12Z",  
        "AssumeRolePolicyDocument": {  
            "Version": "2012-10-17",  
            "Statement": [  
                {  
                    "Effect": "Allow",  
                    "Action": "lambda:InvokeFunction",  
                    "Resource": "arn:aws:lambda:us-east-1:123456789012:function:lambda-ex"  
                }  
            ]  
        }  
    }  
}
```

```
        "Principal": {
            "Service": "lambda.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    }
}
}
```

trust-policy.json 文件是当前目录中的 JSON 文件，该文件定义了角色的[信任策略](#)。此信任策略通过向服务委托人授予调用 AWS Security Token Service AssumeRole 操作所需的 lambda.amazonaws.com 权限来允许 Lambda 使用角色的权限。

Example trust-policy.json

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

您也可以内联方式指定信任策略。JSON 字符串中转义引号的要求因您的 Shell 而异。

```
$ 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"}}]'
```

要向角色添加权限，请使用 attach-policy-to-role 命令。首先，添加 AWSLambdaBasicExecutionRole 托管策略。

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

AWSLambdaBasicExecutionRole 策略具有函数将日志写入 CloudWatch Logs 所需的权限。

创建函数

以下示例记录环境变量和事件对象的值。

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
}
```

创建函数

1. 将示例代码复制到名为 index.js 的文件中。

2. 创建部署程序包。

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

3. 使用 `create-function` 命令创建 Lambda 函数。将角色 ARN 中突出显示的文本替换为您的账户 ID。

```
$ 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
{
    "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",
    ...
}
```

要从命令行获取调用的日志，请使用 `--log-type` 选项。响应包含一个 `LogResult` 字段，该字段包含来自调用的多达 4 KB 的 base64 编码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

您可以使用 `base64` 实用程序来解码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
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: 100 ms          Memory Size: 128 MB      Max Memory Used: 73 MB
```

`base64` 实用工具在 Linux、macOS 和 [Ubuntu on Windows](#) 上可用。对于 macOS，命令为 `base64 -D`。

要从命令行获取完整的日志事件，您可以在函数输出中包含日志流名称，如上例中所示。以下示例脚本调用名为 `my-function` 的函数并下载最后 5 个日志事件。

Example `get-logs.sh` 脚本

此示例要求 `my-function` 返回日志流 ID。

```
#!/bin/bash
aws lambda invoke --function-name my-function --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
```

此脚本使用 sed 从输出文件中删除引号，并休眠 15 秒以等待日志可用。输出包括来自 Lambda 的响应，以及来自 get-log-events 命令的输出。

```
$ ./get-logs.sh
{
    "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: 100 ms\tMemory Size: 128 MB\tMax Memory Used: 75 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

列出您的账户中的 Lambda 函数

执行以下 AWS CLI list-functions 命令可检索您已创建的函数的列表。

```
$ aws lambda list-functions --max-items 10
{
    "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": "eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxMHO="
}
```

作为响应，Lambda 返回一个最多包含 10 个函数的列表。如果有更多功能可供您检索，`NextToken` 将提供一个您可以在下一个 `list-functions` 请求中使用的标记。以下 `list-functions` AWS CLI 命令是一个演示 `--starting-token` 参数的示例。

```
$ aws lambda list-functions --max-items 10 --starting-token eyJNYXJrZXIiOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxMHO=
```

检索 Lambda 函数

Lambda CLI `get-function` 命令将返回 Lambda 函数元数据以及可用来下载函数的部署程序包的预签名 URL。

```
$ aws lambda get-function --function-name my-function
{
  "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-..."
  }
}
```

有关更多信息，请参阅 [GetFunction \(p. 455\)](#)。

清除

执行以下 `delete-function` 命令以删除 `my-function` 函数。

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

删除您在 IAM 控制台中创建的 IAM 角色。有关删除角色的信息，请参阅 IAM 用户指南 中的[删除角色或实例配置文件](#)。

AWS Lambda 概念

利用 AWS Lambda，您可以运行函数以处理事件。您可以通过使用 Lambda API 调用函数或将 AWS 服务或资源配置为调用函数来向函数发送事件。

概念

- [功能 \(p. 16\)](#)
- [Runtime \(p. 16\)](#)
- [Event \(p. 16\)](#)
- [并发 \(p. 17\)](#)
- [Trigger \(p. 17\)](#)

功能

函数是一个资源，您可以调用它来在 AWS Lambda 中运行您的代码。一个函数具有处理事件的代码，以及在 Lambda 与函数代码之间传递请求和响应的运行时。您负责提供代码，并且可以使用提供的运行时或创建自己的运行时。

有关更多信息，请参阅[管理 AWS Lambda 函数 \(p. 47\)](#)。

Runtime

Lambda 运行时允许不同语言的函数在同一基本执行环境中运行。将您的函数配置为使用与您的编程语言匹配的运行时。运行时位于 Lambda 服务和函数代码之间，并在二者之间中继调用事件、上下文信息和响应。您可以使用 Lambda 提供的运行时，或构建您自己的运行时。

有关更多信息，请参阅[AWS Lambda 运行时 \(p. 108\)](#)。

Event

事件是 JSON 格式的文档，其中包含要处理的函数的数据。Lambda 运行时将事件转换为一个对象，并将该对象传递给函数代码。在调用函数时，可以确定事件的结构和内容。

Example 自定义事件 – 天气数据

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

当 AWS 服务调用您的函数时，该服务会定义事件的形状。

Example 服务事件 – Amazon SNS 通知

```
{  
    "Records": [  
        {  
            "Sns": {  
                "Timestamp": "2019-01-02T12:45:07.000Z",  
                "Signature": "tcc6faL2yUC6dgZdmrwh1Y4cGa/ebXEkAi6RibDsvpi+tE/1+82j...65r==",  
                "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",  
                "TopicArn": "arn:aws:sns:us-east-1:123456789012:MyTopic"  
            }  
        }  
    ]  
}
```

```
"Message": "Hello from SNS!",  
...
```

有关来自 AWS 服务的事件的详细信息，请参阅[将 AWS Lambda 与其他服务结合使用 \(p. 138\)](#)。

并发

并发性是您的函数在任何给定时间所服务于的请求的数目。在调用函数时，Lambda 会预配置其实例以处理事件。当函数代码完成运行时，它会处理另一个请求。如果当仍在处理请求时再次调用函数，则预配置另一个实例，从而增加该函数的并发性。

并发性受区域级别限制的约束。您还可以配置单个函数来限制其并发性，或确保它们能够达到特定级别的并发性。有关更多信息，请参阅[管理 Lambda 函数的并发 \(p. 54\)](#)。

Trigger

触发器是调用 Lambda 函数的资源或配置。这包括可配置为调用函数的 AWS 服务、您开发的应用程序以及事件源映射。事件源映射是 Lambda 中的一种资源，它从流或队列中读取项目并调用函数。

有关更多信息，请参阅[调用 AWS Lambda 函数 \(p. 81\)](#)和[将 AWS Lambda 与其他服务结合使用 \(p. 138\)](#)。

AWS Lambda 功能

AWS Lambda 提供了用于管理和调用函数的管理控制台和 API。它提供的运行时支持一组标准功能，以便您能够根据需要轻松地在语言和框架之间进行切换。除了函数之外，您还可以创建版本、别名、层和自定义运行时。

功能

- [编程模型 \(p. 17\)](#)
- [部署程序包 \(p. 19\)](#)
- [层 \(p. 19\)](#)
- [扩展 \(p. 19\)](#)
- [并发控制 \(p. 20\)](#)
- [异步调用 \(p. 22\)](#)
- [事件源映射 \(p. 23\)](#)
- [目标 \(p. 24\)](#)
- [函数蓝图 \(p. 25\)](#)
- [应用程序模板 \(p. 25\)](#)

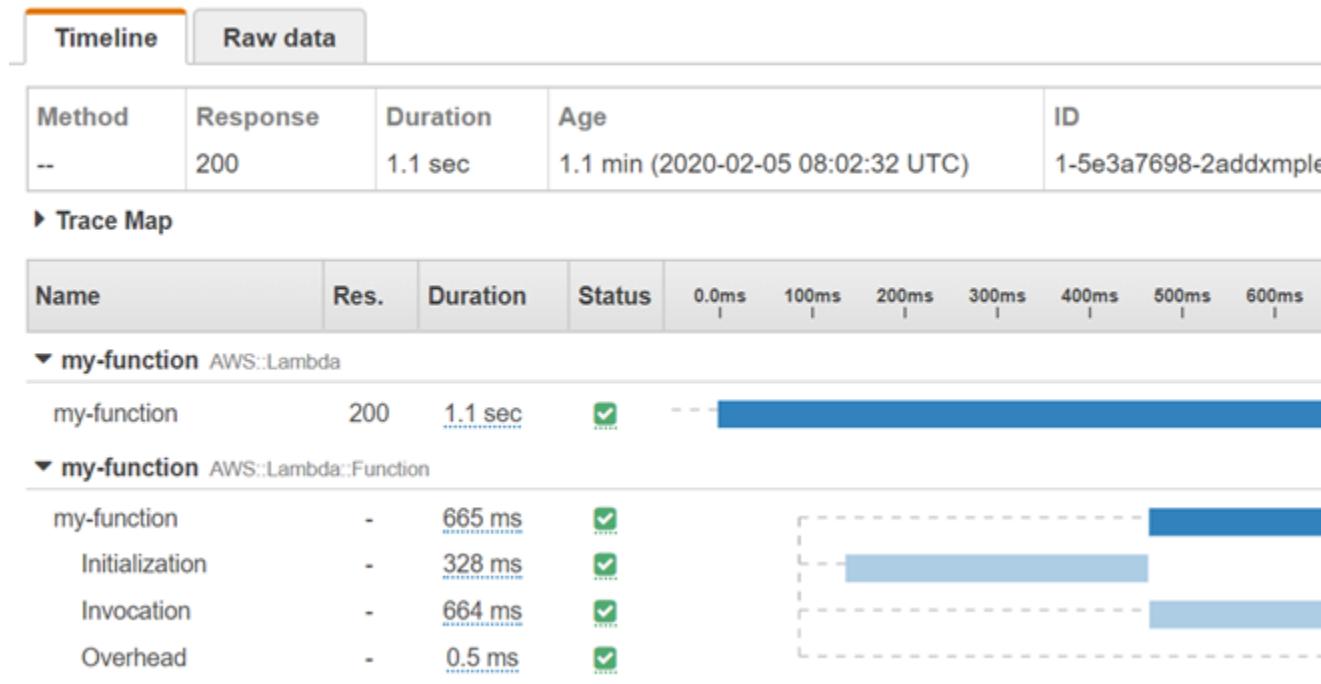
编程模型

代码编写细节因运行时而异，但所有运行时都共用一个通用的编程模型，该模型定义了代码与运行时代码之间的接口。通过在函数配置中定义处理程序来告诉运行时运行哪个方法，然后运行时会运行该方法。运行时将对象（例如函数名和请求 ID）传递给包含调用事件以及上下文的处理程序。

在处理程序完成第一个事件的处理后，运行时会向处理程序发送另一个事件。函数的类保留在内存中，因此，可以重用初始化代码中在处理程序方法外部声明的客户端和变量。要节省后续事件的处理时间，请在初始化期间创建可重用的资源，如 AWS 开发工具包客户端。在初始化后，函数的每个实例都可以处理数千个请求。

初始化将作为函数实例所处理的首次调用的持续时间的一部分计费。在启用 X-Ray 跟踪 (p. 371) 时，运行时会记录初始化和执行的单独子分段。

Traces > Details



此外，您的函数有权访问 /tmp 目录中的本地存储。正在处理请求的函数实例在被回收之前会保持活动状态几个小时。

运行时捕获来自函数的日志记录输出，并将它发送到 Amazon CloudWatch Logs。除了记录函数的输出外，运行时还会在执行开始和结束时记录条目。这包括具有请求 ID、计费持续时间、初始化持续时间和其他详细信息的报告日志。如果函数引发一个错误，则运行时会将该错误返回到调用程序。

Note

日志记录需要遵循 CloudWatch Logs 限制。由于节流或在某些情况下函数实例停止，日志数据可能会丢失。

有关您常用的编程语言中的编程模式的实际操作介绍，请参阅以下章节。

- 使用 Node.js 构建 Lambda 函数 (p. 265)
- 使用 Python 构建 Lambda 函数 (p. 277)
- 使用 Ruby 构建 Lambda 函数 (p. 291)
- 使用 Java 构建 Lambda 函数 (p. 301)
- 使用 Go 构建 Lambda 函数 (p. 330)
- 使用 C# 构建 Lambda 函数 (p. 342)
- 使用 PowerShell 构建 Lambda 函数 (p. 359)

Lambda 通过在需求增加时运行其他实例并在需求减少时停止实例来扩展您的函数。除非另有说明，否则传入请求可能会不按次序处理或同时处理。将应用程序的状态存储在其他服务中，并且不依赖长期存在的函数实例。虽然可使用本地存储和类级别对象来提高性能，但应将部署包的大小和传输到执行环境中的数据量保持在最小值。

部署程序包

您的函数代码由脚本或编译的程序及其依赖项组成。在 Lambda 控制台或工具包中编写函数时，客户端会创建代码的 ZIP 存档（称为部署包）。然后，客户端将该部署包发送到 Lambda 服务。使用 Lambda API、命令行工具或软件开发工具包管理函数时，您可以创建部署包。您还需要为编译语言手动创建部署包，并为您的函数添加依赖项。

有关特定于语言的说明，请参阅以下主题。

- [Node.js 中的 AWS Lambda 部署程序包 \(p. 268\)](#)
- [Python 中的 AWS Lambda 部署程序包 \(p. 279\)](#)
- [Ruby 中的 AWS Lambda 部署程序包 \(p. 293\)](#)
- [Java 中的 AWS Lambda 部署程序包 \(p. 302\)](#)
- [Go 中的 AWS Lambda 部署程序包 \(p. 330\)](#)
- [C# 中的 AWS Lambda 部署程序包 \(p. 342\)](#)
- [PowerShell 中的 AWS Lambda 部署程序包 \(p. 360\)](#)

层

Lambda 层是适用于库、自定义运行时和其他函数依赖项的分配机制。利用层，您可独立于其使用的不变代码和资源来管理开发中的函数代码。您可以将函数配置为使用您创建的层、AWS 提供的层或来自其他 AWS 客户的层。

有关更多信息，请参阅 [AWS Lambda 层 \(p. 68\)](#)。

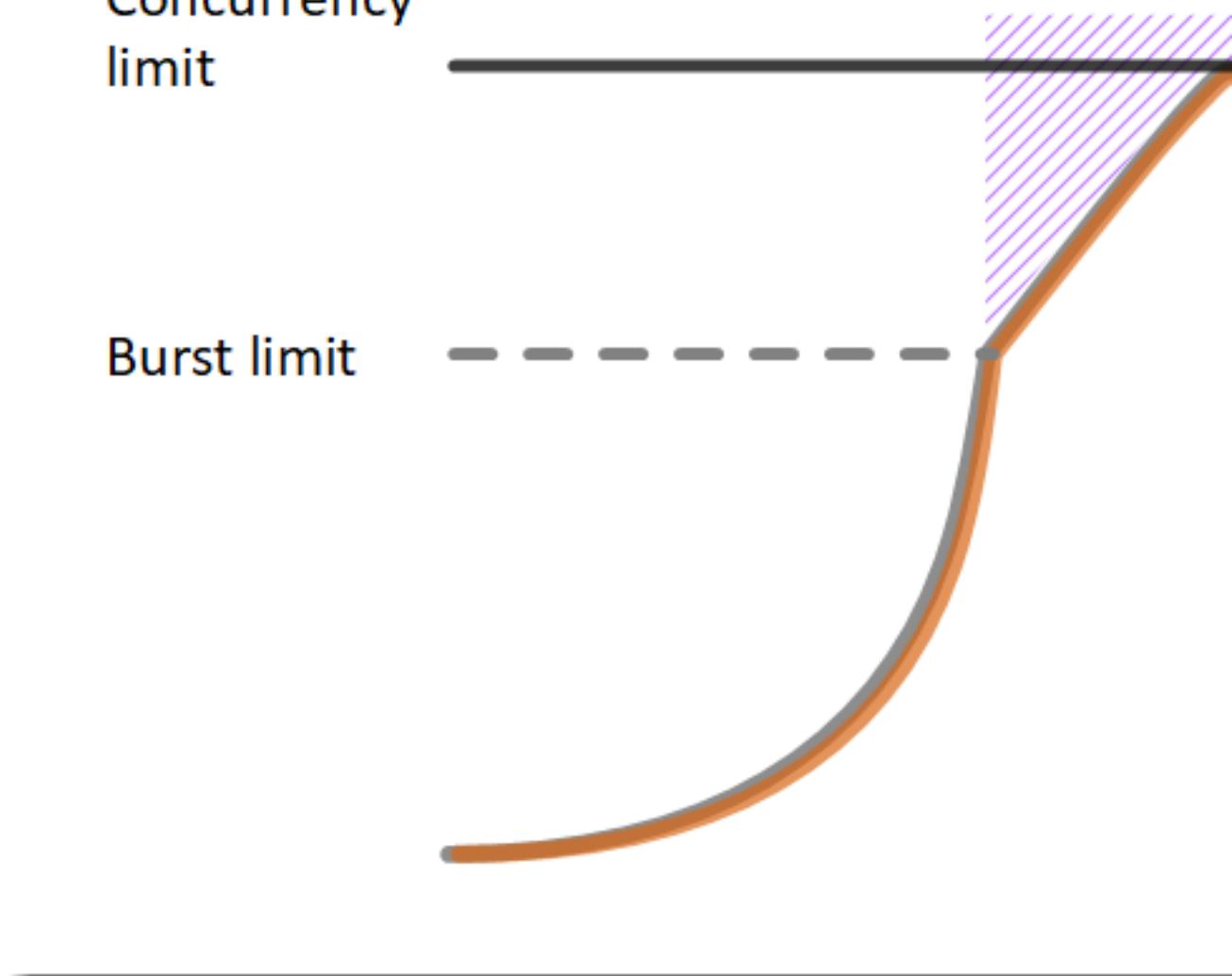
扩展

Lambda 管理运行代码的基础设施，并且会自动扩展以响应传入请求。当您的函数的调用速度快于函数的单个实例可处理事件的速度时，Lambda 会通过运行其他实例来进行扩展。当流量减少时，不活动的实例将被冻结或停止。您只需为函数初始化或处理事件的时间付费。

Function Scaling with Concurrency Limit

Concurrency
limit

Burst limit

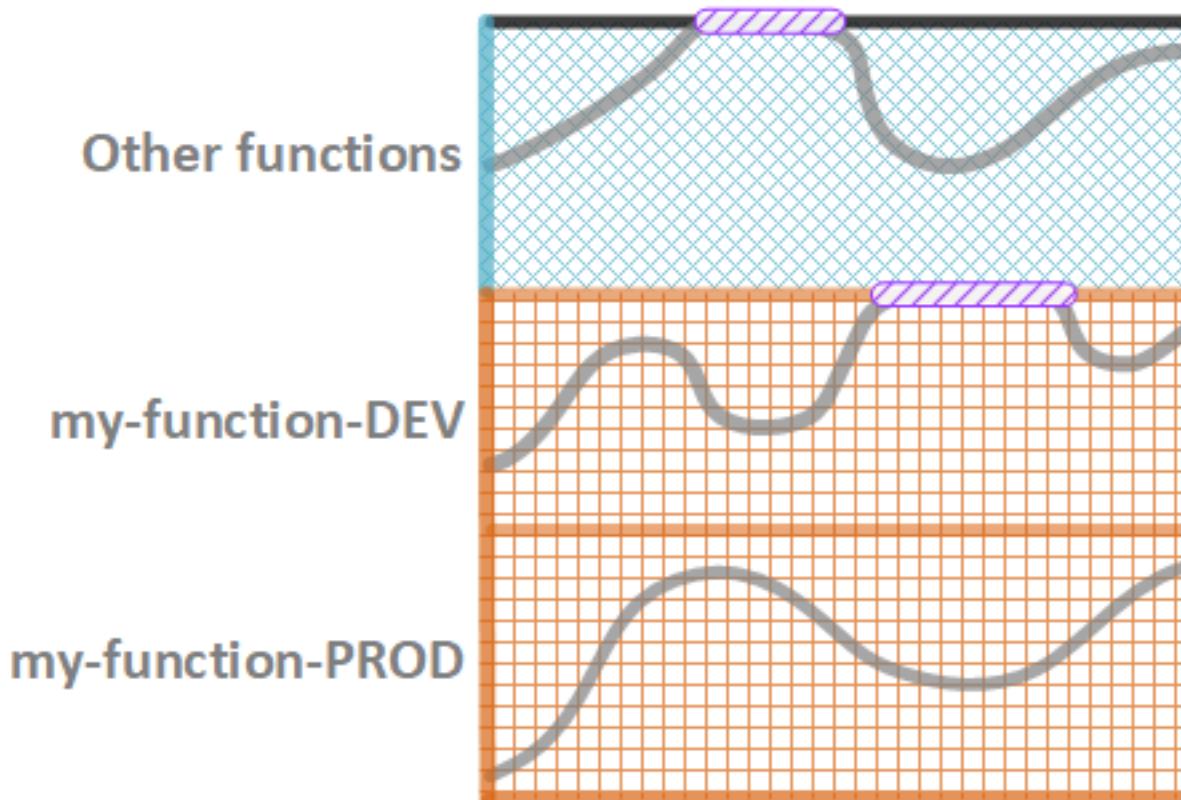


有关更多信息，请参阅 AWS Lambda 函数扩展 (p. 94)。

并发控制

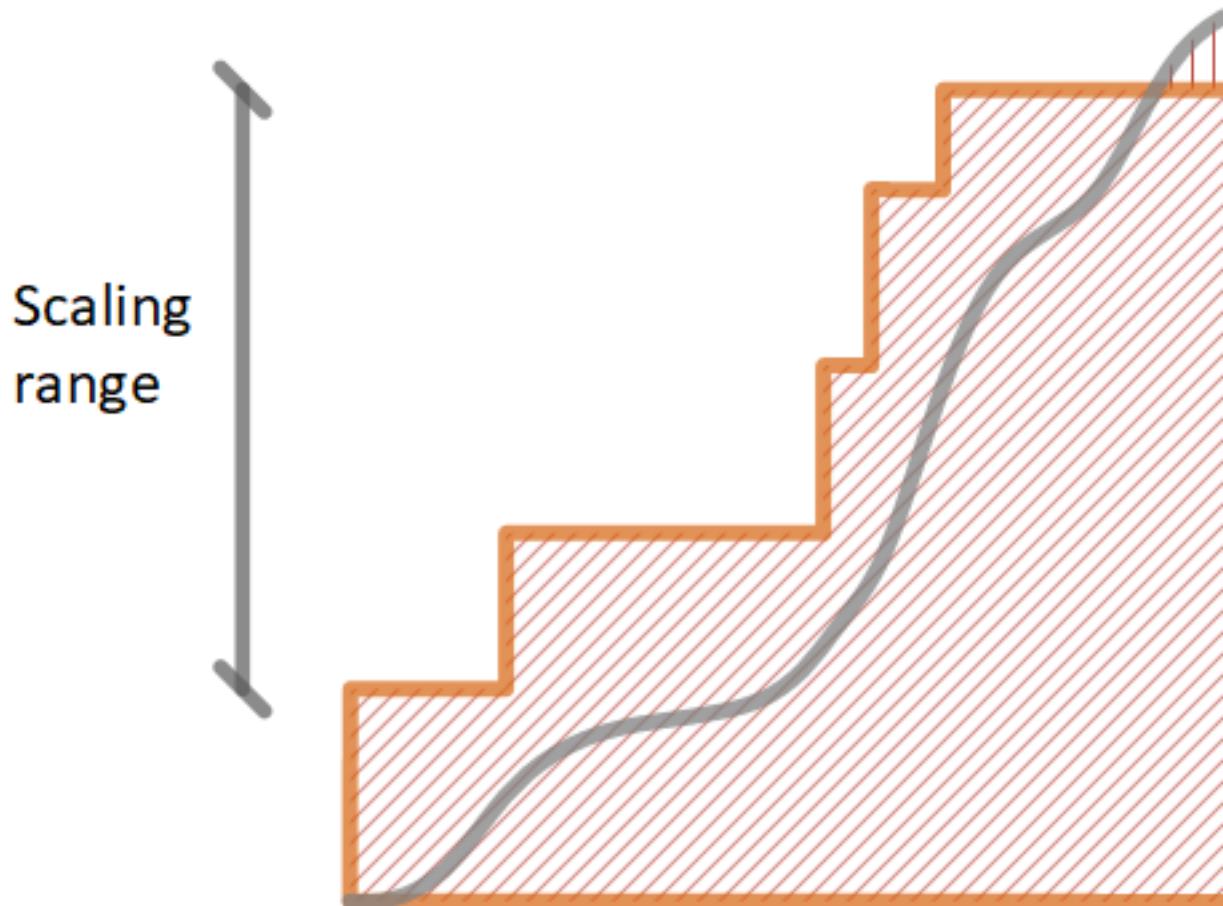
使用并发设置来确保您的生产应用程序实现高可用性和快速响应。为了防止函数使用太多并发，并为函数预留账户可用并发的一部分，请使用预留并发。预留并发将可用并发池拆分为子集。具有预留并发的函数仅使用其专用池中的并发。

Reserved Concurrency



要使函数能够在延迟不发生波动的情况下进行扩展，请使用预配置并发。对于需要很长时间才能初始化的函数，或者所有调用需要极低延迟的函数，预配置并发使您能够预先初始化函数的实例并保持它们始终运行。Lambda 与 Application Auto Scaling 集成以支持基于使用率的预配置并发自动扩展。

Autoscaling with Provisioned Concurrency

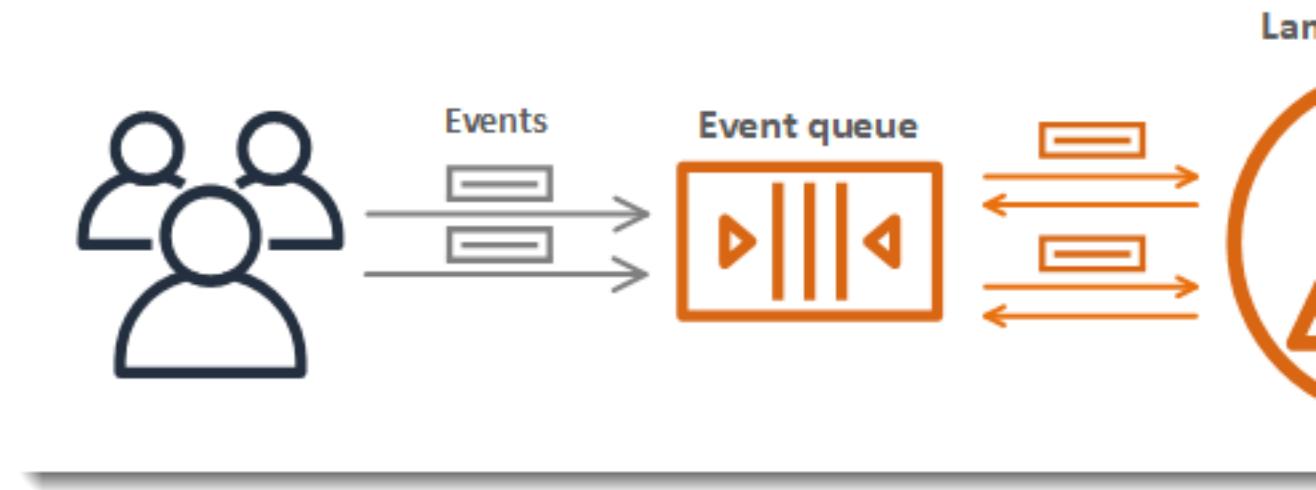


有关更多信息，请参阅管理 Lambda 函数的并发 (p. 54)。

异步调用

调用函数时，您可以选择同步或异步调用。使用[同步调用 \(p. 81\)](#)时，您将等待函数处理该事件并返回响应。使用[异步调用](#)时，Lambda 会将事件排队等待处理并立即返回响应。

Asynchronous Invocation



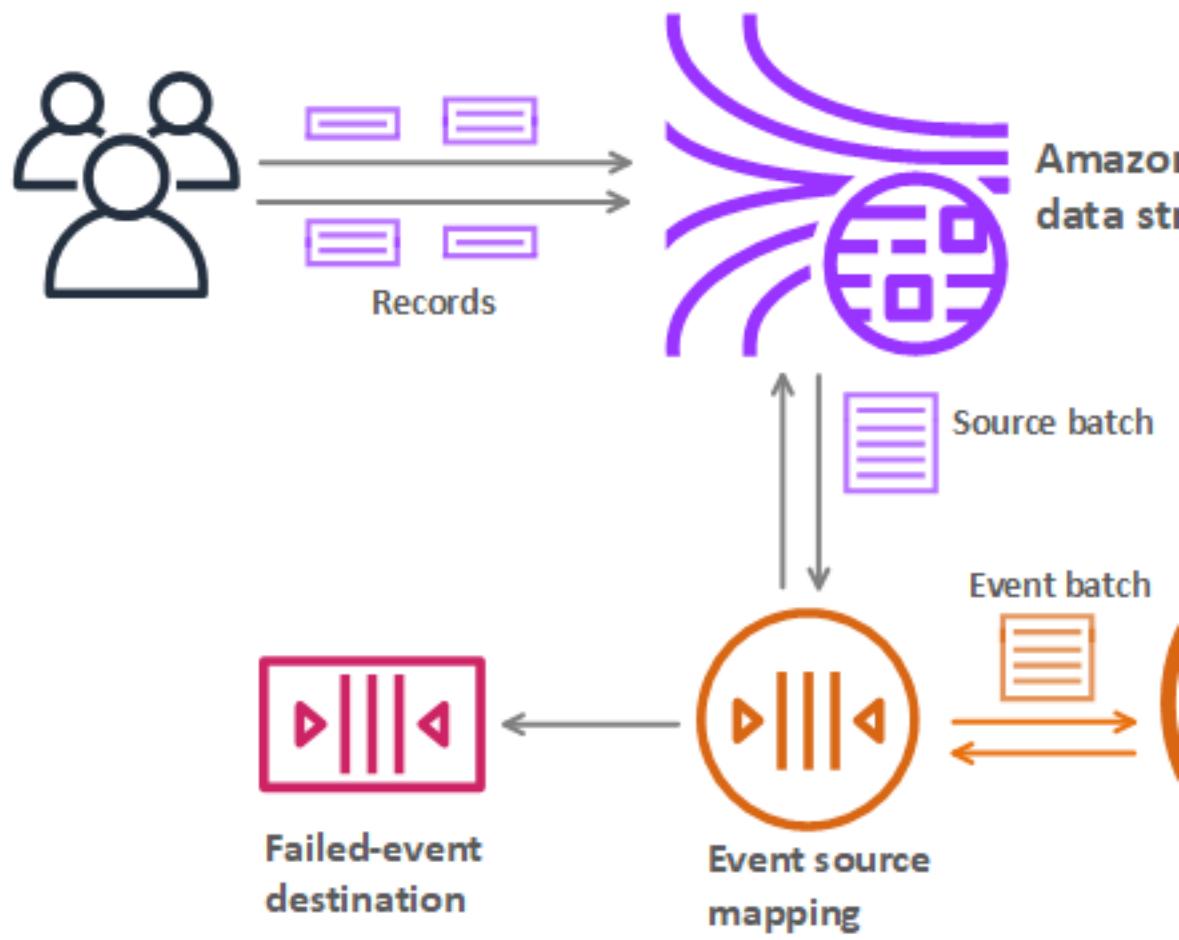
对于异步调用，如果函数返回错误或被限制，则 Lambda 处理重试。要自定义此行为，您可以在函数、版本或别名上配置错误处理设置。您还可以配置 Lambda 以将处理失败的事件发送到死信队列，或将任何调用的记录发送到目标 (p. 24)。

有关更多信息，请参阅异步调用 (p. 83)。

事件源映射

要处理流或队列中的项，您可以创建事件源映射 (p. 90)。事件源映射是 Lambda 中的一个资源，它从 Amazon SQS 队列、Amazon Kinesis 流或 Amazon DynamoDB 流中读取项目，并将它们批量发送到您的函数。您的函数处理的每个事件可以包含数百个或数千个项。

Event Source Mapping with Kinesis Stream



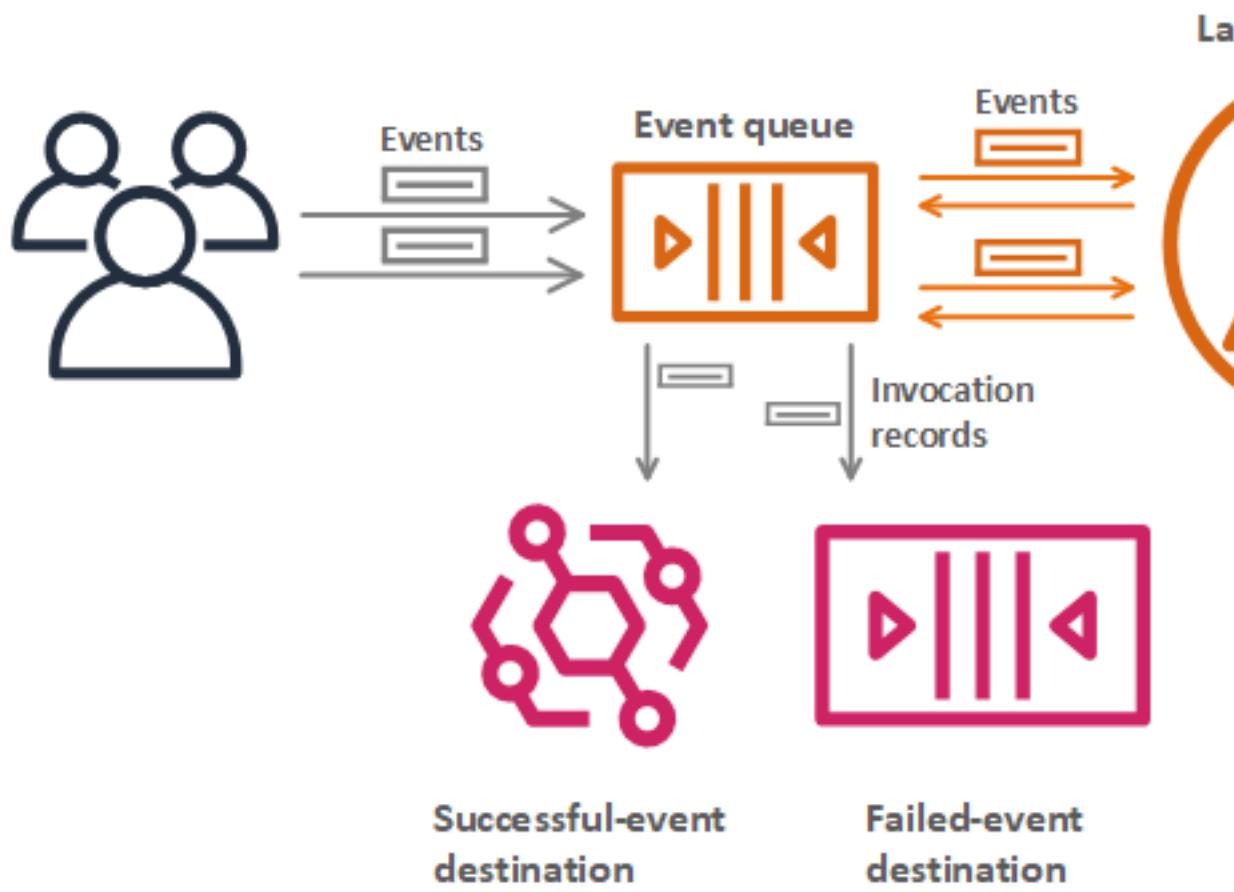
事件源映射维护未处理项目的本地队列，并在函数返回错误或受到限制时处理重试。您可以配置事件源映射以自定义批处理行为和错误处理，或者将处理失败的项目记录发送到[目标 \(p. 24\)](#)。

有关更多信息，请参阅[AWS Lambda 事件源映射 \(p. 90\)](#)。

目标

目标是接收函数调用记录的 AWS 资源。对于[异步调用 \(p. 22\)](#)，您可以配置 Lambda 以将调用记录发送到队列、主题、函数或事件总线。您可以为成功调用和处理失败的事件配置单独的目标。调用记录包含有关事件、函数的响应和记录发送原因的详细信息。

Destinations for Asynchronous Invocation



对于从流读取的[事件源映射 \(p. 23\)](#)，您可以配置 Lambda 以将处理失败的批处理记录发送到队列或主题。事件源映射的失败记录包含有关批处理的元数据，并指向流中的项目。

有关更多信息，请参阅[将 AWS Lambda 与 Amazon DynamoDB 结合使用 \(p. 184\)](#)和[将 AWS Lambda 与 Amazon Kinesis 结合使用 \(p. 212\)](#)的[配置异步调用目标 \(p. 85\)](#)及错误处理部分。

函数蓝图

当您在 Lambda 控制台中创建函数时，可以选择从头开始、使用蓝图或者从 [AWS Serverless Application Repository](#) 中部署应用程序。蓝图提供了示例代码，展示如何将 Lambda 与 AWS 服务或流行第三方应用程序结合使用。蓝图包括 Node.js 和 Python 运行时的示例代码和函数配置预设。

蓝图是根据[无权利保留协议](#)许可证提供给用户使用的。它们仅在 Lambda 控制台中提供。

应用程序模板

您可以使用 Lambda 控制台创建具有持续交付管道的应用程序。Lambda 控制台中的应用程序模板包括一个或多个函数的代码、一个用于定义函数和提供支持的 AWS 资源的应用程序模板，以及一个定义 AWS

CodePipeline 管道的基础设施模板。管道具有构建和部署阶段，在每次将更改推送到包含的 Git 存储库时，都会运行这些阶段。

应用程序模板是根据 [MIT 无署名归属](#) 许可证提供给用户使用的。它们仅在 Lambda 控制台中提供。

有关更多信息，请参阅在 [Lambda 控制台中创建具有持续交付功能的应用程序](#) (p. 122)。

与 AWS Lambda 一起使用的工具

除了 Lambda 控制台之外，您还可以使用以下工具来管理和调用 Lambda 资源。

工具

- [AWS Command Line Interface \(p. 26\)](#)
- [AWS 无服务器应用程序模型 \(p. 26\)](#)
- [SAM CLI \(p. 26\)](#)
- [代码编写工具 \(p. 27\)](#)

AWS Command Line Interface

安装 AWS Command Line Interface 以从命令行管理和使用 Lambda 函数。本指南中的教程使用 AWS CLI，其中包含用于所有 Lambda API 操作的命令。一些功能在 Lambda 控制台中不可用，只能通过 AWS CLI 或 AWS 开发工具包访问。

要设置 AWS CLI，请参阅 [AWS Command Line Interface 用户指南](#) 中的以下主题。

- [开始设置 AWS Command Line Interface](#)
- [配置 AWS Command Line Interface](#)

要验证 AWS CLI 是否配置正确，请运行 `list-functions` 命令以查看当前区域中的 Lambda 函数的列表。

```
$ aws lambda list-functions
```

AWS 无服务器应用程序模型

AWS SAM 是 AWS CloudFormation 模板语言的扩展，可以让您在较高级别定义无服务器应用程序。它消除了函数角色创建等常见任务，更加便于编写模板。AWS SAM 受到 AWS CloudFormation 的直接支持，并且可以通过 AWS CLI 和 AWS SAM CLI 包含额外的功能。

有关 AWS SAM 模板的更多信息，请参阅 [AWS 无服务器应用程序模型 开发人员指南](#) 中的 [AWS SAM 模板基础知识](#)。

SAM CLI

AWS SAM CLI 是一个单独的命令行工具，您可以用它来管理和测试 AWS SAM 应用程序。除了用于上传构件和启动 AWS CloudFormation 堆栈的命令（这些命令同样在 AWS CLI 中提供）之外，SAM CLI 还提供了额外的命令，这些命令可用于验证模板并在 Docker 容器中本地运行应用程序。

要设置 AWS SAM CLI，请参阅 [AWS 无服务器应用程序模型 开发人员指南](#) 中的 [安装 AWS SAM CLI](#)。

代码编写工具

您可以使用 AWS Lambda 所支持的语言编写 Lambda 函数代码。有关受支持的语言的列表，请参阅[AWS Lambda 运行时 \(p. 108\)](#)。有许多可用于编写代码的工具，例如，AWS Lambda 控制台、Eclipse IDE 和 Visual Studio IDE。不过，可用的工具和选项取决于：

- 您选择用来编写 Lambda 函数代码的语言。
- 代码中使用的库。AWS Lambda 运行时提供了一些库，您必须上传您使用的任何其他库。

下表列出了可使用的语言、可用工具和选项。

语言	用于编写代码的工具和选项
Node.js	<ul style="list-style-type: none">• AWS Lambda 控制台• Visual Studio，带 IDE 插件（请参阅Visual Studio 中的 AWS Lambda 支持）• 您自己的编写环境
Java	<ul style="list-style-type: none">• Eclipse，带 AWS Toolkit for Eclipse（请参阅将 AWS Lambda 与 AWS Toolkit for Eclipse 结合使用）• IntelliJ，带 AWS Toolkit for IntelliJ• 您自己的编写环境
C#	<ul style="list-style-type: none">• Visual Studio，带 IDE 插件（请参阅Visual Studio 中的 AWS Lambda 支持）• .NET Core（请参阅.NET Core 安装指南）• 您自己的编写环境
Python	<ul style="list-style-type: none">• AWS Lambda 控制台• PyCharm，带 AWS Toolkit for PyCharm• 您自己的编写环境
Ruby	<ul style="list-style-type: none">• AWS Lambda 控制台• 您自己的编写环境
Go	<ul style="list-style-type: none">• 您自己的编写环境
PowerShell	<ul style="list-style-type: none">• 您自己的编写环境• PowerShell Core 6.0（请参阅安装 PowerShell Core）• .NET Core 3.1 开发工具包（请参阅.NET 下载）• AWSLambdaPSCore 模块（请参阅PowerShell 库）

AWS Lambda 限制

AWS Lambda 将限制可用来运行和存储函数的计算和存储资源量。以下限制按区域应用，并且可以提高这些限制。要请求提高限制，请使用[支持中心控制台](#)。

资源	默认限制
并发执行	1,000

资源	默认限制
函数和层存储	75 GB
每个 VPC 的弹性网络接口数 (p. 72)	250

有关并发以及 Lambda 如何扩展您的函数并发以响应流量的详细信息，请参阅 [AWS Lambda 函数扩展 \(p. 94\)](#)。

以下限制适用于函数配置、部署和执行。无法对其进行更改。

资源	限制
函数内存分配 (p. 47)	128 MB 到 3,008 MB，以 64 MB 为增量。
函数超时 (p. 47)	900 秒 (15 分钟)
函数环境变量 (p. 49)	4 KB
函数基于资源的策略 (p. 33)	20 KB
函数层 (p. 68)	5 层
函数突增并发 (p. 94)	500 - 3000 (每个区域各不相同 (p. 94))
每个区域的调用频率 (每秒请求数)	10 倍并发执行限制 (同步 (p. 81) – 所有资源) 10 倍并发执行限制 (异步 (p. 83) – 非 AWS 资源) 无限制 (异步 – AWS 服务资源 (p. 138))
每个函数版本或别名的调用频率 (每秒请求数)	10 x 分配的 预配置并发 (p. 54) 此限制仅适用于使用预配置并发的函数。
调用负载 (p. 81) (请求和响应)	6 MB (同步) 256 KB (异步)
部署程序包 (p. 19) 大小	50 MB (已压缩，可直接上传) 250 MB (解压缩，包括层) 3 MB (控制台编辑器)
测试事件 (控制台编辑器)	10
/tmp 目录存储	512 MB
文件描述符	1,024
执行进程/线程	1,024

其他服务的限制（如 AWS Identity and Access Management、Amazon CloudFront (Lambda@Edge) 和 Amazon Virtual Private Cloud）会影响您的 Lambda 函数。有关更多信息，请参阅 [AWS 服务限制](#) 和 [将 AWS Lambda 与其他服务结合使用 \(p. 138\)](#)。

AWS Lambda 权限

可以使用 AWS Identity and Access Management (IAM) 来管理对 Lambda API 和资源（如函数和层）的访问。对于您账户中使用 Lambda 的用户和应用程序，您在一个权限策略中管理权限，并可将该策略应用于 IAM 用户、组或角色。要向其他使用您的 Lambda 资源的账户或 AWS 服务授予权限，请使用一个适用于该资源本身的策略。

Lambda 函数也具有一个策略，称为[执行角色 \(p. 30\)](#)，用来向其授予访问 AWS 服务和资源的权限。您的函数至少需要访问 Amazon CloudWatch Logs 来实现日志流。如果您[使用 AWS X-Ray 跟踪您的函数 \(p. 371\)](#)，或者您的函数使用 AWS 开发工具包访问服务，则授予其在执行角色中调用它们的权限。当您使用[事件源映射 \(p. 90\)](#)来触发函数时，Lambda 还使用执行角色来获取从事件源读取的权限。

Note

如果您的函数需要通过网络来访问通过 AWS API 或 Internet 无法访问的资源（如关系数据库），则[配置您的函数以连接到 VPC \(p. 72\)](#)。

使用[基于资源的策略 \(p. 33\)](#)来授予其他账户和 AWS 服务使用您的 Lambda 资源的权限。Lambda 资源包括函数、版本、别名和层版本等。除了适用于用户的各种策略，每个这样的资源也都有一个在资源被访问时适用的权限策略。当 Amazon S3 等 AWS 服务调用您的 Lambda 函数时，基于资源的策略将授予它访问权限。

要管理您账户中的用户和应用程序的权限，请[使用 Lambda 提供的托管策略 \(p. 37\)](#)，或者编写您自己的策略。Lambda 控制台使用多种服务来获取有关您的函数的配置和触发器的信息。您可以原封不动地使用托管策略，或将其作为更严格策略的起点。

可以按操作所影响的资源，以及在某些情况下借助额外条件来限制用户权限。例如，可以为函数的 Amazon 资源名称 (ARN) 指定一个模式，要求用户在其创建的函数的名称中包括用户名。此外，还可以增加一个条件，要求用户配置函数以使用特定层（例如）来拉入日志记录软件。有关每个操作支持的资源和条件，请参阅[资源和条件 \(p. 41\)](#)。

有关 IAM 的更多信息，请参阅 IAM 用户指南 中的[什么是 IAM？](#)。

主题

- [AWS Lambda 执行角色 \(p. 30\)](#)
- [对 AWS Lambda 使用基于资源的策略 \(p. 33\)](#)
- [用于 AWS Lambda 的基于身份的 IAM 策略 \(p. 37\)](#)
- [Lambda 操作的资源和条件 \(p. 41\)](#)
- [使用 AWS Lambda 应用程序的权限边界 \(p. 45\)](#)

AWS Lambda 执行角色

AWS Lambda 函数的执行角色授予该函数访问 AWS 服务和资源的权限。您在创建函数时提供该角色，当您的函数被调用时，Lambda 代入该角色。您可以创建一个有权将日志发送到 Amazon CloudWatch 并将跟踪数据上传到 AWS X-Ray 的开发执行角色。

查看函数的执行角色

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 选择 Permissions。

4. 资源摘要显示了函数有权访问的服务和资源。以下示例显示了当您在 Lambda 控制台中创建一个执行角色时，Lambda 向该执行角色添加的 CloudWatch Logs 权限。

Resource summary



Amazon CloudWatch Logs
3 actions, 2 resources

To view the resources and actions that your function has permission to access, choose a service.

By action

By resource

Resource

Actions

arn:aws:logs:us-east-2:123456789012:*

Allow: log

arn:aws:logs:us-east-2:123456789012:log-group:/aws/lambda/my-function:*

Allow: log

Allow: log

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. 从下拉菜单中选择一个服务可查看与该服务相关的权限。

可以随时在函数的执行角色中添加或删除权限，或配置您的函数以使用不同的角色。为您的函数使用 AWS 开发工具包调用的任何服务以及为 Lambda 用来启用可选功能的服务添加权限。

当您向函数添加权限时，也要更新其代码或配置。这将强制停止并替换正在运行的带过期凭证的函数实例。

在 IAM 控制台中创建执行角色

默认情况下，当您在 Lambda 控制台中[创建函数 \(p. 3\)](#)时，Lambda 会创建具有最少权限的执行角色。您也可以在 IAM 控制台中创建执行角色。

在 IAM 控制台中创建执行角色

- 打开 IAM 控制台中的“[角色](#)”页面。
- 选择 Create role (创建角色)。
- 在 Common use cases (常见使用案例) 下，选择 Lambda。
- 选择 Next: Permissions (下一步：权限)。

5. 在 Attach permissions policies (附加权限策略) 下，选择 AWSLambdaBasicExecutionRole 和 AWSXRayDaemonWriteAccess 托管策略。
6. 选择下一步：标签。
7. 选择下一步：审核。
8. 对于 Role Name (角色名称)，输入 **lambda-role**。
9. 选择创建角色。

有关详细说明，请参阅 IAM 用户指南 中的[创建角色](#)。

使用 IAM API 管理角色

执行角色是在您调用函数时 Lambda 有权代入的 IAM 角色。要使用 AWS CLI 创建执行角色，请使用 `create-role` 命令。

```
$ aws iam create-role --role-name lambda-ex --assume-role-policy-document file://trust-policy.json
{
    "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"
                }
            ]
        }
    }
}
```

`trust-policy.json` 文件是当前目录中的 JSON 文件，该文件定义了角色的[信任策略](#)。此信任策略通过向服务委托人授予调用 AWS Security Token Service `AssumeRole` 操作所需的 `lambda.amazonaws.com` 权限来允许 Lambda 使用角色的权限。

Example `trust-policy.json`

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

您也可以内联方式指定信任策略。JSON 字符串中转义引号的要求因您的 Shell 而异。

```
$ 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"}}]'
```

要向角色添加权限，请使用 `attach-role-policy-to-role` 命令。首先，添加 `AWSLambdaBasicExecutionRole` 托管策略。

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

Lambda 功能的托管策略

以下托管策略提供使用 Lambda 功能所需的权限：

- `AWSLambdaBasicExecutionRole` – 将日志上传到 CloudWatch 的权限。
- `AWSLambdaKinesisExecutionRole` – 读取来自 Amazon Kinesis 数据流或使用者的事件的权限。
- `AWSLambdaDynamoDBExecutionRole` – 读取 Amazon DynamoDB 流记录的权限。
- `AWSLambdaSQSQueueExecutionRole` – 读取 Amazon Simple Queue Service (Amazon SQS) 队列消息的权限。
- `AWSLambdaVPCAccessExecutionRole` – 管理弹性网络接口以将您的函数连接到 VPC 的权限。
- `AWSXRayDaemonWriteAccess` – 将跟踪数据上传到 X-Ray 的权限。

对于某些功能，Lambda 控制台会尝试在客户托管策略中向执行角色添加缺失的权限。这些策略可能会变得很多。在启用功能之前，请将相关的托管策略添加到您的执行角色以避免创建额外策略。

当您使用[事件源映射 \(p. 90\)](#)调用您的函数时，Lambda 将使用执行角色读取事件数据。例如，Amazon Kinesis 的事件源映射从数据流读取事件并将事件成批发送到您的函数。可以将事件源映射用于以下服务：

Lambda 从其读取事件的服务

- [Amazon Kinesis \(p. 212\)](#)
- [Amazon DynamoDB \(p. 184\)](#)
- [Amazon Simple Queue Service \(p. 255\)](#)

除了托管策略，Lambda 控制台还为创建包含与额外用例相关的权限的自定义策略提供模板。当您在 Lambda 控制台中创建函数时，可以选择利用来自一个或多个模板的权限创建新的执行角色。当您从蓝图创建函数，或者配置需要访问其他服务的选项时，也会自动应用这些模板。示例模板可从本指南的[GitHub 存储库](#)中找到。

对 AWS Lambda 使用基于资源的策略

AWS Lambda 支持将基于资源的权限策略用于 Lambda 函数和层。基于资源的策略允许您基于资源向其他账户授予使用权限。您也可以使用基于资源的策略来允许 AWS 服务调用您的函数。

对于 Lambda 函数，您可以[授予账户权限 \(p. 35\)](#)以便其可以调用或管理这些函数。可以添加多个语句来向多个账户授权，或允许任何账户调用您的函数。对于其他 AWS 服务为响应您账户中的活动而调用的函数，您可以使用策略来[向服务授予调用权限 \(p. 35\)](#)。

查看函数的基于资源的策略

1. 打开 Lambda 控制台 [函数页面](#)。

2. 选择函数。
3. 选择 Permissions。
4. 基于资源的策略显示了在其他账户或 AWS 服务尝试访问该函数时应用的权限。以下示例显示了一个语句，该语句允许 Amazon S3 调用为账户 123456789012 中名为 my-bucket 的存储桶调用名为 my-function 的函数。

Resource-based policy [Info](#)

```
1  {
2      "Version": "2012-10-17",
3      "Id": "default",
4      "Statement": [
5          {
6              "Sid": "lambda-d49ff629-xmpl-454d-8473-04c11fdc424c",
7              "Effect": "Allow",
8              "Principal": {
9                  "Service": "s3.amazonaws.com"
10             },
11             "Action": "lambda:InvokeFunction",
12             "Resource": "arn:aws:lambda:us-east-2:123456789012:function:my-function"
13             "Condition": {
14                 "StringEquals": {
15                     "AWS:SourceAccount": "123456789012"
16                 },
17                 "ArnLike": {
18                     "AWS:SourceArn": "arn:aws:s3:::my-bucket"
19                 }
20             }
21         }
22     ]
23 }
```

对于 Lambda 层，在层版本上使用基于资源的策略以允许其他账户使用它。除了向单个或所有账户授权的策略，对于层，您还可以向组织中的所有账户授权。

Note

您只能为 [AddPermission \(p. 407\)](#) 和 [AddLayerVersionPermission \(p. 404\)](#) API 操作范围内的 Lambda 资源更新基于资源的策略。不能为您在 JSON 中的 Lambda 资源编写策略，或使用不映射到那些操作的参数的条件。

基于资源的策略应用于单个函数、版本、别名或层版本。它们向一个或多个服务和账户授权。对于您希望其能够访问多个资源或使用基于资源的策略不支持的 API 操作的信任账户，您可以使用[跨账户角色 \(p. 37\)](#)。

主题

- 向 AWS 服务授予函数访问权 ([p. 35](#))
- 向其他账户授予函数访问权 ([p. 35](#))
- 向其他账户授予层访问权 ([p. 36](#))
- 清除基于资源的策略 ([p. 37](#))

向 AWS 服务授予函数访问权

当您使用 AWS 服务调用您的函数 (p. 138) 时，可以用基于资源的策略语句授权。可以将该语句应用于函数，或将其限制为单个版本或别名。

Note

当您通过 Lambda 控制台向函数添加触发器时，该控制台会更新函数的基于资源的策略以允许服务调用它。要向 Lambda 控制台中不可用的其他账户或服务授予权限，请使用 AWS CLI。

使用 `add-permission` 命令添加一条语句。最简单的基于资源的策略语句是允许一个服务调用某个函数。以下命令授予 Amazon SNS 调用名为 `my-function` 的函数的权限。

```
$ aws lambda add-permission --function-name my-function --action lambda:InvokeFunction --statement-id sns \
--principal sns.amazonaws.com --output text
{"Sid":"sns","Effect":"Allow","Principal": \
{"Service":"sns.amazonaws.com"},"Action":"lambda:InvokeFunction","Resource":"arn:aws:lambda:us-east-2:123456789012:function:my-function"}
```

这允许 Amazon SNS 调用该函数，但不限制触发调用的 Amazon SNS 主题。为确保您的函数只被特定资源调用，请使用 `source-arn` 选项指定资源的 Amazon 资源名称 (ARN)。以下命令只允许 Amazon SNS 调用名为 `my-topic` 的主题的订阅函数。

```
$ 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
```

有些服务可以调用其他账户中的函数。如果您指定的一个源 ARN 中包含您的账户 ID，这不是问题。但对于 Amazon S3 来说，源是其 ARN 中不包含账户 ID 的存储桶。有可能是您删除了该存储桶，而另一个账户用同样的名称创建了这样一个存储桶。使用 `account-id` 选项确保只有您账户中的资源可以调用该函数。

```
$ 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-123456 --source-account 123456789012
```

向其他账户授予函数访问权

要向另一个 AWS 账户授权，请将账户 ID 指定为 `principal`。以下示例向账户 210987654321 授权以 `prod` 别名调用 `my-function`。

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

基于资源的策略向另一个账户授予访问此函数的权限，但不允许该账户中的用户超出其权限。另一个账户中的用户必须具有相应的用户权限 (p. 37) 才能使用 Lambda API。

要限制对另一账户中用户、组或角色的访问，请将身份的完整 ARN 指定为委托人。例如：`arn:aws:iam::123456789012:user/developer`。

别名 (p. 65) 限制了其他账户可以调用哪个版本。它要求其他账户在函数 ARN 中包括该别名。

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

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

然后，您可以根据需要更新该别名以指向新版本。当您更新别名时，其他账户不需要更改其代码以使用新版本，它仅有权调用您选择的版本。

您可以授予对[作用于现有函数 \(p. 43\)](#)的大多数 API 操作的跨账户访问权。例如，您可以授予 `lambda>ListAliases` 权限，以允许一个账户获得别名列表，或授予 `lambda:GetFunction` 权限，以让它们下载您的函数代码。分别添加每个权限，或使用 `lambda:*` 授予有关指定函数的所有操作的权限。

跨账户 API

- [Invoke \(p. 482\)](#)
- [GetFunction \(p. 455\)](#)
- [GetFunctionConfiguration \(p. 460\)](#)
- [UpdateFunctionCode \(p. 553\)](#)
- [DeleteFunction \(p. 436\)](#)
- [PublishVersion \(p. 518\)](#)
- [ListVersionsByFunction \(p. 511\)](#)
- [CreateAlias \(p. 411\)](#)
- [GetAlias \(p. 448\)](#)
- [ListAliases \(p. 489\)](#)
- [UpdateAlias \(p. 543\)](#)
- [DeleteAlias \(p. 430\)](#)
- [GetPolicy \(p. 477\)](#)
- [PutFunctionConcurrency \(p. 525\)](#)
- [DeleteFunctionConcurrency \(p. 438\)](#)
- [ListTags \(p. 509\)](#)
- [TagResource \(p. 539\)](#)
- [UntagResource \(p. 541\)](#)

要授予其他账户对多个函数的权限，或不对某个函数执行的操作的权限，请使用[角色 \(p. 37\)](#)。

向其他账户授予层访问权

要向另一个账户授予层使用权限，请使用 `add-layer-version-permission` 命令向层版本的权限策略添加语句。在每个语句中，您可以向单个账户、所有账户或组织授予权限。

```
$ aws lambda add-layer-version-permission --layer-name xray-sdk-nodejs --statement-id  
xaccount \  
--action lambda:GetLayerVersion --principal 210987654321 --version-number 1 --output text  
e210fffdc-e901-43b0-824b-5fc0d0dd26d16 {"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"}
```

权限仅适用于层的单一版本。每次创建新的层版本时都重复此过程。

要向一个组织中的所有账户授予权限，请使用 `organization-id` 选项。以下示例向组织中的所有账户授予层的版本 3 的使用权限。

```
$ aws lambda add-layer-version-permission --layer-name my-layer \  
--principal arn:aws:organizations::123456789012:* --version-number 3 --output text
```

```
--statement-id engineering-org --version-number 3 --principal '*' \
--action lambda:GetLayerVersion --organization-id o-t194hfs8cz --output text
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"}}}"
```

要向所有 AWS 账户授予权限，请将 * 用于委托人并忽略组织 ID。对于多个账户或组织，请添加多个语句。

清除基于资源的策略

要查看函数的基于资源的策略，请使用 `get-policy` 命令。

```
$ aws lambda get-policy --function-name my-function --output text
{"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*"}}}]}      7c681fc9-b791-4e91-
acdf-eb847fd8aa0f0
```

对于版本和别名，请在函数名后面附加版本号或别名。

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

要从函数中删除权限，请使用 `remove-permission`。

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

使用 `get-layer-version-policy` 命令可查看图层上的权限。使用 `remove-layer-version-
permission` 可从策略中删除语句。

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



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

用于 AWS Lambda 的基于身份的 IAM 策略

您可以使用 AWS Identity and Access Management (IAM) 中基于身份的策略授予您账户中的用户访问 Lambda 的权限。基于身份的策略可以直接应用于用户，也可以应用于与用户相关的组和角色。您也可以授予另一个账户中的用户在您的账户中代入角色和访问您的 Lambda 资源的权限。

Lambda 提供托管策略，可授予访问 Lambda API 操作的权限，有时为访问其他用于开发和管理 Lambda 资源的服务的权限。Lambda 根据需要更新托管策略，以确保您的用户有权在新功能发布时访问它们。

- `AWSLambdaFullAccess` – 授予对 AWS Lambda 操作和其他用于开发和维护 Lambda 资源的服务的完全访问权。
- `AWSLambdaReadOnlyAccess` – 授予对 AWS Lambda 资源的只读访问权限。
- `AWSLambdaRole` – 授予调用 Lambda 函数的权限。

托管策略授予 API 操作的权限，而不限制用户可以修改的函数或层。要进行更精细的控制，您可以创建自己的策略来限制用户权限的范围。

小节目录

- [函数开发 \(p. 38\)](#)
- [层开发和使用 \(p. 40\)](#)
- [跨账户角色 \(p. 41\)](#)

函数开发

以下显示具有有限范围的权限策略示例。该策略允许用户创建和管理名称前带指定前缀 (`intern-`) 并用指定执行角色配置的 Lambda 函数。

Example 函数开发策略

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ReadOnlyPermissions",  
            "Effect": "Allow",  
            "Action": [  
                "lambda:GetAccountSettings",  
                "lambda>ListFunctions",  
                "lambda>ListTags",  
                "lambda:GetEventSourceMapping",  
                "lambda>ListEventSourceMappings",  
                "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:PassRole"
    ],
    "Resource": "arn:aws:iam::*:role/intern-lambda-execution-role"
},
{
    "Sid": "ViewExecutionRolePolicies",
    "Effect": "Allow",
    "Action": [
        "iam:GetPolicy",
        "iam:GetPolicyVersion"
    ],
    "Resource": "arn:aws:iam::aws:policy/*"
},
{
    "Sid": "ViewLogs",
    "Effect": "Allow",
    "Action": [
        "logs:)"
    ],
    "Resource": "arn:aws:logs:***:log-group:/aws/lambda/intern-*"
}
]
}

```

策略中的权限基于它们所支持的[资源和条件 \(p. 41\)](#)组织成语句。

- **ReadOnlyPermissions** – 当您浏览和查看函数时，Lambda 控制台使用这些权限。它们不支持资源模式或条件。

```

    "Action": [
        "lambda:GetAccountSettings",
        "lambda>ListFunctions",
        "lambda>ListTags",
        "lambda:GetEventSourceMapping",
        "lambda>ListEventSourceMappings",
        "iam>ListRoles"
    ],
    "Resource": "*"

```

- **DevelopFunctions** – 使用任何对前缀为 intern- 的函数执行的 Lambda 操作，但 AddPermission 和 PutFunctionConcurrency 除外。AddPermission 修改函数上[基于资源的策略 \(p. 33\)](#)并可能影响安全性。PutFunctionConcurrency 保留函数的扩展容量并且可以从其他函数那里取得容量。

```

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

```

- **DevelopEventSourceMappings** – 管理前缀为 intern- 的函数的事件源映射。虽然这些操作在事件源映射上运行，但您可以通过附带条件的函数限制它们。

```

    "Action": [
        "lambda>DeleteEventSourceMapping",
        "lambda>UpdateEventSourceMapping",
        "lambda>CreateEventSourceMapping"
    ],
    "Resource": "*",

```

```
"Condition": {
    "StringLike": {
        "lambda:FunctionArn": "arn:aws:lambda:*::function:intern-*"
    }
}
```

- **PassExecutionRole** – 查看并仅传递名为 intern-lambda-execution-role 的角色，该角色必须由具有 IAM 权限的用户创建和管理。当您为函数分配执行角色时，使用 PassRole。

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

- **ViewExecutionRolePolicies** – 查看附加到执行角色的 AWS 提供的托管策略。这使得您可以在控制台中查看该函数的权限，但不包含查看由账户中其他用户创建的策略的权限。

```
"Action": [
    "iam>GetPolicy",
    "iam>GetPolicyVersion"
],
"Resource": "arn:aws:iam::aws:policy/*"
```

- **ViewLogs** – 使用 CloudWatch Logs 查看前缀为 intern- 的函数的日志。

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

该策略允许用户开始使用 Lambda，而不会将其他用户的资源置于风险之下。它不允许用户配置能被触发或调用其他 AWS 服务的函数，这需要更广的 IAM 权限。它也不包含对于不支持有限范围策略的服务（如 CloudWatch 和 X-Ray）的权限。将只读策略用于这些服务以便让用户能够访问指标和跟踪数据。

当您为您的函数配置触发器时，需要有权使用调用您的函数的 AWS 服务。例如，要配置 Amazon S3 触发器，您需要拥有 Amazon S3 操作权限以便管理存储桶通知。其中很多权限包括在 AWSLambdaFullAccess 托管策略中。示例策略可从本指南的 [GitHub 存储库](#) 中找到。

层开发和使用

以下策略授予用户创建层并通过函数使用层的权限。资源模式允许用户在任何 AWS 区域工作并使用任何层版本，只要层的名称以 test- 开头即可。

Example 层开发策略

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

您还可以附加 `lambda:Layer` 条件以在函数创建和配置过程中强制使用层。例如，您可以防止用户使用其他账户发布的层。以下策略在 `CreateFunction` 和 `UpdateFunctionConfiguration` 操作中添加一个条件，以要求任何指定层来自账户 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:/*"
                    ]
                }
            }
        }
    ]
}
```

为确保条件应用，应验证没有其他语句向用户授予这些操作的权限。

跨账户角色

您可以将任何上述策略和语句应用于一个角色，然后与另一个账户共享该角色，使其可以访问您的 Lambda 资源。与 IAM 用户不同，角色没有用于身份验证的凭据。相反，它具有信任策略，可以指定谁能够代入该角色并使用其权限。

可以使用跨账户角色来允许您信任的账户访问 Lambda 操作和资源。如果您只是想授予调用函数或使用层的权限，请改为使用 [基于资源的策略 \(p. 33\)](#)。

有关更多信息，请参阅 IAM 用户指南 中的 [IAM 角色](#)。

Lambda 操作的资源和条件

您可以通过在 IAM 策略中指定资源和条件来限制用户权限的范围。每个 API 操作都支持资源和条件类型的组合，这些类型根据操作的行为而有所不同。

每条 IAM 策略语句为对一个资源执行的一个操作授予权限。如果操作不对指定资源执行操作，或者您授予对所有资源执行操作的权限，则策略中资源的值为通配符 (*)。对于许多 API 操作，可以通过指定资源的 Amazon 资源名称 (ARN) 或与多个资源匹配的 ARN 模式来限制用户可修改的资源。

要按资源限制权限，请指定资源的 ARN。

Lambda 资源 ARN 格式

- 函数 – arn:aws:lambda:*us-west-2:123456789012:function:my-function*
- 函数版本 – arn:aws:lambda:*us-west-2:123456789012:function:my-function:1*
- 函数别名 – arn:aws:lambda:*us-west-2:123456789012:function:my-function:TEST*
- 事件源映射 – arn:aws:lambda:*us-west-2:123456789012:event-source-mapping:fa123456-14a1-4fd2-9fec-83de64ad683de6d47*
- 层 – arn:aws:lambda:*us-west-2:123456789012:layer:my-layer*
- 层版本 – arn:aws:lambda:*us-west-2:123456789012:layer:my-layer:1*

例如，以下策略允许 123456789012 账户中的用户调用 美国西部（俄勒冈）区域中名为 my-function 的函数。

Example 调用函数策略

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

这是一种特殊情形，其中，操作标识符 (lambda:InvokeFunction) 不同于 API 操作 ([Invoke \(p. 482\)](#))。对于其他操作，操作标识符为操作名称加上 lambda: 前缀。

条件是可选的策略元素，它应用其他逻辑来确定是否允许执行操作。除了所有操作支持的[公用条件](#)之外，Lambda 定义了一些条件类型，您可以用来限制某些操作的额外参数的值。

例如，lambda:Principal 条件允许您限制用户可以在函数的基于资源的策略上为其授予调用权限的服务或账户。以下策略允许用户授予对 SNS 主题的权限，以调用名为 test 的函数。

Example 管理函数策略权限

```
{  
    "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": {  
                    "AWS:Principal": "user@example.com"  
                }  
            }  
        }  
    ]  
}
```

```

        "lambda:Principal": "sns.amazonaws.com"
    }
}
]
```

该条件要求委托人是 Amazon SNS 而不是其他服务或账户。资源模式要求函数名称为 `test` 并包含版本号或别名。例如：`test:v1`。

有关 Lambda 和其他 AWS 服务的资源和条件的更多信息，请参阅 IAM 用户指南 中的操作、资源和条件键。

小节目录

- [函数 \(p. 43\)](#)
- [事件源映射 \(p. 44\)](#)
- [层 \(p. 44\)](#)

函数

对函数进行的操作可以通过函数、版本或别名 ARN 限制为特定函数，如下表中所述。只能为所有资源 (*) 授予不支持资源限制的操作。

函数

操作	资源	Condition
AddPermission (p. 407)	Function	<code>lambda:Principal</code>
RemovePermission (p. 537)	函数版本	
	函数别名	
Invoke (p. 482)	函数	无
权限： <code>lambda:InvokeFunction</code>	函数版本	
	函数别名	
CreateFunction (p. 421)	函数	<code>lambda:Layer</code>
UpdateFunctionConfiguration (p. 560)		
CreateAlias (p. 411)	Function	无
DeleteAlias (p. 430)		
DeleteFunction (p. 436)		
DeleteFunctionConcurrency (p. 438)		
GetAlias (p. 448)		
GetFunction (p. 455)		
GetFunctionConfiguration (p. 460)		
GetPolicy (p. 477)		
ListAliases (p. 489)		

操作	资源	Condition
ListVersionsByFunction (p. 511) PublishVersion (p. 518) PutFunctionConcurrency (p. 525) UpdateAlias (p. 543) UpdateFunctionCode (p. 553)		
GetAccountSettings (p. 446) ListFunctions (p. 498) ListTags (p. 509) TagResource (p. 539) UntagResource (p. 541)	*	无

事件源映射

对于事件源映射，可以将删除和更新权限限制为特定事件源。`lambda:FunctionArn` 条件允许您限制用户可配置事件源以调用的函数。

对于这些操作，资源是事件源映射，因此 Lambda 提供了一个条件，允许您根据事件源映射调用的函数来限制权限。

事件源映射

操作	资源	Condition
DeleteEventSourceMapping (p. 432)	事件源映射	<code>lambda:FunctionArn</code>
UpdateEventSourceMapping (p. 547)		
CreateEventSourceMapping (p. 415)	*	<code>lambda:FunctionArn</code>
GetEventSourceMapping (p. 451)	*	无
ListEventSourceMappings (p. 492)		

层

通过层操作，您可以限制用户可通过函数管理或使用的层。与层使用和权限相关的操作作用于层的版本，而 `PublishLayerVersion` 作用于层名称。两者都可以与通配符一起使用，以通过名称限制用户可以使用的层。

层

操作	资源	Condition
AddLayerVersionPermission (p. 404)	层版本	无
RemoveLayerVersionPermission (p. 535)		

操作	资源	Condition
GetLayerVersion (p. 469)		
GetLayerVersionPolicy (p. 475)		
DeleteLayerVersion (p. 442)		
PublishLayerVersion (p. 514)	层	无
ListLayers (p. 501)	*	无
ListLayerVersions (p. 503)		

使用 AWS Lambda 应用程序的权限边界

当您在 AWS Lambda 控制台中[创建应用程序 \(p. 122\)](#)时，Lambda 会将权限边界 应用到应用程序的 IAM 角色。权限边界限定了应用程序模板为其每个函数创建的[执行角色 \(p. 30\)](#) 范围，以及您添加到模板的任何角色。权限边界可以防止对应用程序 Git 存储库具有写入访问权限的用户将应用程序的权限提升到自身资源的范围之外。

Lambda 控制台中的应用程序模板包含一个全局属性，该属性将权限边界应用于通过这些模板创建的所有函数。

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

边界限定了函数角色的权限。您可以在模板中向函数的执行角色添加权限，但该权限只有在权限边界允许的范围内有效。对于 AWS CloudFormation 在部署应用程序时加入的角色，强制使用权限边界。该角色只有创建和传递已附加应用程序权限边界的角色的权限。

默认情况下，应用程序权限边界允许函数对应用程序中的资源执行操作。例如，如果应用程序包含 Amazon DynamoDB 表，则对于可限制为在具有资源级权限的特定表上操作的任何 API 操作，边界允许对这些 API 操作进行访问。只有在边界中专门允许的情况下，才能使用不支持资源级权限的操作。这些包括用于日志记录和跟踪的 Amazon CloudWatch Logs 和 AWS X-Ray API 操作。

Example 权限边界

```
{
  "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:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:us-east-1:123456789012:log-group:/aws/lambda/my-app:*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

```
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:PutLogEvents",
        "xray:Put*"
    ],
    "Resource": "*",
    "Effect": "Allow",
    "Sid": "StaticPermissions"
},
...
]
```

要访问其他资源或 API 操作，您或管理员必须扩展权限边界，在其中包括这些资源。您可能还需要更新应用程序的执行角色或部署角色，允许使用其他操作。

- 权限边界 – 在您向应用程序添加资源时，或执行角色需要访问更多操作时，扩展应用程序的权限边界。在 IAM 中，向边界添加资源，以便允许使用支持该资源类型的资源级权限的 API 操作。对于不支持资源级权限的操作，将这些操作添加到不限定为任何资源范围的语句中。
- 执行角色 – 在需要使用其他操作时，扩展函数的执行角色。在应用程序模板中，将策略添加到执行角色。向函数授予边界与执行角色的权限交集。
- 部署角色 – 在应用程序需要其他权限来创建或配置资源时，扩展应用程序的部署角色。在 IAM 中，将策略添加到应用程序的部署角色。部署角色需要与您在 AWS CloudFormation 中部署或更新应用程序时所需的相同用户权限。

有关演练如何向应用程序添加资源并扩展其权限的教程，请参阅[??? \(p. 122\)](#)。

有关更多信息，请参阅《IAM 用户指南》中的 [IAM 实体的权限边界](#)。

管理 AWS Lambda 函数

您可以使用 AWS Lambda API 或控制台配置 Lambda 函数设置。基本函数设置 (p. 47) 包括您在 Lambda 控制台中创建函数时指定的描述、角色和运行时。您可以在创建函数后配置更多设置，或者在创建过程中使用 API 设置处理程序名称、内存分配和安全组等内容。

要将密钥保留在函数代码之外，请将密钥存储在函数配置中，并在初始化期间从执行环境读取它们。**环境变量 (p. 49)** 始终进行静态加密，也可以在客户端加密。可以使用环境变量通过删除外部资源的连接字符串、密码和终端节点来使您的函数代码变得可移植。

版本和别名 (p. 63) 是辅助资源，可创建这类资源以管理函数部署和调用。发布函数的**版本 (p. 63)** 以将其代码和配置存储为无法更改的单独资源，并创建指向特定版本的**别名 (p. 65)**。然后，您可以配置客户端以调用函数别名，并在要将客户端指向新版本时更新别名，而不是更新客户端。

在将库和其他依赖项添加到函数时，创建并上传部署程序包可能会降低开发速度。使用**层 (p. 68)** 可以单独管理您的函数依赖项，并使部署包保持较小。您也可以使用层来与其他客户共享您自己的库，并将公开可用的层与您的函数结合使用。

要在 Amazon VPC 中将 Lambda 函数与 AWS 资源结合使用，请使用安全组和子网配置它以**创建 VPC 连接 (p. 72)**。通过将函数连接到 VPC，您可以访问私有子网中的资源，例如关系数据库和缓存。您还可为 MySQL 和 Aurora 数据库实例**创建数据库代理 (p. 75)**。数据库代理使函数能够在不耗尽数据库连接的情况下达到高并发级别。

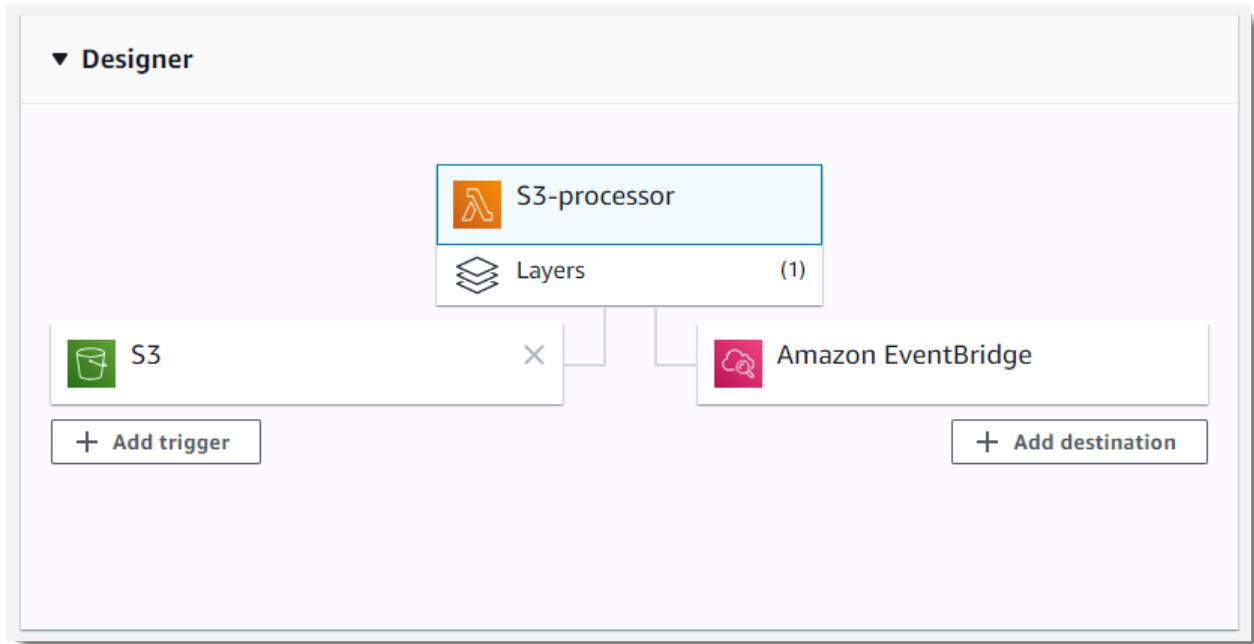
主题

- 在 AWS Lambda 控制台中配置函数 (p. 47)
- 使用 AWS Lambda 环境变量 (p. 49)
- 管理 Lambda 函数的并发 (p. 54)
- AWS Lambda 函数版本 (p. 63)
- AWS Lambda 函数别名 (p. 65)
- AWS Lambda 层 (p. 68)
- 配置 Lambda 函数以访问 VPC 中的资源 (p. 72)
- 配置 Lambda 函数的数据库访问 (p. 75)
- 标记 Lambda 函数 (p. 77)

在 AWS Lambda 控制台中配置函数

您可以使用 Lambda 控制台配置函数设置、添加触发器和目标以及更新和测试代码。

要管理函数，请打开 Lambda 控制台的**函数**页面并选择函数。函数设计器位于配置页面的顶部。



设计器显示您的函数及其上游和下游资源的概述。您可以使用它来配置触发器、层和目标。

- 触发器 – 触发器是您配置用来调用函数的服务和资源。选择 Add trigger (添加触发器) 以创建 Lambda 事件源映射 (p. 90) 或在 Lambda 控制台集成的其他服务中配置触发器。有关这些服务和其他服务的详细信息，请参阅将 AWS Lambda 与其他服务结合使用 (p. 138)。
- 层 – 选择 Layers (层) 节点以向应用程序添加层 (p. 68)。层是包含库、自定义运行时或其他依赖项的 ZIP 存档。
- 目标 – 向您的函数添加目标以将有关调用结果的详细信息发送给另一个服务。您可以在异步 (p. 83) 调用函数时发送调用记录，或者通过从流读取的事件源映射 (p. 90) 调用时发送。

在设计器中选择函数节点后，您可以修改以下设置。

函数设置

- 代码 – 函数的代码和依赖项。对于脚本语言，您可以在嵌入式编辑器 (p. 5) 中编辑函数代码。要添加库，或对于编辑器不支持的语言，请上传部署包 (p. 19)。如果您的部署包大于 50 MB，请选择 Upload a file from Amazon S3 (从 S3 上传文件)。
- 运行时 – 执行您的函数的 Lambda 运行时 (p. 108)。
- 处理程序 – 在调用您的函数时运行时执行的方法，如 `index.handler`。第一个值是文件或模块的名称。第二个值是方法的名称。
- 环境变量 – Lambda 在执行环境中设置的键值对。使用环境变量 (p. 49) 在代码之外扩展函数的配置。
- 标签 – Lambda 附加到您的函数资源的键值对。使用标签 (p. 77) 将 Lambda 函数组织到组中，以便在 Lambda 控制台中进行成本报告和筛选。

标签应用到整个函数，包括所有版本和别名。

- 执行角色 – AWS Lambda 执行函数时代入的 IAM 角色 (p. 30)。
- 描述 – 该函数的描述。
- 内存 – 执行期间函数可用的内存量。请选择介于 128 MB 与 3,008 MB (p. 27) 之间的值，以 64 MB 为增量。

Lambda 以与配置的内存量成正比的方式线性分配 CPU 处理能力。在 1792 MB 时，函数拥有相当于一个完整 vCPU（每秒一个 vCPU 秒的积分）的处理能力。

- 超时 – Lambda 在停止函数前允许其运行的时间。默认值为 3 秒。允许的最大值为 900 秒。
- Virtual Private Cloud (VPC) – 如果您的函数需要通过网络访问无法在 Internet 上获得的资源，请[将其配置为连接到 VPC \(p. 72\)](#)。
- 数据库代理 – 为使用 Amazon RDS 数据库实例或集群的函数[创建数据库代理 \(p. 75\)](#)。
- 活动跟踪 – 对传入请求进行采样并[使用 AWS X-Ray 跟踪采样的请求 \(p. 371\)](#)。
- 并发 – [为函数预留并发 \(p. 54\)](#)，以设置函数的最大同时执行数。预配置并发性，以确保函数可以扩展而不会引起延迟的波动。

预留并发应用到整个函数，包括所有版本和别名。

- 异步调用 – [配置错误处理行为 \(p. 83\)](#)以减少 Lambda 尝试的重试次数，或者缩短未处理事件在 Lambda 丢弃事件之前保留在队列中的时间。[配置死信队列 \(p. 88\)](#)以保留已丢弃的事件。

您可以配置有关函数、版本或别名的错误处理设置。

除了前面列表中所列项之外，您只能更改函数未发布版本中的函数设置。发布版本后，代码和大多数设置将锁定，以确保为该版本的用户提供一致的体验。使用[别名 \(p. 65\)](#)以受控方式传播配置更改。

要使用 Lambda API 配置函数，请使用以下操作：

- [UpdateFunctionCode \(p. 553\)](#) – 更新函数代码
- [UpdateFunctionConfiguration \(p. 560\)](#) – 更新特定于版本的设置。
- [TagResource \(p. 539\)](#) – 标记函数。
- [AddPermission \(p. 407\)](#) – 修改函数、版本或别名的基于资源的策略 (p. 33)。
- [PutFunctionConcurrency \(p. 525\)](#) – 配置函数的预留并发。
- [PublishVersion \(p. 518\)](#) – 使用当前代码和配置创建不可变版本。
- [CreateAlias \(p. 411\)](#) – 为函数版本创建别名。
- [PutFunctionEventInvokeConfig](#) – 配置异步调用的错误处理。

例如，要使用 AWS CLI 更新函数的内存设置，请使用 `update-function-configuration` 命令。

```
$ aws lambda update-function-configuration --function-name my-function --memory-size 256
```

有关函数配置的最佳实践，请参阅[函数配置 \(p. 136\)](#)。

使用 AWS Lambda 环境变量

您可以使用环境变量来安全地存储密钥并调整函数的行为，而无需更新代码。环境变量是存储在函数的版本特定配置中的一对字符串。Lambda 运行时使环境变量可用于您的代码，并设置其他环境变量，这些变量包含有关函数和调用请求的信息。

通过指定键和值，您可以在函数的未发布版本上设置环境变量。发布一个版本时，会锁定该版本的环境变量以及其他[特定于版本的配置 \(p. 47\)](#)。

在 Lambda 控制台中设置环境变量

1. 打开 Lambda 控制台 [函数页面](#)。

2. 选择函数。
3. 在 Environment variables (环境变量) 下，选择 Edit (编辑)。
4. 选择 Add environment variable (添加环境变量)。
5. 输入密钥和值。

要求

- 密钥以字母开头，并且至少为两个字符。
 - 键仅包含字母、数字和下划线字符 (_)。
 - 键不是 [Lambda 预留 \(p. 51\)](#) 的值。
 - 所有环境变量的总大小不超过 4 KB。
6. 选择保存。

使用环境变量将环境特定的设置传递给您的代码。例如，您可以有两个具有相同代码但不同配置的函数。一个函数连接到测试数据库，另一个函数连接到生产数据库。在这种情况下，您可以使用环境变量告诉函数数据库的主机名和其他连接详细信息。您还可以设置环境变量来配置测试环境，以使用更详细的日志记录或更详细的跟踪。

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

要检索函数代码中的环境变量，请使用编程语言的标准方法。

Node.js

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

Python

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

Ruby

```
region = ENV["AWS_REGION"]
```

Java

```
String region = System.getenv("AWS_REGION");
```

Go

```
var region = os.Getenv("AWS_REGION")
```

C#

```
string region = Environment.GetEnvironmentVariable("AWS_REGION");
```

PowerShell

```
$region = $env:AWS_REGION
```

Lambda 通过静态加密来安全地存储环境变量。您可以[配置 Lambda 以使用不同的加密密钥 \(p. 52\)](#)、在客户端加密环境变量值或使用 AWS Secrets Manager 在 AWS CloudFormation 模板中设置环境变量。

小节目录

- [运行时环境变量 \(p. 51\)](#)
- [保护环境变量 \(p. 52\)](#)
- [使用 Lambda API 配置环境变量 \(p. 53\)](#)
- [示例代码和模板 \(p. 54\)](#)

运行时环境变量

Lambda [运行时 \(p. 108\)](#)会在初始化过程中设置多个环境变量。大多数环境变量提供有关函数或运行时的信息。这些环境变量的键是预留的，无法在函数配置中设置。

预留环境变量

- `_HANDLER` – 函数上配置的处理程序位置。
- `AWS_REGION` – 执行 Lambda 函数的 AWS 区域。
- `AWS_EXECUTION_ENV` – [运行时标识符 \(p. 108\)](#)，前缀为 `AWS_Lambda_` — 例如 `AWS_Lambda_java8`。
- `AWS_LAMBDA_FUNCTION_NAME` – 函数的名称。
- `AWS_LAMBDA_FUNCTION_MEMORY_SIZE` – 对函数可用的内存量（以 MB 为单位）。
- `AWS_LAMBDA_FUNCTION_VERSION` – 要执行的函数的版本。
- `AWS_LAMBDA_LOG_GROUP_NAME` , `AWS_LAMBDA_LOG_STREAM_NAME` – 函数的 Amazon CloudWatch Logs 组和流的名称。
- `AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY`、`AWS_SESSION_TOKEN` – 从函数的[执行角色 \(p. 30\)](#)中获取的访问密钥。
- `AWS_LAMBDA_RUNTIME_API` – ([自定义运行时 \(p. 111\)](#)) [运行时 API \(p. 112\)](#) 的主机和端口。
- `LAMBDA_TASK_ROOT` – 您的 Lambda 函数代码的路径。
- `LAMBDA_RUNTIME_DIR` – 运行时库的路径。
- `TZ` – 环境的时区 (UTC)。执行环境使用 NTP 同步系统时钟。

以下附加环境变量并非预留，可以在函数配置中扩展。

非预留环境变量

- `LANG` – 运行时的区域设置 (`en_US.UTF-8`)。
- `PATH` – 执行路径 (`/usr/local/bin:/usr/bin:/bin:/opt/bin`)。
- `LD_LIBRARY_PATH` – 系统库路径 (`/lib64:/usr/lib64:$LAMBDA_RUNTIME_DIR:$LAMBDA_RUNTIME_DIR/lib:$LAMBDA_TASK_ROOT:$LAMBDA_TASK_ROOT/lib:/opt/lib`)。

- **NODE_PATH** – ([Node.js \(p. 265\)](#)) Node.js 库路径 (/opt/nodejs/node12/node_modules/:/opt/nodejs/node_modules:\$LAMBDA_RUNTIME_DIR/node_modules)。
- **PYTHONPATH** – ([Python 2.7、3.6、3.8 \(p. 277\)](#)) Python 库路径 (\$LAMBDA_RUNTIME_DIR)。
- **GEM_PATH** – ([Ruby \(p. 291\)](#)) Ruby 库路径 (\$LAMBDA_TASK_ROOT/vendor/bundle/ruby/2.5.0:/opt/ruby/gems/2.5.0)。

显示的示例值反映了最新的运行时。特定变量或其值是否存在会因早先的运行时而异。

保护环境变量

Lambda 使用在您的账户中创建的密钥 (AWS 托管客户主密钥 (CMK)) 对环境变量进行加密。此密钥的使用是免费的。您也可以选择提供您自己的密钥，以供 Lambda 使用，而不是使用默认密钥。

当您提供密钥时，只有您账户中有权访问密钥的用户才能查看或管理函数上的环境变量。您的组织还可能有内部或外部要求，以管理用于加密的密钥并控制其轮换时间。

使用客户托管 CMK

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Environment variables (环境变量) 下，选择 Edit (编辑)。
4. 展开 Encryption configuration (加密配置)。
5. 选择 Use a customer master key (使用客户主密钥)。
6. 选择您的客户托管 CMK。
7. 选择保存。

客户托管的 CMK 产生标准 [AWS KMS 费用](#)。

您的用户或函数的执行角色不需要 AWS KMS 权限，即可使用默认加密密钥。要使用客户托管 CMK，您需要具有使用密钥的权限。Lambda 使用您的权限创建密钥授予。这允许 Lambda 使用它进行加密。

- `kms>ListAliases` – 在 Lambda 控制台中查看密钥。
- `kms>CreateGrant kms:Encrypt` – 在函数上配置客户托管 CMK。
- `kmsDecrypt` – 查看和管理使用客户托管的 CMK 加密的环境变量。

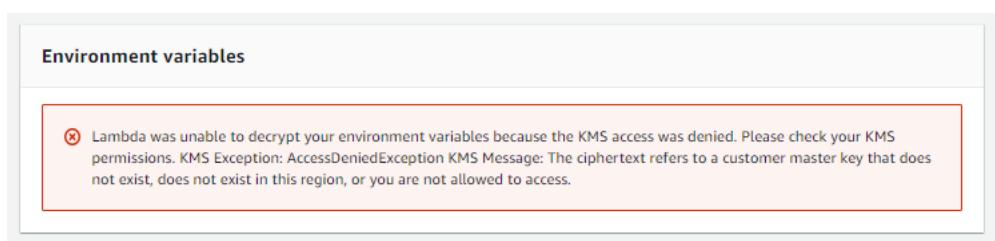
您可以从您的用户账户或从密钥的基于资源的权限策略获取这些权限。`ListAliases` 由 [Lambda 的托管策略 \(p. 37\)](#) 提供。密钥策略将剩余权限授予密钥用户组中的用户。

没有 `Decrypt` 权限的用户仍然可以管理函数，但无法在 Lambda 控制台中查看环境变量或管理它们。要防止用户查看环境变量，请向用户的权限添加一条语句，该语句拒绝访问默认密钥、客户托管密钥或所有密钥。

Example IAM 策略 – 按密钥 ARN 拒绝访问

```
{  
    "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"
}
}
```



有关管理密钥权限的详细信息，请参阅[在 AWS KMS 中使用密钥策略](#)。

您还可以在将环境变量值发送到 Lambda 之前在客户端对其进行加密，并在函数代码中对其解密。这会掩盖 Lambda 控制台和 API 输出中的密钥值，即使对于有权使用密钥的用户也是如此。在您的代码中，您可以从环境中检索加密的值并使用 AWS KMS API 对其进行解密。

在客户端加密环境变量

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Environment variables (环境变量) 下，选择 Edit (编辑)。
4. 展开 Encryption configuration (加密配置)。
5. 选择 Enable helpers for encryption in transit (为传输过程中的加密启用帮助程序)。
6. 选择变量旁边的 Encrypt (加密) 以加密其值。
7. 选择 Save。

要查看函数语言的示例代码，请选择环境变量旁边的 Code (代码)。示例代码显示如何检索函数中的环境变量并解密其值。

另一种选择是将密码存储在 AWS Secrets Manager 密钥中。您可以在 AWS CloudFormation 模板中引用密钥来设置数据库上的密码。您还可以在 Lambda 函数上设置环境变量的值。有关示例，请参阅下一节。

使用 Lambda API 配置环境变量

要使用 AWS CLI 或 AWS 开发工具包管理环境变量，请使用以下 API 操作。

- [UpdateFunctionConfiguration \(p. 560\)](#)
- [GetFunctionConfiguration \(p. 460\)](#)
- [CreateFunction \(p. 421\)](#)

以下示例在名为 my-function 的函数上设置两个环境变量。

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

使用 `update-function-configuration` 命令应用环境变量时，会替换 `Variables` 结构的整个内容。要在添加新环境变量时保留现有环境变量，请在请求中包含所有现有值。

要获取当前配置，请使用 `get-function-configuration` 命令。

```
$ aws lambda get-function-configuration --function-name my-function
{
    "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",
    ...
}
```

为了确保在读取配置和更新配置之间的间隔时间内值不会更改，您可以将 `get-function-configuration` 输出中的修订 ID 作为参数传递给 `update-function-configuration`。

要配置函数的加密密钥，请设置 `KMSKeyARN` 选项。

```
$ 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
```

示例代码和模板

本指南 GitHub 存储库中的示例应用程序演示了在函数代码和 AWS CloudFormation 模板中使用环境变量的情况。

示例应用程序

- **空白** – 在同一模板中创建函数和 Amazon SNS 主题。将主题名称传递给环境变量中的函数。读取代码中的环境变量（多种语言）。
- **RDS MySQL** – 在一个模板中使用存储在 Secrets Manager 中的密码创建 VPC 和 Amazon RDS 数据库实例。在应用程序模板中，从 VPC 堆栈导入数据库详细信息，从 Secrets Manager 读取密码，然后将所有连接配置传递给环境变量中的函数。

管理 Lambda 函数的并发

并发性是您的函数在任何给定时间所服务于的请求的数目。在调用函数时，Lambda 会分配它的一个实例以处理事件。当函数代码完成运行时，它会处理另一个请求。如果当仍在处理请求时再次调用函数，则分配另一个实例，从而增加该函数的并发。

并发受区域限制 (p. 27) 的约束，该限制由区域中的所有函数共享。为了确保函数始终可以达到一定的并发级别，您可以配置具有 [预留并发](#) (p. 55) 的函数。当一个函数有预留并发时，任何其他函数都不可以使用该并发。预留并发还限制了函数的最大并发数，并且适用于整个函数，包括版本和别名。

当 Lambda 分配函数的实例时，[运行时](#) (p. 108) 会加载函数的代码并运行您在处理程序之外定义的初始化代码。如果您的代码和依赖项很大，或者您在初始化过程中创建 SDK 客户端，则此过程可能需要一些时间。随着函数 [纵向扩展](#) (p. 94)，这会导致新实例提供的请求部分比其余部分具有更长的延迟。

要使函数能够在延迟不发生波动的情况下进行扩展，请使用[预配置并发 \(p. 57\)](#)。通过在调用增加之前分配预配置并发，您可以确保所有请求都由延迟非常低的初始化实例来提供。您可以在函数的版本或别名上配置预配置并发。

Lambda 也与 [Application Auto Scaling](#) 集成。您可以将 Application Auto Scaling 配置为根据计划或基于利用率管理预配置并发。使用计划扩展在预期流量高峰将至时增加预配置并发。要根据需要自动增加预配置的并发数，请使用[Application Auto Scaling API \(p. 60\)](#) 注册目标并创建扩展策略。

预配置并发计入函数的预留并发和区域限制。如果函数版本和别名上的预配置并发数加起来达到函数的预留并发，则所有调用都在预配置并发上运行。此配置还具有限制函数 (`$LATEST`) 未发布版本的效果，从而阻止其执行。

小目录

- [配置预留并发 \(p. 55\)](#)
- [配置预配置并发 \(p. 57\)](#)
- [使用 Lambda API 配置并发 \(p. 60\)](#)

配置预留并发

要管理函数的预留并发设置，请使用 Lambda 控制台。

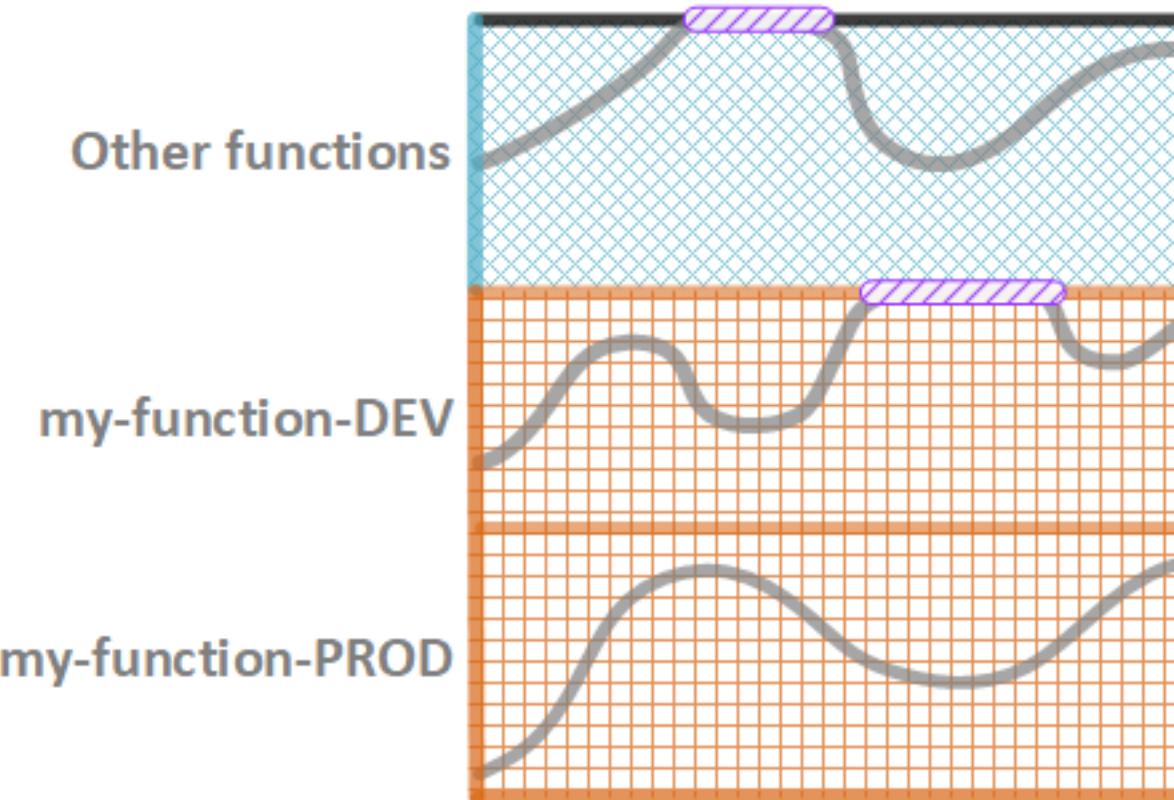
要为函数预留并发性

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Concurrency (并发) 中，选择 Reserve concurrency (预留并发)。
4. 输入要为该函数预留的并发数量。
5. 选择 Save。

您最多可以预留所显示的 Unreserved account concurrency (未预留账户并发) 值；对于没有预留并发的函数，可预留的并发数量为该值减去 100。要限制函数，请将预留并发设置为零。这将停止处理任何事件，直到您删除限制。

以下示例显示了两个具有预留并发池的函数，以及其他函数使用的非预留并发池。当池中的所有并发都已使用时，会出现限制错误。

Reserved Concurrency



图例

- 函数并发
- 预留并发
- 非预留并发
- 限制

预留并发具有以下效果。

- 其他函数无法阻止您的函数扩展 – 在没有预留并发的同一区域中，您所有账户的函数共享未预留并发的池。如果没有预留并发，其他函数可能会耗尽所有可用的并发，从而导致您的函数无法根据需要进行扩展。
- 您的函数不能无节制地扩展 – 预留并发还限制您的函数使用非预留池中的并发数量，从而限制了它的最大并发数量。您可以预留并发以防止您的函数使用该区域中的所有可用并发，或者防止下游资源过载。

设置每函数并发会影响可用于其他函数的并发池。为避免出现问题，请限制可以使用 PutFunctionConcurrency 和 DeleteFunctionConcurrency API 操作的用户数量。

配置预配置并发

要管理版本或别名的预配置并发设置，请使用 Lambda 控制台。

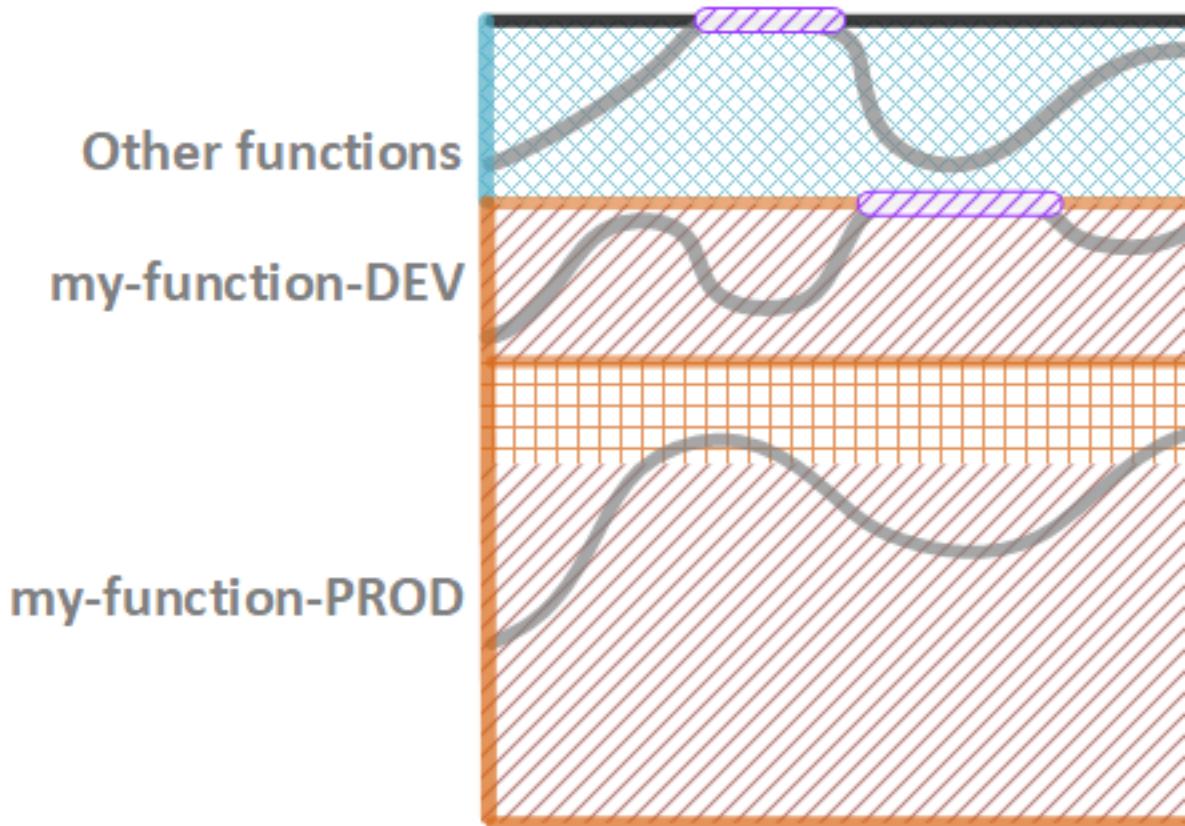
要为别名预留并发

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Provisioned concurrency configurations (预配置并发配置) 下，选择 Add (添加)。
4. 选择别名或版本。
5. 输入要分配的预配置并发数量。
6. 选择 Save。

您可以从函数配置页面管理所有别名和版本的预配置并发。预配置并发配置的列表显示每个配置的分配进度。预配置并发设置也可以在每个版本和别名的配置页面上使用。

在以下示例中，使用预留和预配置的并发配置了 my-function-DEV 和 my-function-PROD 函数。对于 my-function-DEV，预留并发的整个池也是预配置的并发。在这种情况下，所有调用都在预配置并发上运行，或者受限制。对于 my-function-PROD，预留并发池的一部分是标准并发。当所有预配置的并发都在使用中时，函数在标准并发上扩展以满足任何其他请求。

Provisioned Concurrency with Reserved Concurrency

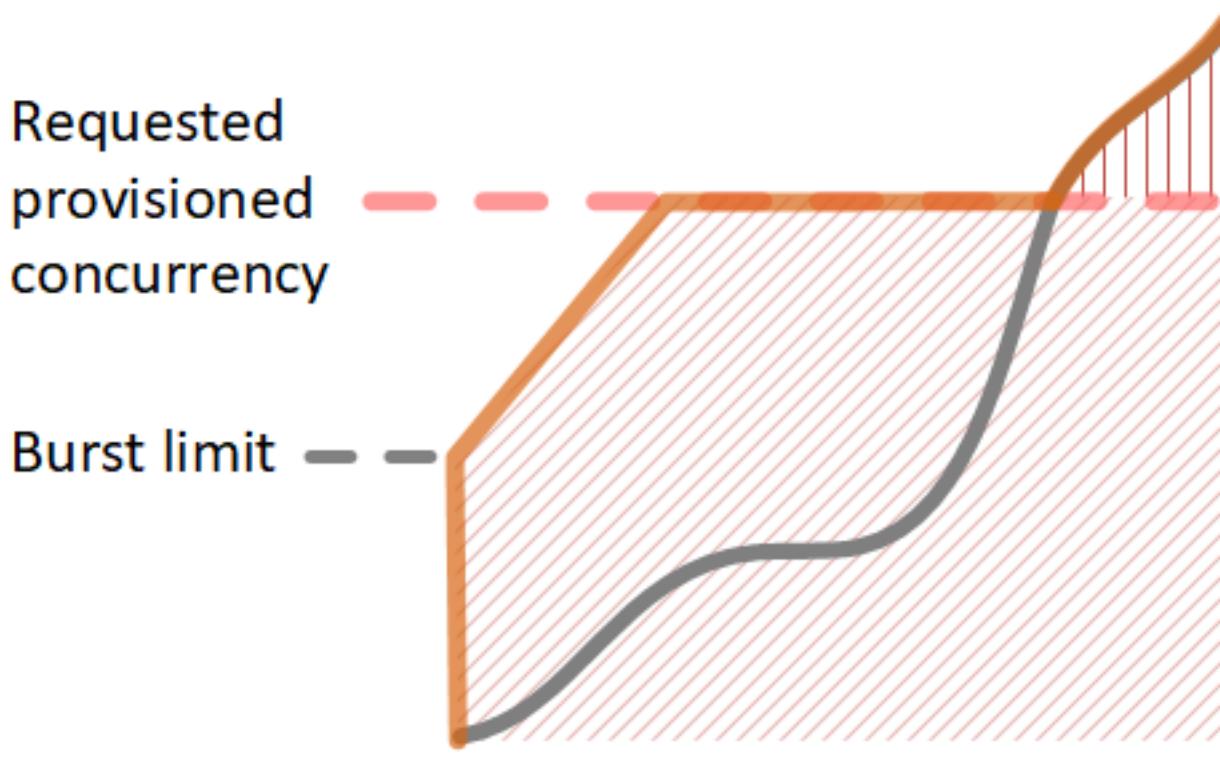


图例

- 函数并发
- 预留并发
- 预配置并发
- 非预留并发
- 限制

预配置并发不会在您配置它后立即联机。Lambda 会在一两分钟的准备时间后开始分配预配置并发。与函数在[负载下扩展 \(p. 94\)](#)的方式类似，最多可以同时初始化 3000 个函数实例，具体取决于区域。初始突增后，实例将按每分钟 500 个的稳定速率分配，直到请求完成。当您为同一区域中的多个函数或一个函数的多个版本请求预配置并发时，扩展限制适用于所有请求。

Function Scaling with Provisioned Concurrency



图例

- 函数实例
- 打开请求
- 预配置并发
- 标准并发

您函数的[初始化代码](#) (p. 17) 在分配期间每隔几个小时运行一次，因为正在运行的函数实例会被回收。在实例处理请求后，您可以在日志和[跟踪](#) (p. 371) 中查看初始化时间。但是，即使实例从不处理请求，也会对初始化进行计费。预配置并发连续运行，并且与初始化和调用成本分开计费。有关详细信息，请参阅 [AWS Lambda 定价](#)。

函数的每个版本只能有一个预配置并发配置。此配置可以直接在版本本身上，也可以在指向版本的别名上。两个别名无法为同一版本分配预配置并发。此外，您无法在指向未发布版本 (`$LATEST`) 的别名上分配预配置并发。

当您更改别名指向的版本时，预配置并发将从旧版本中解除分配，然后分配给新版本。您可以向拥有预配置并发的别名添加路由配置。但是，在路由配置到位时，您无法管理别名上的预配置并发设置。

Lambda 发出以下预配置并发指标：

预配置并发指标

- ProvisionedConcurrentExecutions
- ProvisionedConcurrencyInvocations
- ProvisionedConcurrencySpilloverInvocations
- ProvisionedConcurrencyUtilization

有关详细信息，请参阅[使用 AWS Lambda 函数指标 \(p. 368\)](#)。

使用 Lambda API 配置并发

要使用 AWS CLI 或 AWS 开发工具包管理并发设置和自动扩展，请使用以下 API 操作。

- PutFunctionConcurrency ([p. 525](#))
- GetFunctionConcurrency
- DeleteFunctionConcurrency ([p. 438](#))
- PutProvisionedConcurrencyConfig
- GetProvisionedConcurrencyConfig
- ListProvisionedConcurrencyConfigs
- DeleteProvisionedConcurrencyConfig
- GetAccountSettings ([p. 446](#))
- (Application Auto Scaling) [RegisterScalableTarget](#)
- (Application Auto Scaling) [PutScalingPolicy](#)

要使用 AWS CLI 配置预留并发，请使用 `put-function-concurrency` 命令。以下命令为名为 `my-function` 的函数预留 100 的并发：

```
$ aws lambda put-function-concurrency --function-name my-function --reserved-concurrent-executions 100
{
    "ReservedConcurrentExecutions": 100
}
```

要为函数分配预配置并发，请使用 `put-provisioned-concurrency-config`。以下命令为名为 `my-function` 的函数的 `BLUE` 别名分配 100 的并发：

```
$ aws lambda put-provisioned-concurrency-config --function-name my-function \
--qualifier BLUE --provisioned-concurrent-executions 100
{
    "Requested ProvisionedConcurrentExecutions": 100,
    "Allocated ProvisionedConcurrentExecutions": 0,
    "Status": "IN_PROGRESS",
    "LastModified": "2019-11-21T19:32:12+0000"
}
```

要配置 Application Auto Scaling 来管理预配置的并发，请使用 Application Auto Scaling 配置[目标跟踪扩展](#)。首先，将函数的别名注册为扩展目标。以下示例注册名为 `my-function` 的函数的 `BLUE` 别名：

```
$ 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
```

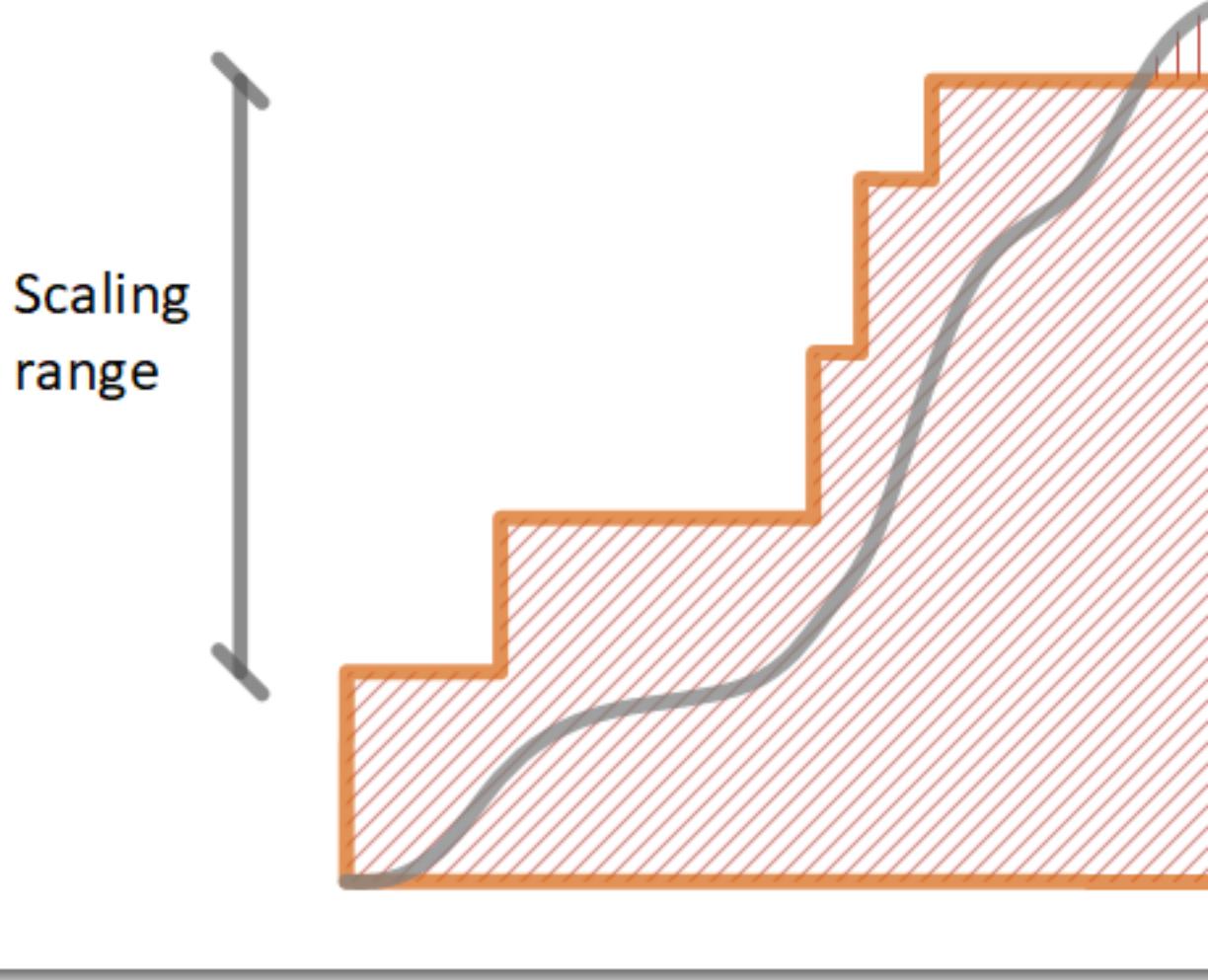
接下来，将扩展策略应用于目标。以下示例配置 Application Auto Scaling，调节别名的预配置并发配置，以使利用率保持接近 70%。

```
$ aws application-autoscaling put-scaling-policy --service-namespace lambda \
--scalable-dimension lambda:function:ProvisionedConcurrency --resource-id function:my-
function:BLUE \
--policy-name my-policy --policy-type TargetTrackingScaling \
--target-tracking-scaling-policy-configuration '{ "TargetValue": 
0.7, "PredefinedMetricSpecification": { "PredefinedMetricType": 
"LambdaProvisionedConcurrencyUtilization" }}'
{
    "PolicyARN": "arn:aws:autoscaling:us-east-2:123456789012:scalingPolicy:12266ddb-1524-
xmpl-a64e-9a0a34b996fa:resource/lambda/function:my-function:BLUE:policyName/my-policy",
    "Alarms": [
        {
            "AlarmName": "TargetTracking-function:my-function:BLUE-AlarmHigh-aed0e274-
xmpl-40fe-8cba-2e78f000c0a7",
            "AlarmARN": "arn:aws:cloudwatch:us-east-2:123456789012:alarm:TargetTracking-
function:my-function:BLUE-AlarmHigh-aed0e274-xmpl-40fe-8cba-2e78f000c0a7"
        },
        {
            "AlarmName": "TargetTracking-function:my-function:BLUE-AlarmLow-7e1a928e-
xmpl-4d2b-8c01-782321bc6f66",
            "AlarmARN": "arn:aws:cloudwatch:us-east-2:123456789012:alarm:TargetTracking-
function:my-function:BLUE-AlarmLow-7e1a928e-xmpl-4d2b-8c01-782321bc6f66"
        }
    ]
}
```

Application Auto Scaling 在 CloudWatch 中创建两个警报。当预配置的并发利用率持续超过 70% 时，第一个警报会触发。发生这种情况时，Application Auto Scaling 会分配更多的预配置并发以降低利用率。当利用率持续低于 63%（70% 目标的 90%）时，第二个警报会触发。发生这种情况时，Application Auto Scaling 会减少别名的预配置并发。

在以下示例中，函数根据利用率在预配置并发的最小数量和最大数量之间缩放。当打开请求的数量增加时，Application Auto Scaling 将大幅增加预配置并发，直到达到配置的最大数量。该函数继续扩展标准并发，直到利用率开始下降。当利用率持续较低时，Application Auto Scaling 将定期小幅减少预配置并发。

Autoscaling with Provisioned Concurrency



图例

- 函数实例
- 打开请求
- 预配置并发
- 标准并发

要查看您的账户在某个区域中的并发限制，请使用 `get-account-settings`。

```
$ aws lambda get-account-settings
{
    "AccountLimit": {
        "TotalCodeSize": 80530636800,
```

```
    "CodeSizeUnzipped": 262144000,  
    "CodeSizeZipped": 52428800,  
    "ConcurrentExecutions": 1000,  
    "UnreservedConcurrentExecutions": 900  
},  
"AccountUsage": {  
    "TotalCodeSize": 174913095,  
    "FunctionCount": 52  
}  
}
```

AWS Lambda 函数版本

您可以使用版本来管理您的 AWS Lambda 函数的部署。例如，您可以发布函数的新版本以用于 Beta 测试，而不会影响稳定生产版本的用户。

每当您发布 Lambda 函数时，系统都会创建函数的新版本。新版本是函数的未发布版本的副本。函数版本包括以下信息：

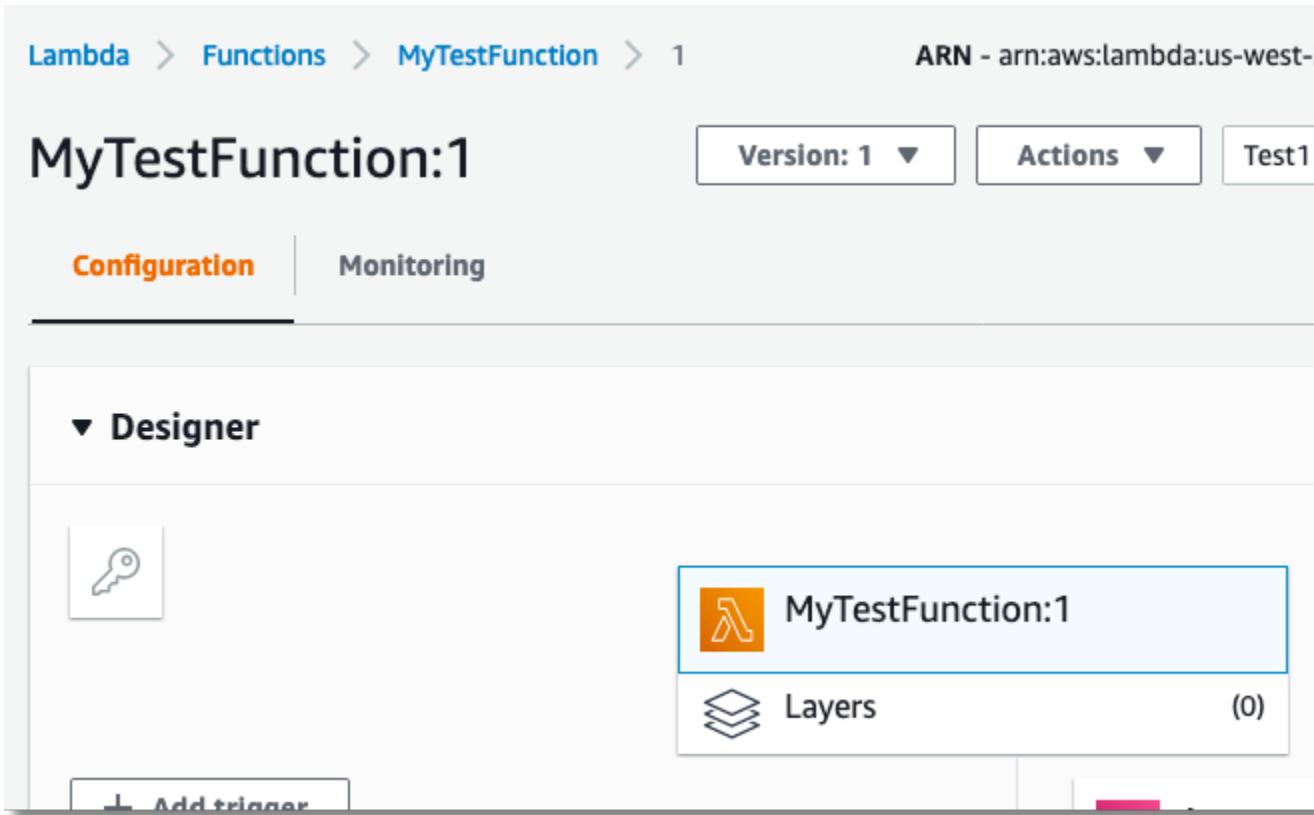
- 函数代码以及所有关联的依赖项。
- 执行该函数的 Lambda 运行时。
- 所有函数设置，包括环境变量。
- 用于标识函数的此版本的唯一的 Amazon 资源名称 (ARN)。

只能在函数的未发布版本上更改函数代码和设置。当您发布版本后，代码和大多数设置将被锁定，从而确保为该版本的用户提供一致的体验。有关配置函数设置的更多信息，请参阅[在 AWS Lambda 控制台中配置函数 \(p. 47\)](#)。

创建函数的新版本

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择您希望发布的函数。
3. 在操作中，选择发布新版本。

在您发布函数的第一个版本后，Lambda 控制台将显示可用版本的下拉菜单。在设计器面板中，函数名称的末尾会显示一个版本限定符。



要查看函数的当前版本，请选择一个函数，然后选择限定符。在展开的限定符菜单中，选择版本选项卡。版本面板显示选定函数的版本列表。如果您尚未发布选定函数的版本，版本面板将仅列出 \$LATEST 版本。

使用 Lambda API 管理版本

要发布某个函数的版本，请使用 [PublishVersion \(p. 518\)](#) API 操作。

以下示例发布函数的新版本。响应返回有关新版本的配置信息，包括版本号和具有版本后缀的函数 ARN。

```
$ aws lambda publish-version --function-name my-function
{
    "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",
    ...
}
```

使用版本

可以使用 ARN 引用您的 Lambda 函数。有两个与该初始版本关联的 ARN：

- 限定的 ARN – 具有版本后缀的函数 ARN。

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

- 非限定的 ARN – 不具有版本后缀的函数 ARN。

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

您可以在所有相关的操作中使用该非限定 ARN。不过，您无法使用该 ARN 创建别名。

如果您决定不发布函数版本，可以在事件源映射中使用限定或非限定 ARN 调用函数。

只有在代码从未发布过，或者代码与最近发布的版本相比发生了更改时，Lambda 才会发布新的函数版本。如果没有任何更改，则函数版本将保留为最近发布的版本。

每个 Lambda 函数版本都有唯一的 ARN。发布版本后，您无法更改 ARN 或函数代码。

资源策略

使用[基于资源的策略 \(p. 33\)](#)为服务、资源或账户提供对您的函数的访问时，这种权限的范围取决于您是将权限应用于某个函数还是函数的某个版本：

- 如果使用限定函数名称（如 helloworld:1），该权限仅在使用限定 ARN 调用 helloworld 函数版本 1 时有效。使用任何其他 ARN 时会导致权限错误。
- 如果使用非限定的函数名称（例如 helloworld），则权限仅在使用非限定的函数 ARN 调用 helloworld 函数时有效。使用任何其他 ARN（包括 \$LATEST）时会导致权限错误。
- 如果使用 \$LATEST 限定函数名称（如 helloworld:\$LATEST），该权限仅在使用限定 ARN 调用 helloworld 函数时有效。使用非限定 ARN 时会导致权限错误。

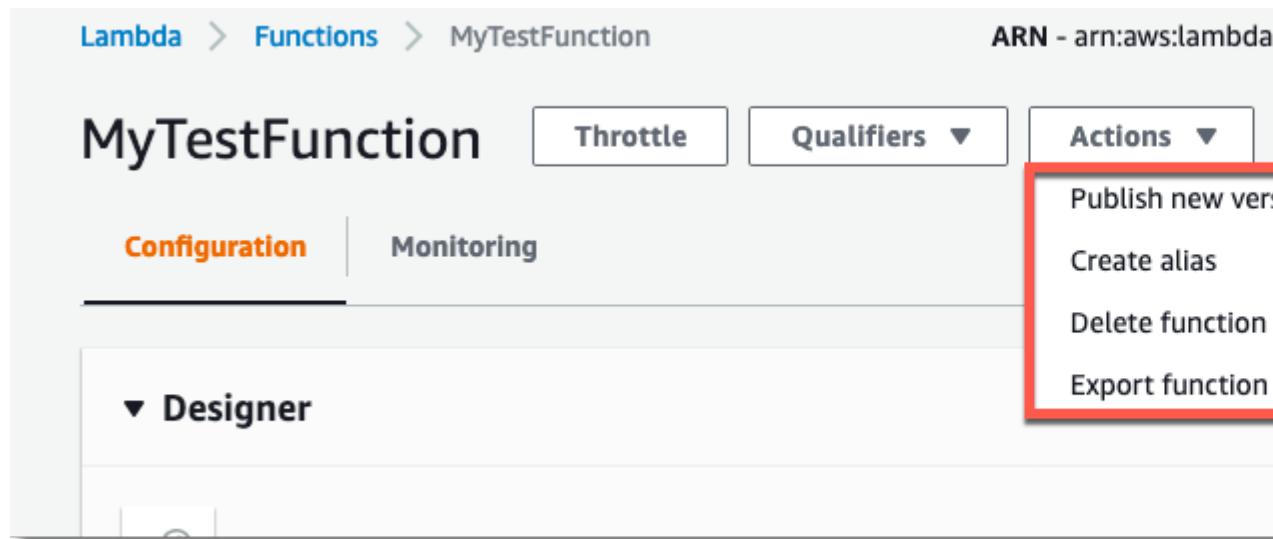
可以通过使用函数别名，简化事件源和资源策略的管理。有关更多信息，请参阅[AWS Lambda 函数别名 \(p. 65\)](#)。

AWS Lambda 函数别名

您可以为 AWS Lambda 函数创建一个或多个别名。Lambda 别名类似于指向特定 Lambda 函数版本的指针。用户可以使用别名 ARN 来访问函数版本。

创建别名

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在操作中，选择创建别名。



4. 在创建新别名表单中，输入别名的名称以及可选的描述。为此别名选择函数版本。

要查看当前为某个函数定义的别名，请选择限定符，然后选择别名选项卡。

使用 Lambda API 管理别名

要创建别名，请使用 `create-alias` 命令。

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

要更改别名以便指向函数的新版本，请使用 `update-alias` 命令。

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

以上步骤中的 AWS CLI 命令对应于下面的 AWS Lambda API：

- [CreateAlias \(p. 411\)](#)
- [UpdateAlias \(p. 543\)](#)

使用别名

每个别名都有唯一的 ARN。别名只能指向函数版本，而不能指向其他别名。可以更新别名以便指向函数的新版本。

事件源（例如 Amazon S3）会调用您的 Lambda 函数。这些事件源维护一个映射，该映射标识在发生事件时要调用的函数。如果在映射配置中指定 Lambda 函数别名，则当函数版本发生更改时，您不需要更新映射。

在资源策略中，您可以为事件源授予权限，以便使用您的 Lambda 函数。如果在策略中指定别名 ARN，则当函数版本发生更改时，您不需要更新策略。

资源策略

使用[基于资源的策略 \(p. 33\)](#)为服务、资源或账户提供对您的函数的访问时，这种权限的范围取决于您是将权限应用于别名、版本还是函数。如果您使用别名（例如 `helloworld:PROD`），则权限仅在使用别名 ARN 调用 `helloworld` 函数时有效。如果您使用版本 ARN 或函数 ARN，则会遇到权限错误。这包括别名指向的版本 ARN。

例如，以下 AWS CLI 命令向 Amazon S3 授予调用 `helloworld` Lambda 函数的 PROD 别名的权限。请注意，`--qualifier` 参数指定别名。

```
$ 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
```

在此情况下，Amazon S3 现在可以调用 PROD 别名。然后，Lambda 可执行 PROD 别名引用的 `helloworld` Lambda 函数版本。您必须在 S3 存储桶的通知配置中使用 PROD 别名 ARN 才能使其生效。

别名路由配置

对别名使用路由配置，将一部分流量发送到第二个函数版本。例如，您可以通过配置别名以便将大部分流量发送到现有版本，并且仅将一小部分流量发送到新版本，来降低部署新版本的风险。

您可以将别名最多指向两个 Lambda 函数版本。版本必须满足以下条件：

- 两个版本必须具有相同的 IAM 执行角色。
- 两个版本必须具有相同的[死信队列](#)配置，或者都没有死信队列配置。
- 这两个版本必须都已发布。别名不能指向 `$LATEST`。

对别名配置路由

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 确保函数至少具有两个已发布版本。为此，请选择限定符，然后选择版本，以便显示版本列表。如果您需要创建其他版本，请按照[AWS Lambda 函数版本 \(p. 63\)](#)中的说明操作。
4. 从操作菜单中，选择创建别名。
5. 在创建新别名窗口中，输入名称的值，可以选择性地输入描述的值，然后选择别名所引用的 Lambda 函数的版本。
6. 在其他版本中，指定以下项：
 - a. 选择第二个 Lambda 函数版本。
 - b. 键入函数的权重值。权重是在调用别名时分配给该版本的流量百分比。第一个版本接收剩余权重。
例如，如果为其他版本指定 10%，则会自动为第一个版本分配 90%。
7. 选择 Create。

配置别名路由

使用 `create-alias` 和 `update-alias` 命令可以配置两个函数版本之间的流量权重。创建或更新别名时，您可以使用 `routing-config` 参数指定流量权重。

以下示例为 Lambda 函数创建别名（名为 `routing-alias`）。该别名指向函数的版本 1。函数的版本 2 接收 3% 的流量。其余 97% 的流量路由到版本 1。

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

使用 update-alias 命令可提高流入版本 2 的流量的百分比。在下面的示例中，将该流量提高到 5%。

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

要将所有流量路由到版本 2，请使用 UpdateAlias 命令更改 function-version 属性，使别名指向版本 2。该命令还将重置路由配置。

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

以上步骤中的 CLI 命令对应于下面的 AWS Lambda API 操作：

- [CreateAlias \(p. 411\)](#)
- [UpdateAlias \(p. 543\)](#)

确定已调用的版本

配置两个函数版本之间的流量权重时，可以通过两种方法确定已调用的 Lambda 函数版本：

- CloudWatch Logs – 对于每个函数调用，Lambda 自动将包含调用的版本 ID 的 START 日志条目发出到 CloudWatch Logs。以下是示例：

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

对于别名调用，Lambda 使用 Executed Version 维度按执行的版本筛选指标数据。有关更多信息，请参阅[使用 AWS Lambda 函数指标 \(p. 368\)](#)。

- 响应负载（同步调用）– 同步函数调用的响应包含 x-amz-executed-version 标头以指示已调用的函数版本。

AWS Lambda 层

您可以将 Lambda 函数配置为以层的形式拉入其他代码和内容。层是包含库、[自定义运行时 \(p. 111\)](#)或其他依赖项的 ZIP 存档。利用层，您可以在函数中使用库，而不必将库包含在部署程序包中。

使用层可以使您的部署包保持较小，从而使开发变得更轻松。您可以避免在使用函数代码安装和打包依赖项时可能出现的错误。对于 Node.js、Python 和 Ruby 函数，只要将部署程序包保持在 3 MB 以下，就可以在[Lambda 控制台中开发函数代码 \(p. 5\)](#)。

Note

一个函数一次可使用最多 5 层。函数和所有层的总解压缩大小不能超出 250 MB 的解压缩部署程序包大小限制。有关更多信息，请参阅[AWS Lambda 限制 \(p. 27\)](#)。

您可以创建层，也可以使用由 AWS 和其他 AWS 客户发布的层。层支持[基于资源的策略 \(p. 72\)](#)，这些策略用于向特定 AWS 账户、[AWS Organizations](#) 或所有账户授予层使用权限。

在函数执行环境中，层将提取到 /opt 目录。每个运行时将查找位于 /opt 下的不同位置的库（具体取决于语言）。[构建层 \(p. 71\)](#)，以便函数代码无需额外配置即可访问库。

您还可以使用 AWS 无服务器应用程序模型 (AWS SAM) 管理层和函数的层配置。有关说明，请参阅 AWS 无服务器应用程序模型 开发人员指南 中的[声明无服务器资源](#)。

小节目录

- [将函数配置为使用层 \(p. 69\)](#)
- [管理层 \(p. 69\)](#)
- [在层中包括库依赖项 \(p. 71\)](#)
- [层权限 \(p. 72\)](#)

将函数配置为使用层

在函数创建期间或之后，您可以在函数配置中指定最多 5 层。选择要使用的层的特定版本。如果您稍后要使用其他版本，请更新函数的配置。

要将层添加到函数，请使用 `update-function-configuration` 命令。以下示例将添加两个层：一个来自与函数相同的账户的层，一个来自其他账户的层。

```
$ 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
{
    "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",
    ...
}
```

您必须通过提供层版本的完整 ARN 来指定要使用的每个层的版本。在将层添加到已具有层的函数时，新层会覆盖上一个列表。每次更新层配置时都包括所有层。要删除所有层，请指定空列表。

```
$ aws lambda update-function-configuration --function-name my-function --layers []
```

您的函数在 `/opt` 目录中执行期间可访问层内容。层按指定的顺序应用，这将合并任何具有相同名称的文件夹。如果同一文件出现在多个层中，则使用最后应用的层中的版本。

层的创建者可删除您使用的层的版本。在发生此情况时，您的函数将继续运行，就好像层版本仍然存在一样。但在更新层配置时，必须先删除对已删除版本的引用。

管理层

要创建层，请将 `publish-layer-version` 命令与名称、描述、ZIP 存档和与层兼容的[运行时 \(p. 108\)](#)的列表结合使用。运行时列表是可选的，但它使层更易发现。

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

每次调用 `publish-layer-version` 时，都将创建一个新版本。使用层的函数将直接引用层版本。您可以在现有层版本上[配置权限 \(p. 72\)](#)，但要进行任何其他更改，则必须创建新版本。

要查找与您函数的运行时兼容的层，请使用 `list-layers` 命令。

```
$ aws lambda list-layers --compatible-runtime python3.8
{
    "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",
                ]
            }
        }
    ]
}
```

您可以忽略运行时选项来列出所有层。响应中的详细信息反映层的最新版本。使用 `list-layer-versions` 查看层的所有版本。要查看有关版本的更多信息，请使用 `get-layer-version`。

```
$ aws lambda get-layer-version --layer-name my-layer --version-number 2
{
    "Content": {
        "Location": "https://awslambda-us-east-2-layers.s3.us-east-2.amazonaws.com/
snapshots/123456789012/my-layer-91e9ea6e-492d-4100-97d5-a4388d442f3f?
versionId=GmvPV.3090EpkfN...", 
        "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:2",
    "Description": "My layer",
    "CreatedDate": "2018-11-15T00:37:46.592+0000",
    "Version": 2,
    "CompatibleRuntimes": [
        "python3.6",
        "python3.7",
        "python3.8"
    ]
}
```

响应中的链接可用于下载层存档，并且其有效时间为 10 分钟。要删除层版本，请使用 `delete-layer-version` 命令。

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

在删除一个层版本后，您无法再将函数配置为使用该层版本。但是，已使用此版本的任何函数仍能访问它。层名称永远不重复使用版本号。

在层中包括库依赖项

您可通过将运行时依赖项放入层中来将这些依赖项移出函数代码。Lambda 运行时包括 `/opt` 目录中的路径以确保您的函数代码有权访问层中包含的库。

要在层中包含库，请将库放入运行时支持的文件夹之一。

- Node.js – `nodejs/node_modules`、`nodejs/node8/node_modules` (`NODE_PATH`)

Example 适用于 Node.js 的 AWS X-Ray 开发工具包

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

- Python – `python/python/lib/python3.8/site-packages` (站点目录)

Example Pillow

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

- Java – `java/lib` (类路径)

Example Jackson

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

- Ruby – `ruby/gems/2.5.0` (`GEM_PATH`)、`ruby/lib` (`RUBY_LIB`)

Example JSON

```
json.zip
# ruby/gems/2.5.0/
```

```
| build_info
| cache
| doc
| extensions
| gems
| # json-2.1.0
# specifications
# json-2.1.0.gemspec
```

- 全部 - bin (PATH)、lib (LD_LIBRARY_PATH)

Example JQ

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

有关 Lambda 执行环境中的路径设置的更多信息，请参阅[运行时环境变量 \(p. 51\)](#)。

层权限

层使用权限是在资源上进行管理的。要配置包含层的函数，您需要权限才能对层版本调用 `GetLayerVersion`。对于您账户中的函数，可通过您的[用户策略 \(p. 37\)](#)或函数的[基于资源的策略 \(p. 33\)](#)获取此权限。要使用另一个账户中的层，您需要用户策略的权限，并且另一个账户的拥有者必须使用基于资源的策略向您的账户授予权限。

要向另一个账户授予权限，请使用 `add-layer-version-permission` 命令向层版本的权限策略添加语句。在每个语句中，您可以向单个账户、所有账户或组织授予权限。

```
$ aws lambda add-layer-version-permission --layer-name xray-sdk-nodejs --statement-id
xaccount \
--action lambda:GetLayerVersion --principal 210987654321 --version-number 1 --output text
e210ffdc-e901-43b0-824b-5fc0dd26d16 {"Sid":"xaccount","Effect":"Allow","Principal":
{"AWS":"arn:aws:iam::210987654321:root"},"Action":"lambda:GetLayerVersion","Resource":"arn:aws:lambda:u
east-2:123456789012:layer:xray-sdk-nodejs:1"}
```

权限仅适用于层的单一版本。每次创建新的层版本时都重复此过程。

有关更多示例，请参阅[向其他账户授予权限 \(p. 36\)](#)。

配置 Lambda 函数以访问 VPC 中的资源

您可以配置函数来连接到您账户中 Virtual Private Cloud (VPC) 中的私有子网。使用 Amazon Virtual Private Cloud (Amazon VPC) 为资源（如数据库、缓存实例或内部服务）创建专用网络。将您的函数连接到 VPC 来在执行期间访问私有资源。

将函数连接到 VPC

- 打开[Lambda 控制台](#)。
- 选择函数。
- 在 VPC 下，选择 `Edit` (编辑)。
- 选择 `Custom VPC` (自定义 VPC)。
- 选择 VPC、子网和安全组。

Note

将函数连接到私有子网以访问私有资源。如果函数需要互联网访问权限，请使用 [NAT \(p. 74\)](#)。将函数连接到公有子网不会授予其互联网访问权限或公有 IP 地址。

6. 选择保存。

将函数连接到 VPC 时，Lambda 在函数的 VPC 配置中为每个安全组和子网组合创建一个弹性网络接口。此过程可能需要大约一分钟。在此期间，您无法执行针对该函数的其他操作，例如[创建版本 \(p. 63\)](#)或更新函数的代码。对于新函数，在函数状态从 Pending 转换为 Active 之前，您无法调用该函数。对于现有函数，在更新过程中您仍然可以调用旧版本。有关函数状态的更多信息，请参阅[使用 Lambda API 监控函数的状态 \(p. 92\)](#)。

连接到相同子网的多个函数共享网络接口，因此将其他函数连接到已具有 Lambda 托管网络接口的子网要快得多。但是，如果您有许多函数或非常繁忙的函数，Lambda 可能会创建额外的网络接口。

如果您的函数长时间未活动，Lambda 会回收其网络接口，且函数变为 Idle。调用空闲函数以重新将其激活。第一次调用失败，并且函数再次进入挂起状态，直到网络接口可用。

Lambda 函数无法直接连接到具有[专用实例租赁](#)的 VPC。要连接到专用 VPC 中的资源，请使其与具有默认租赁的第二个 VPC 对等。

VPC 教程

- 教程：配置 Lambda 函数以访问 Amazon VPC 中的 Amazon RDS (p. 230)
- 教程：配置 Lambda 函数以访问 Amazon VPC 中的 Amazon ElastiCache (p. 198)

小目录

- [执行角色和用户权限 \(p. 73\)](#)
- [使用 Lambda API 配置 Amazon VPC 访问权限 \(p. 74\)](#)
- [对 VPC 连接函数的 Internet 和服务访问 \(p. 74\)](#)
- [VPC 配置示例 \(p. 74\)](#)

执行角色和用户权限

Lambda 使用函数的权限来创建和管理网络接口。要连接到 VPC，您的函数的执行角色必须具有以下权限。

执行角色权限

- ec2:CreateNetworkInterface
- ec2:DescribeNetworkInterfaces
- ec2:DeleteNetworkInterface

这些权限包含在 AWSLambdaVPCAccessExecutionRole 托管策略中。

配置 VPC 连接时，Lambda 使用您的权限来验证网络资源。要配置函数以连接到 VPC，您的 IAM 用户需要以下权限。

用户权限

- ec2:DescribeSecurityGroups
- ec2:DescribeSubnets

- ec2:DescribeVpcs

使用 Lambda API 配置 Amazon VPC 访问权限

您可以使用以下 API 将函数连接到 VPC。

- [CreateFunction \(p. 421\)](#)
- [UpdateFunctionConfiguration \(p. 560\)](#)

要在使用 AWS CLI 创建期间将函数连接到 VPC，请将 `vpc-config` 选项与私有子网 ID 和安全组列表结合使用。以下示例创建一个连接到 VPC（包含两个子网和一个安全组）的函数。

```
$ 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
```

要连接到现有函数，请将 `vpc-config` 选项与 `update-function-configuration` 命令结合使用。

```
$ aws lambda update-function-configuration --function-name my-function \
--vpc-config
SubnetIds=subnet-071f712345678e7c8,subnet-07fd123456788a036,SecurityGroupIds=sg-085912345678492fb
```

要断开函数与 VPC 的连接，请使用子网和安全组的空白列表更新函数配置。

```
$ aws lambda update-function-configuration --function-name my-function \
--vpc-config SubnetIds=[],SecurityGroupIds=[]
```

对 VPC 连接函数的 Internet 和服务访问

默认情况下，Lambda 在可访问 AWS 服务和 Internet 的安全 VPC 中运行您的函数。VPC 由 Lambda 拥有，并且不连接到您账户的默认 VPC。在将函数连接到您账户中的 VPC 时，除非 VPC 提供访问权限，否则它将无法访问 Internet。

Note

有多个服务提供 [VPC 终端节点](#)。您可以使用 VPC 终端节点从 VPC 内连接到 AWS 服务，而无需 Internet 访问权限。

从私有子网访问 Internet 需要网络地址转换 (NAT)。要使您的函数能够访问 Internet，请将出站流量路由到公共子网中的 NAT 网关。NAT 网关具有公共 IP 地址，可以通过 VPC 的 Internet 网关连接到 Internet。有关更多信息，请参阅 Amazon VPC 用户指南 中的 [NAT 网关](#)。

VPC 配置示例

本指南的 GitHub 存储库中提供了您可以与 Lambda 函数一起使用的 VPC 配置的示例 AWS CloudFormation 模板。存在以下两种模板：

- [vpc-private.yaml](#) – 此 VPC 具有两个私有子网以及适用于 Amazon Simple Storage Service 和 Amazon DynamoDB 的 VPC 终端节点。您可以使用此模板为不需要 Internet 访问权限的函数创建 VPC。此配置支持与 AWS 软件开发包一起使用 Amazon S3 和 DynamoDB，以及通过本地网络连接访问相同 VPC 中的数据资源。

- [vpc-privatepublic.yaml](#) – 此 VPC 具有两个私有子网、VPC 终端节点、一个具有 NAT 网关的公有子网，以及一个 Internet 网关。来自私有子网中函数的 Internet 相关流量将通过路由表路由到 NAT 网关。

要使用模板创建 VPC，请选择[AWS CloudFormation 控制台](#)中的 Create stack (创建堆栈)，然后按照说明操作。

配置 Lambda 函数的数据访问

您可以使用 Lambda 控制台为您的函数创建 Amazon RDS Proxy 数据库代理。数据库代理管理数据库连接池，并从函数中继查询。这使得函数能够在不耗尽数据库连接的情况下达到高并发 (p. 17) 级别。

创建数据库代理

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 选择 Add database proxy (添加数据库代理)。
4. 配置以下选项。
 - 代理标识符 – 代理的名称。
 - RDS 数据库实例 – MySQL 5.6 或 MySQL 5.7 数据库实例或集群。
 - 密钥 – 包含数据库用户名和密码的 Secrets Manager 密钥。

Example 密钥

```
{  
    "username": "admin",  
    "password": "e2abcecxmpldc897"  
}
```

5. 选择 Add。
- IAM 角色 – 拥有使用密钥的权限和允许 Amazon RDS 代入角色的信任策略的 IAM 角色。

代理创建需要几分钟。当代理可用时，请将函数配置为连接到代理终端节点而不是数据库终端节点。

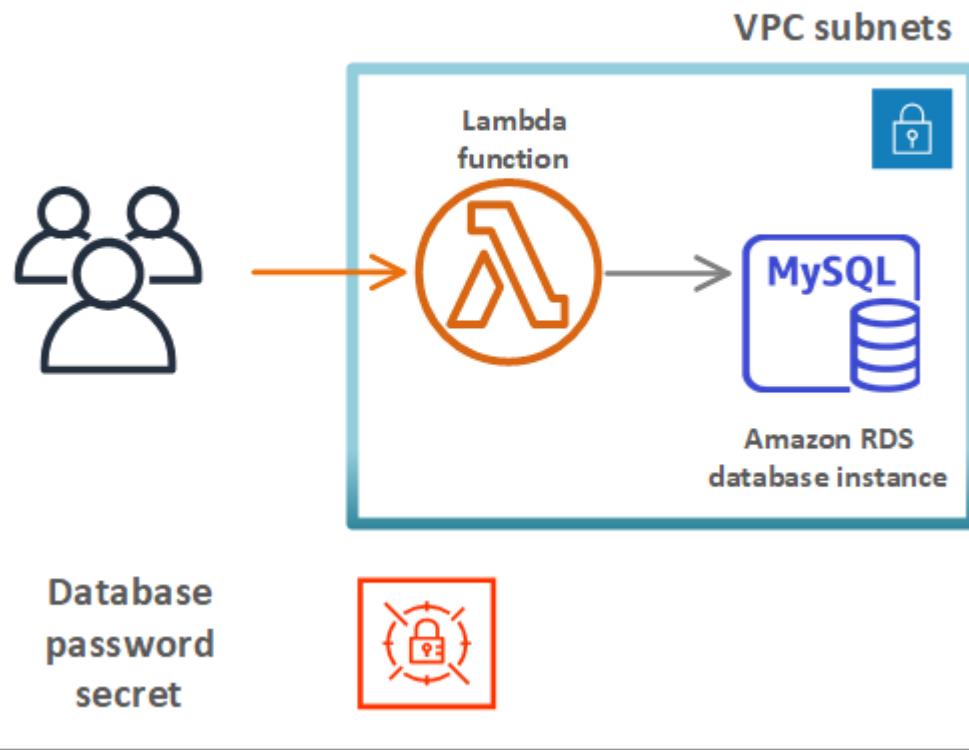
适用标准 [Amazon RDS 代理定价](#)。有关更多信息，请参阅 Amazon Aurora 用户指南 中的[使用 Amazon RDS 代理管理连接](#)。

示例应用程序

使用本指南的 GitHub 存储库中提供的 Amazon RDS 数据库演示 Lambda 用法的示例应用程序。共有两个应用程序：

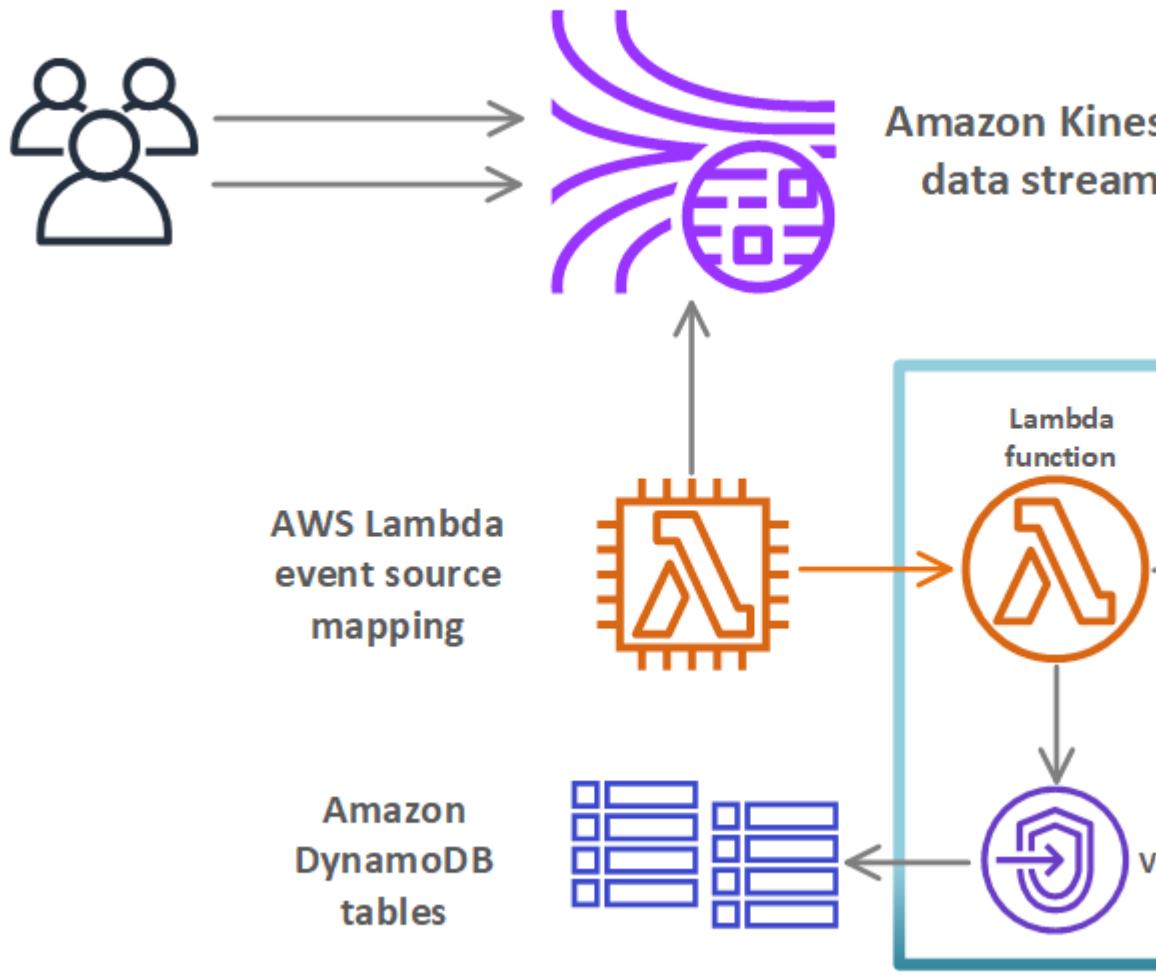
- [RDS MySQL](#) – AWS CloudFormation 模板 `template-vpcrds.yaml` 在私有 VPC 中创建 MySQL 5.7 数据库。在示例应用程序中，Lambda 函数将查询委托给数据库。函数和数据库模板都使用 Secrets Manager 来访问数据库凭证。

RDS MySQL Application



- [列表管理器](#) – 处理器函数从 Kinesis 流读取事件。它使用事件中的数据来更新 DynamoDB 表，并将事件的副本存储在 MySQL 数据库中。

List Manager Application



要使用示例应用程序，请按照 GitHub 存储库中的说明操作：[RDS MySQL](#)、[列表管理器](#)。

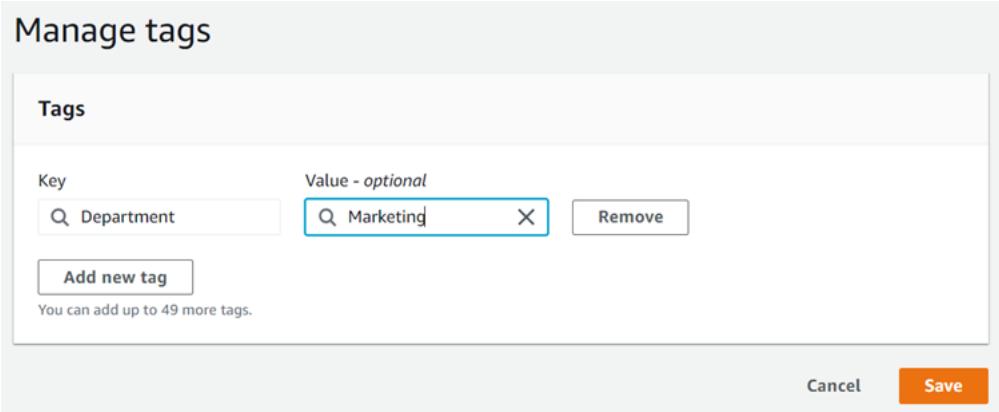
标记 Lambda 函数

您可以通过为 Lambda 函数添加标签，按所有者、项目或部门组织函数。标签是各类 AWS 服务均支持的自由格式键-值对，用于筛选资源和向账单报告添加详细信息。

为函数添加标签

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Tags (标签) 下，选择 Manage tags (管理标签)。

4. 输入密钥和值。要添加其他标签，请选择 Add new tag (添加新标签)。

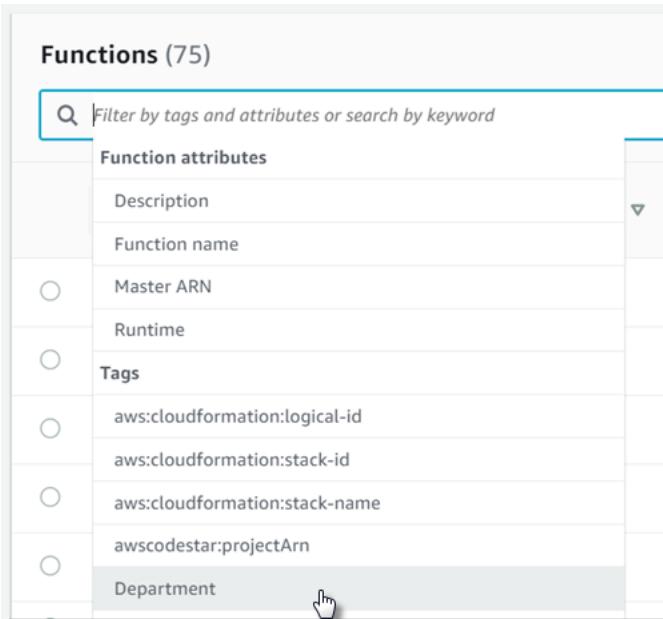


5. 选择保存。

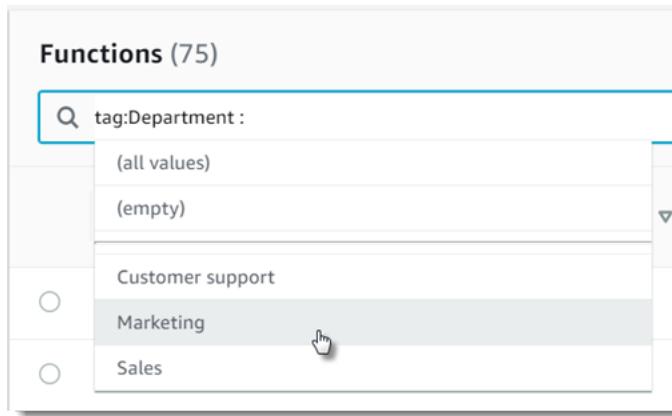
您可以使用 Lambda 控制台或 AWS 资源组 API 根据标签的存在与否或者标签值来筛选函数。标签应用在函数级别，而不是应用于版本或别名。标签不会包含在发布版本时为版本特定配置创建的快照中。

使用标签过滤函数

1. 打开 Lambda 控制台 [函数页面](#)。
2. 在搜索栏中单击可查看函数属性和标签键的列表。



3. 选择一个标签键可查看当前区域中正在使用的值的列表。
4. 选择一个值以查看具有该值的函数，或者选择 (all values) (所有值) 查看具有含有此键的所有函数。



搜索栏还支持搜索标签键。输入 `tag` 以仅查看标签键列表，或者开始输入键名称，在列表中查找该键。

借助 AWS Billing and Cost Management，您可以使用标签来自定义账单报告并创建成本分配报告。有关更多信息，请参阅 AWS 账单和成本管理用户指南中的[月度成本分配报告](#)和[使用成本分配标签](#)。

小目录

- [通过 AWS CLI 使用标签 \(p. 79\)](#)
- [标签键和值要求 \(p. 80\)](#)

通过 AWS CLI 使用标签

创建新 Lambda 函数时，可以使用 `--tags` 选项包含标签。

```
$ 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
```

要将标签添加到现有函数，请使用 `tag-resource` 命令。

```
$ aws lambda tag-resource \
--resource arn:aws:lambda:us-east-2:123456789012:function:my-function \
--tags Department=Marketing,CostCenter=1234ABCD
```

要删除标签，请使用 `untag-resource` 命令。

```
$ aws lambda untag-resource --resource function arn \
--tag-keys Department
```

如果您要查看应用于特定的 Lambda 函数的标签，可以使用以下任一 Lambda API 命令：

- [ListTags \(p. 509\)](#) – 由您提供 Lambda 函数 ARN (Amazon 资源名称)，以查看与该函数关联的标签列表：

```
$ aws lambda list-tags --resource function arn
```

- [GetFunction \(p. 455\)](#) – 由您提供 Lambda 函数的名称，以查看与该函数关联的标签列表：

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

您还可以使用 AWS Tagging Service 的 [GetResources API](#)，根据标签筛选您的资源。GetResources API 最多可接收 10 个筛选条件，每个筛选条件包含一个标签键和最多 10 个标签值。提供具有 "ResourceType" 的 GetResources，可按特定资源类型进行筛选。有关 AWS Tagging Service 的更多信息，请参阅[使用资源组](#)。

标签键和值要求

以下要求适用于标签：

- 每个资源的标签数上限 – 50
- 最大密钥长度 – 128 个 Unicode 字符 (采用 UTF-8 格式)
- 最大值长度 – 256 个 Unicode 字符 (采用 UTF-8 格式)
- 标签键和值要区分大小写。
- 请勿在标签名称或值中使用 aws: 前缀，因为它专为 AWS 使用预留。您无法编辑或删除带此前缀的标签名称或值。具有此前缀的标签不计入每个资源的标签数限制。
- 如果您的标记方案将在多个服务和资源中使用，请记得其他服务可能对允许使用的字符有限制。通常允许使用的字符包括：可用 UTF-8 格式表示的字母、空格和数字以及特殊字符 + - = . _ : / @。

调用 AWS Lambda 函数

您可以使用 Lambda 控制台、Lambda API、AWS 开发工具包、AWS CLI 和 AWS 工具包直接调用 Lambda 函数。您还可以配置其他 AWS 服务以调用您的函数，或者可以配置 Lambda 以从流或队列中读取并调用您的函数。

调用函数时，您可以选择同步或异步调用。使用[同步调用 \(p. 81\)](#)时，您将等待函数处理该事件并返回响应。使用[异步调用 \(p. 83\)](#)时，Lambda 会将事件排队等待处理并立即返回响应。对于异步调用，Lambda 可以处理重试并将调用记录发送到[目标 \(p. 85\)](#)。

要使用您的函数自动处理数据，请添加一个或多个触发器。触发器是 Lambda 资源或另一个服务中的资源，可以配置它来调用您的函数以响应生命周期事件、外部请求或计划。您的函数可具有多个触发器。每个触发器充当单独调用函数的客户端。Lambda 传递给函数的每个事件仅具有来自一个客户端或触发器的数据。

要处理流或队列中的项，您可以创建[事件源映射 \(p. 90\)](#)。事件源映射是 Lambda 中的一个资源，它从 Amazon SQS 队列、Amazon Kinesis 流或 Amazon DynamoDB 流中读取项目，并将它们批量发送到您的函数。您的函数处理的每个事件可以包含数百个或数千个项。

其他 AWS 服务和资源会直接调用您的函数。例如，您可以配置 CloudWatch Events 在计时器上调用您的函数，或者可以配置 Amazon S3 在创建对象时调用您的函数。每种服务在调用函数的方法、事件的结构以及配置方式上都有所不同。有关更多信息，请参阅[将 AWS Lambda 与其他服务结合使用 \(p. 138\)](#)。

根据调用函数的人员以及调用函数的方式，扩展行为和可能发生的错误类型会有所不同。同步调用函数时，您会在响应中收到错误并且可以重试。异步调用函数时，可使用事件源映射或配置另一个服务来调用您的函数，而重试要求以及您的函数扩展处理大量事件的方式各不相同。有关详细信息，请参阅[AWS Lambda 函数扩展 \(p. 94\)](#) 和[AWS Lambda 中的错误处理和自动重试 \(p. 98\)](#)。

主题

- [同步调用 \(p. 81\)](#)
- [异步调用 \(p. 83\)](#)
- [AWS Lambda 事件源映射 \(p. 90\)](#)
- [使用 Lambda API 监控函数的状态 \(p. 92\)](#)
- [AWS Lambda 函数扩展 \(p. 94\)](#)
- [AWS Lambda 中的错误处理和自动重试 \(p. 98\)](#)
- [使用适用于 Android 的 AWS 移动软件开发工具包 调用 Lambda 函数 \(p. 99\)](#)

同步调用

当您同步调用某个函数时，Lambda 会运行该函数并等待响应。当函数执行结束时，Lambda 会返回函数代码的响应以及其他数据，例如已执行函数的版本。要使用 AWS CLI 同步调用函数，请使用 `invoke` 命令。

```
$ aws lambda invoke --function-name my-function --payload '{ "key": "value" }'
response.json
{
    "ExecutedVersion": "$LATEST",
    "StatusCode": 200
}
```

下图显示了同步调用 Lambda 函数的客户端。Lambda 将事件直接发送给函数，并将函数的响应发回调用方。

Synchronous Invocation



`payload` 是一个包含 JSON 格式事件的字符串。AWS CLI 用来写入函数响应的文件的名称为 `response.json`。如果函数返回对象或错误，则响应是 JSON 格式的对象或错误。如果函数退出时没有错误，则响应为 `null`。

该命令的输出（显示在终端中）包含来自 Lambda 响应中的标头的信息。这包括处理事件的版本（在使用[别名 \(p. 65\)](#)时非常有用），以及 Lambda 返回的状态代码。如果 Lambda 能够运行该函数，则状态代码为 200，即使该函数返回错误也是如此。

Note

对于超时很长的函数，在等待响应的同步调用期间，客户端可能会断开连接。配置您的 HTTP 客户端、软件开发工具包、防火墙、代理或操作系统，以允许针对超时或保持活动设置保持长时间的连接。

如果 Lambda 无法运行该函数，则将在输出中显示错误。

```
$ aws lambda invoke --function-name my-function --payload value response.json
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]
```

要从命令行获取调用的日志，请使用 `--log-type` 选项。响应包含一个 `LogResult` 字段，该字段包含来自调用的多达 4 KB 的 base64 编码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

您可以使用 base64 实用程序来解码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
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: 100 ms      Memory Size: 128 MB      Max Memory Used: 73 MB
```

base64 实用工具在 Linux、macOS 和 [Ubuntu on Windows](#) 上可用。对于 macOS，命令为 `base64 -D`。

有关 `Invoke` API 的更多信息（包括参数、标头和错误的完整列表），请参阅[Invoke \(p. 482\)](#)。

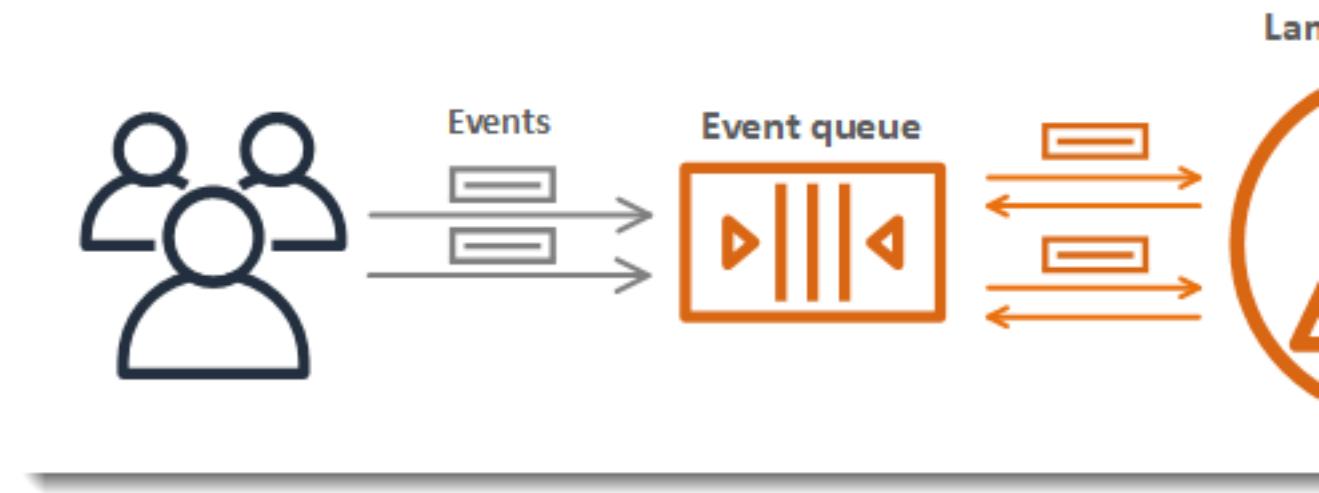
当您直接调用函数时，可以检查错误的响应并重试。在出现客户端超时、限制和服务错误时，AWS CLI 和 AWS 开发工具包也会自动重试。有关更多信息，请参阅[AWS Lambda 中的错误处理和自动重试 \(p. 98\)](#)。

异步调用

多个 AWS 服务（如 Amazon Simple Storage Service (Amazon S3) 和 Amazon Simple Notification Service (Amazon SNS)）异步调用函数来处理事件。在异步调用函数时，您不会等待函数代码的响应。您将事件交给 Lambda，Lambda 处理其余部分。您可以配置 Lambda 如何处理错误，并将调用记录发送到下游资源，以将应用程序的组件链接在一起。

下图显示了客户端异步调用函数 Lambda。Lambda 在将事件发送到函数之前对事件进行排队。

Asynchronous Invocation



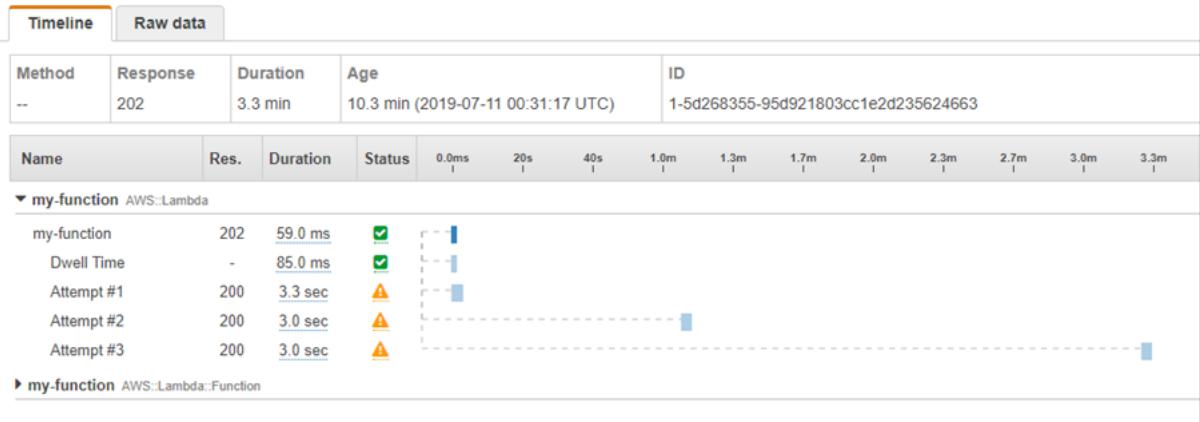
对于异步调用，Lambda 将事件放在队列中并返回成功响应，没有附加信息。一个单独的进程会从队列中读取事件并将其发送到函数。要异步调用函数，请将调用类型参数设置为 `Event`。

```
$ aws lambda invoke --function-name my-function --invocation-type Event --payload
'{ "key": "value" }' response.json
{
    "StatusCode": 202
}
```

输出文件 (`response.json`) 不包含任何信息，但运行此命令时仍会创建该文件。如果 Lambda 无法将事件添加到队列，则错误消息将显示在命令输出中。

Lambda 管理函数的异步事件队列以及在出错时重试的尝试次数。如果函数返回错误，Lambda 会尝试再运行两次，前两次尝试之间等待一分钟，第二次与第三次尝试之间等待两分钟。函数错误包括函数代码返回的错误，以及函数运行时返回的错误，例如超时。

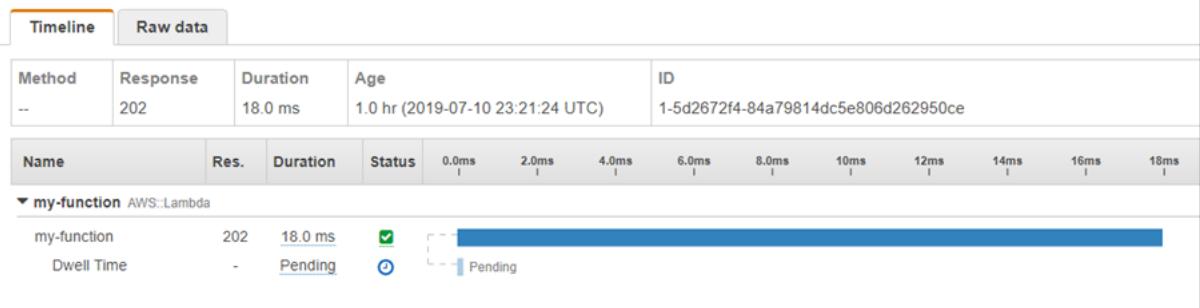
Traces > Details



如果该函数没有足够的并发性可用于处理所有事件，则其他请求将受到限制。对于限制错误 (429) 和系统错误 (500 系列)，Lambda 会将事件返回到队列并尝试再次运行该函数长达 6 小时。在第一次尝试后，重试间隔从 1 秒以指数级增加到最多 5 分钟。但是，如果队列已备份，则时间可能会更长。Lambda 还会降低从队列中读取事件的速率。

以下示例显示事件已成功添加到队列中，但由于限制，一小时后仍处于等待状态。

Traces > Details



即使您的函数没有返回错误，它也可能多次从 Lambda 接收相同的事件，因为队列本身具有最终一致性。如果函数无法跟上传入事件，则也可能从队列中删除事件而不将其发送到函数。确保您的函数代码正常处理重复事件，并且您有足够的并发可用于处理所有调用。

备份队列时，新事件可能会在 Lambda 有机会将它们发送到您的函数之前过期。当事件过期或所有处理尝试都失败时，Lambda 将放弃该事件。您可为函数配置错误处理 (p. 85) 以减少 Lambda 执行的重试次数，或者更快地放弃未处理的事件。

您还可以配置 Lambda 以将调用记录发送到另一个服务。Lambda 支持以下异步调用目标 (p. 85)。

- Amazon SQS – SQS 队列。
- Amazon SNS – SNS 主题。
- AWS Lambda – Lambda 函数。
- Amazon EventBridge – EventBridge 事件总线。

调用记录包含有关 JSON 格式的请求和响应的详细信息。您可为成功处理的事件以及处理尝试失败的事件配置单独的目标。或者，您可以将 SQS 队列或 SNS 主题配置为丢弃事件的死信队列 (p. 88)。对于死信队列，Lambda 只发送事件的内容，不包含有关响应的详细信息。

小节目录

- [配置异步调用的错误处理 \(p. 85\)](#)
- [配置异步调用目标 \(p. 85\)](#)
- [异步调用配置 API \(p. 87\)](#)
- [AWS Lambda 函数死信队列 \(p. 88\)](#)

配置异步调用的错误处理

使用 Lambda 控制台配置有关函数、版本或别名的错误处理设置。

配置错误处理

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Asynchronous invocation (异步调用) 下，选择 Edit (编辑)。
4. 配置以下设置。
 - Maximum age of event (事件的最长期限) – Lambda 在异步事件队列中保留事件的最长时间（最长 6 小时）。
 - Retry attempts (重试次数) – 函数返回错误时 Lambda 重试的次数（0 到 2 之间）。
5. 选择 Save。

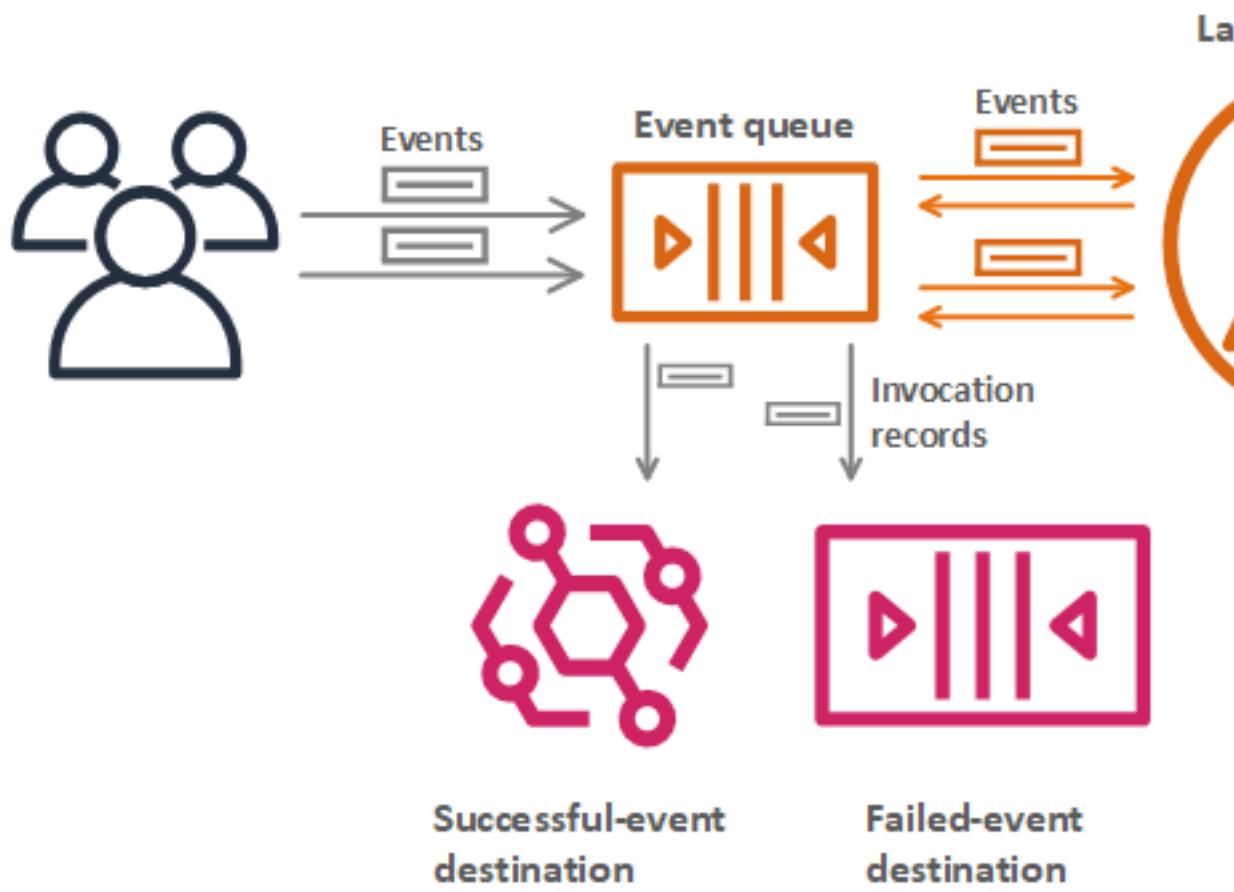
当调用事件超过最长期限或所有重试均失败时，Lambda 会丢弃该事件。要保留已丢弃事件的副本，请配置失败事件目标。

配置异步调用目标

要将异步调用记录发送到另一个服务，请向函数添加目标。您可为处理失败的事件和处理成功的事件配置单独的目标。与错误处理设置一样，您可以在函数、版本或别名上配置目标。

以下示例显示正在处理异步调用的函数。如果函数返回了成功响应或退出而未引发错误，则 Lambda 会将调用的记录发送到 EventBridge 事件总线。当事件的所有处理尝试失败时，Lambda 会将调用记录发送到 Amazon SQS 队列。

Destinations for Asynchronous Invocation



要将事件发送到目标，您的函数需要其他权限。添加具有函数[执行角色 \(p. 30\)](#)所需权限的策略。每项目标服务均需要不同的权限，如下所示：

- Amazon SQS – `sq:SendMessage`
- Amazon SNS – `sns:Publish`
- Lambda – `lambda:InvokeFunction`
- EventBridge – `events:PutEvents`

在 Lambda 控制台的函数设计器中向函数添加目标。

为异步调用记录配置目标

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Designer (设计器) 下，选择 Add destination (添加目标)。
4. 对于 Source (源)，选择 Asynchronous invocation (异步调用)。

5. 对于 Condition (条件) , 请从以下选项中选择 :
 - On failure (失败时) – 当事件的所有处理尝试均失败或超过最长期限发送记录。
 - On success (成功时) – 函数成功处理异步调用时发送记录。
6. 对于 Destination type (目标类型) , 请选择接收调用记录的资源类型。
7. 对于 Destination (目标) , 请选择一个资源。
8. 选择 Save。

当调用与条件匹配时 , Lambda 会向目标发送包含调用详细信息的 JSON 文档。以下示例显示了由于函数错误而导致三次处理尝试失败的事件的调用记录。

Example 调用记录

```
{  
    "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"  
    }  
}
```

调用记录包含有关事件、响应和记录发送原因的详细信息。

异步调用配置 API

要使用 AWS CLI 或 AWS 开发工具包管理异步调用设置 , 请使用以下 API 操作。

- [PutFunctionEventInvokeConfig](#)
- [GetFunctionEventInvokeConfig](#)
- [UpdateFunctionEventInvokeConfig](#)
- [ListFunctionEventInvokeConfigs](#)
- [DeleteFunctionEventInvokeConfig](#)

要使用 AWS CLI 配置异步调用 , 请使用 put-function-event-invoke-config 命令。以下示例配置一个最长事件期限为 1 小时且无重试的函数。

```
$ aws lambda put-function-event-invoke-config --function-name error \
--maximum-event-age-in-seconds 3600 --maximum-retry-attempts 0
{
    "LastModified": 1573686021.479,
    "FunctionArn": "arn:aws:lambda:us-east-2:123456789012:function:error:$LATEST",
    "MaximumRetryAttempts": 0,
    "MaximumEventAgeInSeconds": 3600,
    "DestinationConfig": {
        "OnSuccess": {},
        "OnFailure": {}
    }
}
```

`put-function-event-invoke-config` 命令覆盖函数、版本或别名上的任何现有配置。要配置某个选项而不重置其他选项，请使用 `update-function-event-invoke-config`。以下示例配置 Lambda，以便在无法处理事件时将记录发送到名为 `destination` 的 SQS 队列。

```
$ aws lambda update-function-event-invoke-config --function-name error \
--destination-config '{"OnFailure":{"Destination": "arn:aws:sqs:us-
east-2:123456789012:destination"}}'
{
    "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"
        }
    }
}
```

AWS Lambda 函数死信队列

作为[失败时的目标](#)(p. 85)的替代，您可以使用死信队列配置函数，以保存丢弃的事件供进一步处理。死信队列的作用与失败时的目标相同，在某个事件的所有处理尝试都失败或者已过期而未处理时使用。但是，死信队列是函数特定于版本的配置的一部分，因此在发布某个版本时锁定。失败时的目标还支持其他目标，并在调用记录中包含有关函数响应的详细信息。

如果您没有队列或主题，请创建一个队列或主题。选择与您的使用案例匹配的目标类型。

- [Amazon SQS 队列](#) – 队列会保存失败的事件，直到检索这些事件为止。您可以手动检索事件，也可以配置 Lambda 以从队列中读取(p. 255)并调用函数。

在[Amazon SQS 控制台](#)中，创建一个队列。

- [Amazon SNS 主题](#) – 主题将失败的事件中继到一个或多个目标。您可以配置主题以将事件发送到电子邮件地址、Lambda 函数或 HTTP 终端节点。

在[Amazon SNS 控制台](#)中，创建一个主题。

要将事件发送到队列或主题，您的函数需要其他权限。添加具有函数[执行角色](#)(p. 30)所需权限的策略。

- Amazon SQS – `sqs:SendMessage`
- Amazon SNS – `sns:Publish`

如果已使用客户管理的密钥加密目标队列或主题，则执行角色也必须是密钥的[基于资源的策略](#)中的用户。

创建目标并更新函数的执行角色后，将死信队列添加到函数中。您可以配置多个函数，以便将事件发送到同一目标。

配置死信队列

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Asynchronous invocation (异步调用) 下，选择 Edit (编辑)。
4. 将 DLQ resource (DLQ 资源) 设置为 Amazon SQS 或 Amazon SNS。
5. 选择目标队列或主题。
6. 选择 Save。

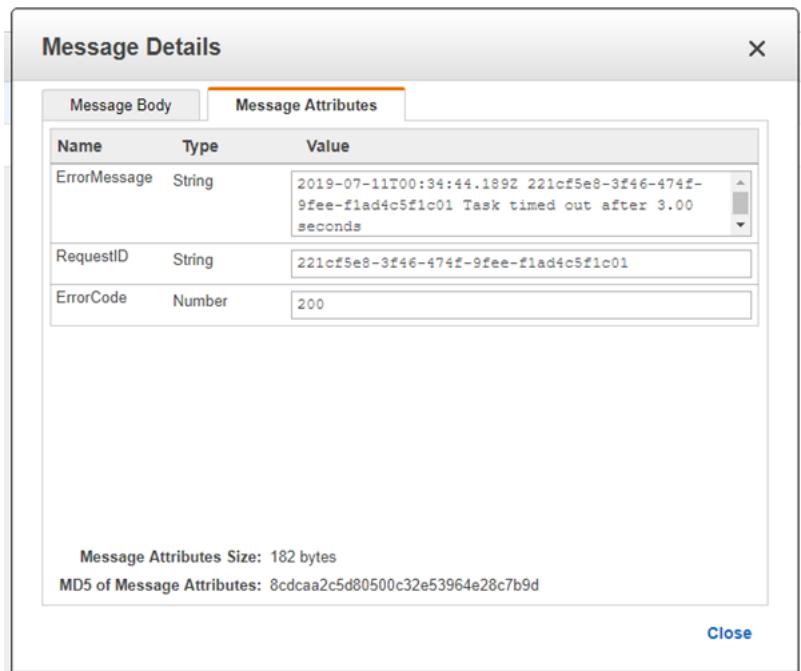
要通过 AWS CLI 配置死信队列，请使用 `update-function-configuration` 命令。

```
$ aws lambda update-function-configuration --function-name my-function \
--dead-letter-config TargetArn=arn:aws:sns:us-east-2:123456789012:my-topic
```

Lambda 按原样将事件发送到死信队列，并在属性中包含其他信息。您可以使用此信息来标识函数返回的错误，或者将事件与日志或 AWS X-Ray 跟踪相关联。

死信队列消息属性

- RequestID (字符串) – 调用请求的 ID。请求 ID 显示在函数日志中。您还可以使用 X-Ray 开发工具包，在跟踪中的属性上记录请求 ID。然后，可以在 X-Ray 控制台中按请求 ID 搜索跟踪。有关示例，请参阅[错误处理程序示例 \(p. 128\)](#)。
- ErrorCode (数字) – HTTP 状态代码。
- ErrorMessage (字符串) – 错误消息的第一个 1 KB 文本块。



如果 Lambda 无法向死信队列发送消息，则会删除该事件并发出 [DeadLetterErrors \(p. 368\)](#) 指标。之所以发生这种情况，可能是由于缺少权限，或者消息的总大小超过目标队列或主题的限制。例如，如果正文接近

256 KB 的 Amazon SNS 通知触发导致错误的功能，则 Amazon SNS 添加的附加事件数据与 Lambda 添加的属性相结合，可能会导致消息超过死信队列中允许的最大大小。

如果您正在使用 Amazon SQS 作为事件源，请在 Amazon SQS 队列本身而不是 Lambda 函数上配置死信队列。有关更多信息，请参阅[将 AWS Lambda 与 Amazon SQS 结合使用 \(p. 255\)](#)。

AWS Lambda 事件源映射

事件源映射是一个从事件源读取并调用 Lambda 函数的 AWS Lambda 资源。您可以使用事件源映射来处理未直接调用 Lambda 函数的服务中的流或队列中的项。Lambda 为以下服务提供事件源映射。

Lambda 从其读取事件的服务

- [Amazon Kinesis \(p. 212\)](#)
- [Amazon DynamoDB \(p. 184\)](#)
- [Amazon Simple Queue Service \(p. 255\)](#)

事件源映射使用函数[执行角色 \(p. 30\)](#)中的权限来读取和管理事件源中的项。权限、事件结构、设置和轮询行为因事件源而异。有关更多信息，请参阅用作事件源的服务的链接主题。

要使用 AWS CLI 或 AWS 开发工具包管理事件源映射，请使用以下 API 操作：

- [CreateEventSourceMapping \(p. 415\)](#)
- [ListEventSourceMappings \(p. 492\)](#)
- [GetEventSourceMapping \(p. 451\)](#)
- [UpdateEventSourceMapping \(p. 547\)](#)
- [DeleteEventSourceMapping \(p. 432\)](#)

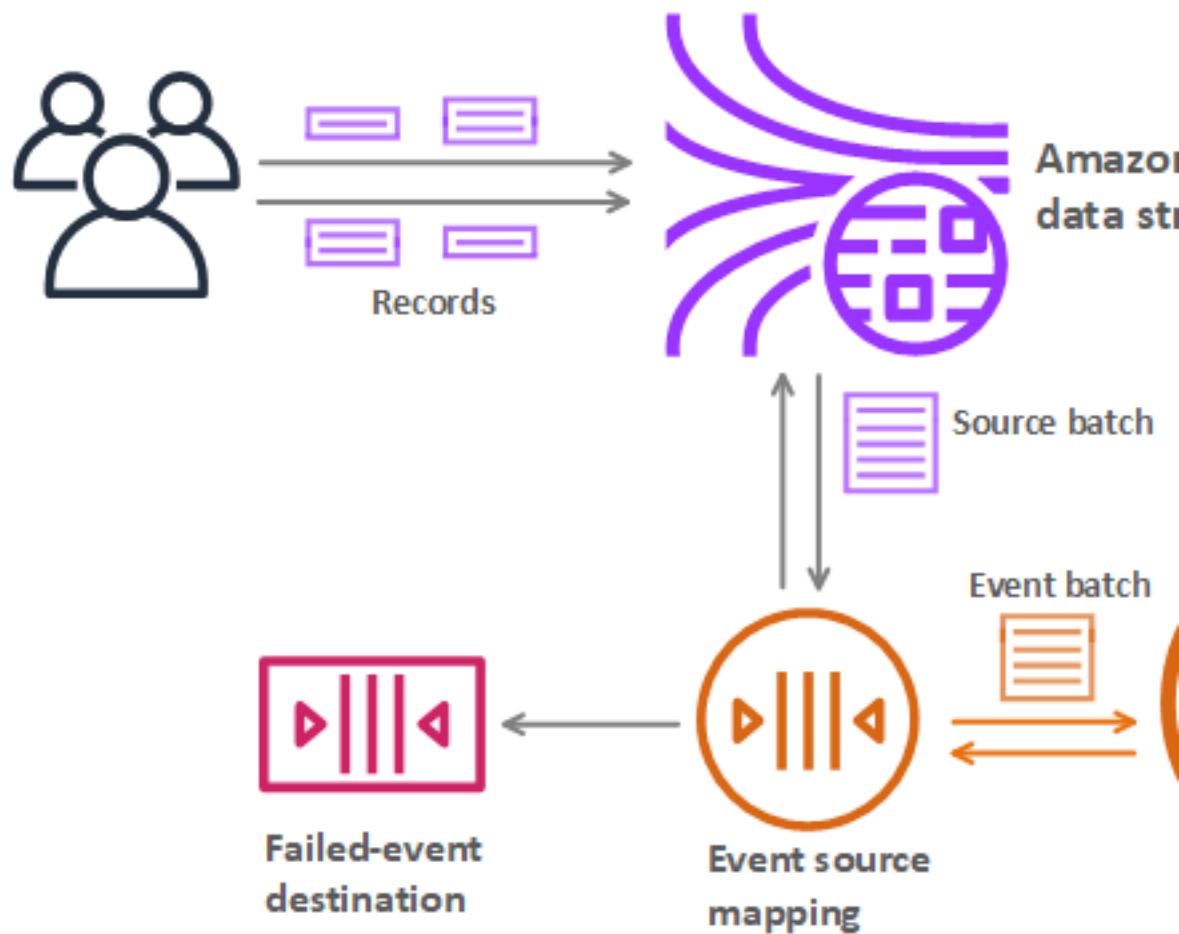
以下示例使用 AWS CLI 将名为 my-function 的函数映射到由 Amazon 资源名称 (ARN) 指定的 DynamoDB 流 (批处理大小为 500)。

```
$ 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  
{  
    "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  
}
```

事件源映射从流或队列中批量读取项目。它们包括您的函数收到的事件中的多个项目。您可以配置事件源映射发送到函数的批次大小，最大值因服务而异。如果没有足够可用的项，或者批次太大而无法在一个事件中发送并且必须拆分，则事件中的项数可能小于批次大小。

以下示例显示了从 Kinesis 流读取的事件源映射。如果一批事件的所有处理尝试失败，则事件源映射将有关该批次的详细信息发送到 SQS 队列。

Event Source Mapping with Kinesis Stream



事件批次是 Lambda 发送到函数的事件。它是一批记录或消息，编译自事件源映射从流或队列读取的项目。批次大小和其他设置仅适用于该事件批次。

对于流，事件源映射为流中的每个分片创建迭代器，并按顺序处理每个分片中的项。您可以将事件源映射配置为只读取流中显示的新项，或者从较旧的项开始。已处理的项不会从流中删除，并且可由其他函数或使用者处理。

默认情况下，如果您的函数返回错误，则重新处理整个批次，直到函数成功，或直到批次中的项目到期。为确保按顺序处理，将暂停对受影响的分片的处理，直到解决错误为止。您可以将事件源映射配置为放弃旧事件、限制重试次数或并行处理多个批次。如果并行处理多个批次，仍然保证每个分区键按顺序处理，但同一分片中的多个分区键会同时处理。

您还可以将事件源映射配置为在放弃某个事件批次时向其他服务发送调用记录。Lambda 支持以下事件源映射的目标 (p. 85)。

- Amazon SQS – SQS 队列。
- Amazon SNS – SNS 主题。

调用记录包含 JSON 格式的失败事件批次的详细信息。

以下示例显示了 Kinesis 流的调用记录。

Example 调用记录

```
{  
    "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"  
    }  
}
```

Lambda 还支持 [FIFO \(先进先出\) 队列的有序处理 \(p. 255\)](#)，可向上扩展到活动消息组的数量。对于标准队列，项目不一定按顺序处理。Lambda 向上扩展以尽可能快地处理标准队列。出现错误时，失败的批次作为单个项返回到队列，并且可在与原始批次不同的分组中处理。有时，即使没有发生任何函数错误，事件源映射也可能会从队列中接收相同的项两次。Lambda 在成功处理后会删除队列中的项。如果无法处理项目，您可以配置源队列以将项目发送到死信队列。

有关直接调用 Lambda 函数的服务的信息，请参阅 [将 AWS Lambda 与其他服务结合使用 \(p. 138\)](#)。

使用 Lambda API 监控函数的状态

创建或更新函数时，Lambda 预配置使其能够运行的计算和网络资源。在大多数情况下，这个过程非常快，并且您的函数已经准备好立即调用或修改。

如果您将函数配置为连接到 Virtual Private Cloud (VPC)，则该过程可能需要更长时间。当您首次将函数连接到 VPC 时，Lambda 会预配置网络接口，这需要大约一分钟。为了传达函数的当前状态，Lambda 在[函数配置 \(p. 590\)](#)文档中包括由几个 Lambda API 操作返回的附加字段。

创建函数时，函数最初处于 Pending 状态。当函数准备好调用时，状态从 Pending 更改为 Active。当状态为 Pending 时，在函数上运行的调用和其他 API 操作返回错误。如果围绕创建和更新函数构建自动化，请等待函数变为活动状态，然后再执行对该函数运行的其他操作。

您可以使用 Lambda API 获取函数状态的相关信息。状态信息包括在多个 API 操作返回的[FunctionConfiguration \(p. 590\)](#) 文档中。要使用 AWS CLI 查看函数的状态，请使用 `get-function-configuration` 命令。

```
$ aws lambda get-function-configuration --function-name my-function
{
    "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",
    ...
}
```

当状态不是 Active 时，StateReason 和 StateReasonCode 包含有关状态的附加信息。在函数创建处于挂起状态时，以下操作会失败：

- [Invoke \(p. 482\)](#)
- [UpdateFunctionCode \(p. 553\)](#)
- [UpdateFunctionConfiguration \(p. 560\)](#)
- [PublishVersion \(p. 518\)](#)

当您更新函数的配置时，更新会触发预配置资源的异步操作。在此过程中，您可以调用该函数，但对该函数的其他操作会失败。在更新过程中发生的调用将针对之前的配置运行。函数的状态是 Active，但其 LastUpdateStatus 是 InProgress。

Example 函数配置 – 连接到 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",
    ...
}
```

正在进行异步更新时，以下操作会失败：

- [UpdateFunctionCode \(p. 553\)](#)
- [UpdateFunctionConfiguration \(p. 560\)](#)
- [PublishVersion \(p. 518\)](#)

正在进行更新时，其他操作（包括调用）会起作用。

例如，当您将函数连接到 Virtual Private Cloud (VPC) 时，Lambda 会为每个子网预配置弹性网络接口。此过程会使您的函数处于挂起状态一分钟左右。Lambda 还会回收未使用的网络接口，使您的函数处于 Inactive 状态。当函数处于不活动状态时，调用会导致它在网络访问恢复时进入 Pending 状态。在操作处于待处理状态时触发还原的调用以及进一步的调用失败，并引发 ResourceNotReadyException。

如果在还原函数的网络接口时 Lambda 遇到错误，则该函数将恢复到 Inactive 状态。下一次调用可能会触发另一次尝试。对于某些配置错误，Lambda 等待至少 5 分钟，然后尝试创建另一个网络接口。这些错误具有以下 LastUpdateStatusReasonCode 值：

- InsufficientRolePermission – 角色不存在或缺少权限。
- SubnetOutOfIPAddresses – 子网中的所有 IP 地址正在使用中。

有关状态如何与 VPC 连接结合使用的更多信息，请参阅[配置 Lambda 函数以访问 VPC 中的资源 \(p. 72\)](#)。

AWS Lambda 函数扩展

第一次调用函数时，AWS Lambda 会创建函数的实例并运行其处理程序方法来处理事件。当函数返回响应时，它会保持活动并等待处理其他事件。如果在处理第一个事件时再次调用该函数，则 Lambda 初始化另一个实例，并且该函数同时处理这两个事件。随着更多事件的进入，Lambda 将它们路由到可用实例并根据需要创建新实例。当请求数量减少时，Lambda 会停止未使用的实例以释放其他函数的扩展容量。

您函数的并发 数量是在给定时间为请求提供服务的实例数。对于最初的流量突增，您的函数在一个区域中的累积并发数量可以达到 500 到 3000 之间的初始级别，该级别因区域而异。

突增并发限制

- 3000 – 美国西部（俄勒冈），美国东部（弗吉尼亚北部），欧洲（爱尔兰）
- 1000 – 亚太区域（东京），欧洲（法兰克福）
- 500 – 其他区域

在初始突增之后，您函数的并发可按每分钟增加 500 个实例的速度扩展。这将一直持续到有足够的实例来服务于所有请求，或者直到达到并发限制。当请求进入的速度超过函数可扩展的速度，或者当函数处于最大并发时，其他请求会因限制错误而失败（状态代码为 429）。

以下示例显示了处理流量峰值的函数。随着调用数呈指数增加，函数会向上扩展。它为无法路由到可用实例的任何请求初始化一个新实例。当达到突增并发限制时，函数开始线性缩放。如果没有足够的并发数量来满足所有请求，其他请求将会受到限制并且应重试。

Function Scaling with Concurrency Limit

Concurrency
limit

Burst limit

图例

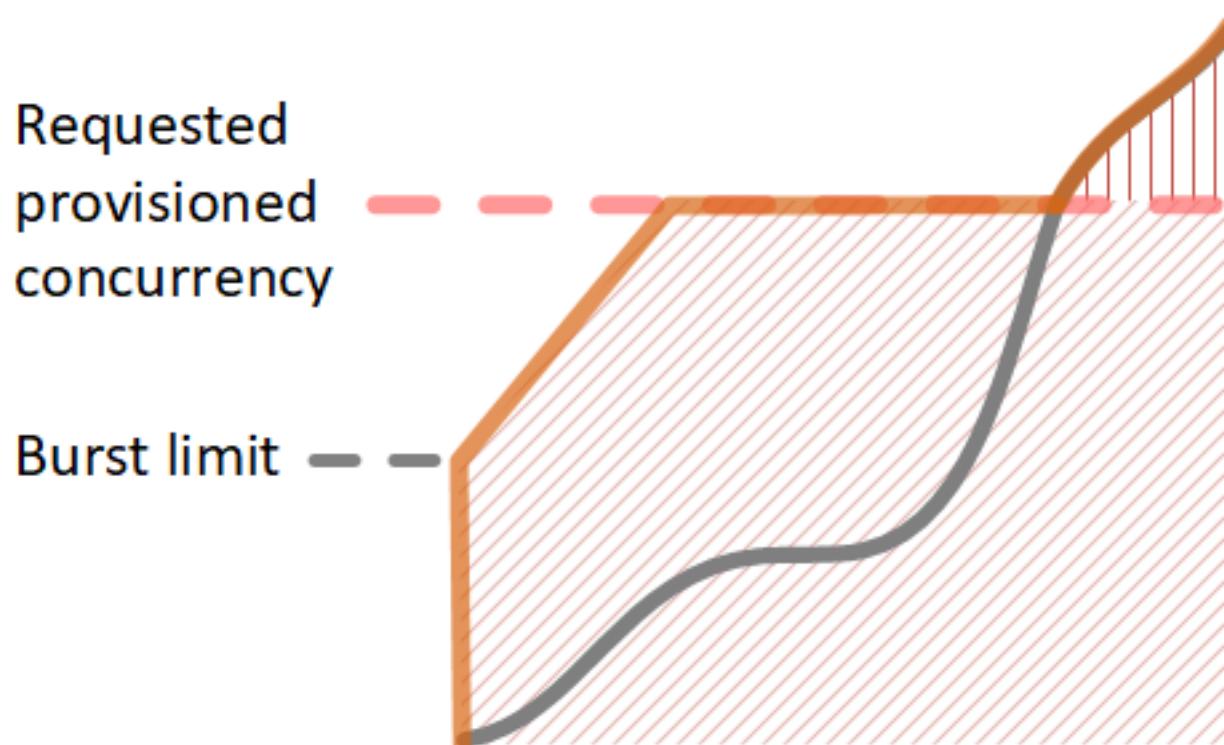
- 函数实例
- 打开请求
- 可以限制

函数将继续扩展，直到达到函数区域对账户的并发限制。该函数能够满足需求，请求减少，并且在空闲一段时间后，函数的未使用实例会停止。未使用的实例在等待请求时会被冻结，且不会产生任何费用。

区域并发限制从 1,000 开始。您可以通过在[支持中心控制台](#)中提交请求，来提高限制。要在每个函数的基础上分配容量，可以使用[预留并发 \(p. 54\)](#)配置函数。预留并发创建一个只能由其函数使用的池，并且还防止其函数使用非预留并发。

当函数扩展时，每个实例处理的第一个请求都会受到它加载和初始化代码所需时间的影响。如果[初始化代码 \(p. 17\)](#)需要很长时间，则对平均延迟和百分位数延迟的影响可能很大。要使函数能够在延迟不发生波动的情况下进行扩展，请使用[预配置并发 \(p. 54\)](#)。以下示例显示了一个具有处理流量峰值的预配置并发的函数。

Function Scaling with Provisioned Concurrency



图例

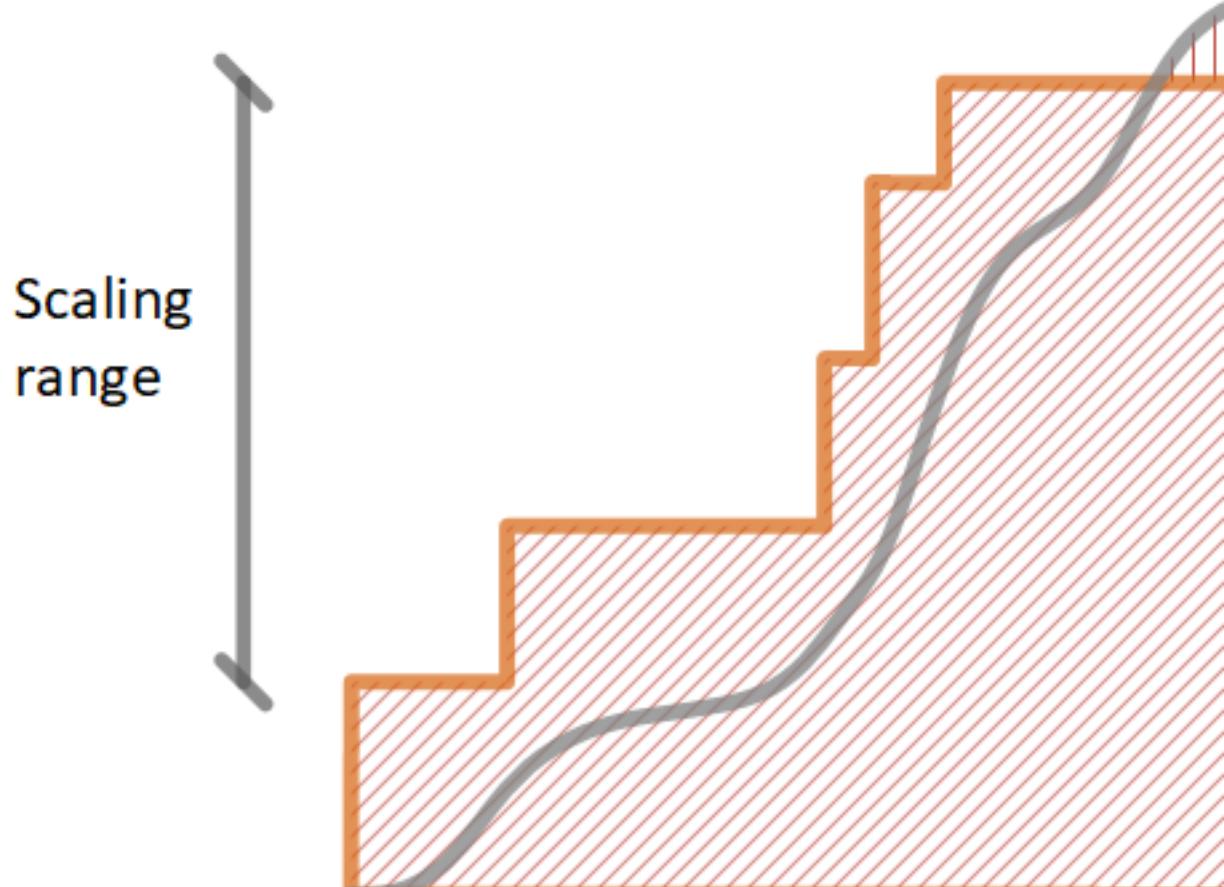
- 函数实例
- 打开请求
- 预配置并发
- 标准并发

分配预配置并发时，函数的缩放行为与标准并发的突增行为相同。在分配之后，预配置并发将以非常低的延迟处理传入请求。当所有预配置并发都在使用中时，函数会正常扩展以处理任何其他请求。

Application Auto Scaling 通过为预配置的并发提供自动扩展来更进一步。使用 Application Auto Scaling，您可以创建目标跟踪扩展策略，该策略可根据 Lambda 发出的利用率指标自动调整预配置的并发级别。[使用 Application Auto Scaling API \(p. 60\)](#) 将别名注册为可扩展目标并创建扩展策略。

在以下示例中，函数根据利用率在预配置并发的最小数量和最大数量之间缩放。当打开请求的数量增加时，Application Auto Scaling 将大幅增加预配置并发，直到达到配置的最大数量。该函数继续扩展标准并发，直到利用率开始下降。当利用率持续较低时，Application Auto Scaling 将定期小幅减少预配置并发。

Autoscaling with Provisioned Concurrency



图例

- 函数实例
- 打开请求
- 预配置并发

- ||||||| 标准并发

当您通过使用事件源映射或其他 AWS 服务异步调用您的函数时，扩展行为会有所不同。例如，从流中读取的事件源映射会受到流中分片数量的限制。事件源未使用的扩展容量可供其他客户端和事件源使用。有关更多信息，请参阅以下主题。

- 异步调用 (p. 83)
- AWS Lambda 事件源映射 (p. 90)
- AWS Lambda 中的错误处理和自动重试 (p. 98)
- 将 AWS Lambda 与其他服务结合使用 (p. 138)

您可以通过使用以下指标监控帐户中的并发级别。

并发指标

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

有关更多信息，请参阅[使用 AWS Lambda 函数指标 \(p. 368\)](#)。

AWS Lambda 中的错误处理和自动重试

调用函数时，可能会发生两种类型的错误。在您的函数收到调用请求之前拒绝调用请求时会发生调用错误。当函数的代码或[运行时 \(p. 108\)](#)返回错误时，会发生函数错误。根据错误类型、调用类型以及调用该函数的客户端或服务，重试行为和管理错误的策略会有所不同。

请求、调用方或账户的问题可能会导致调用错误。调用错误包括响应中的错误类型和状态代码，它们指示错误的原因。

常见调用错误

- 请求 – 请求事件太大或 JSON 无效；函数不存在；或者参数值是错误的类型。
- 调用方 – 用户或服务无权调用该函数。
- 账户 – 已在运行最大数量的函数实例，或者请求过快。

在出现客户端超时、限制错误 (429) 以及不是由错误请求 (500 系列) 引起的其他错误时，诸如 AWS CLI 和 AWS 开发工具包之类的客户端会重试。有关调用错误的完整列表，请参阅[Invoke \(p. 482\)](#)。

当您的函数代码或代码使用的运行时返回错误时，会发生函数错误。

常见函数错误

- 函数 – 您的函数的代码抛出异常或返回错误对象。
- 运行时 – 运行时终止了您的函数（因为它耗尽了时间）；检测到语法错误；或者无法将响应对象编组为 JSON。函数退出，并返回错误代码。

与调用错误不同，函数错误不会导致 Lambda 返回 400 系列或 500 系列状态代码。如果函数返回错误，则 Lambda 通过包含名为 `X-Amz-Function-Error` 的标头和带有错误消息和其他详细信息的 JSON 格式响应来指示此错误。有关每个语言中的函数错误的示例，请参阅以下主题。

- [Node.js 中的 AWS Lambda 函数错误 \(p. 274\)](#)
- [Python 中的 AWS Lambda 函数错误 \(p. 287\)](#)
- [Ruby 中的 AWS Lambda 函数错误 \(p. 299\)](#)
- [Java 中的 AWS Lambda 函数错误 \(p. 319\)](#)
- [Go 中的 AWS Lambda 函数错误 \(p. 339\)](#)
- [C# 中的 AWS Lambda 函数错误 \(p. 355\)](#)
- [PowerShell 中的 AWS Lambda 函数错误 \(p. 366\)](#)

直接调用函数时，您负责确定处理错误的策略。您可以重试、将事件发送到队列以进行调试，或者忽略该错误。您的函数代码可能已完全、部分或根本不运行。如要重试，请确保您的函数代码可以多次处理相同的事件，而不会导致重复的事务或其他不必要的副作用。

间接调用函数时，您需要了解调用者的重试行为以及请求在此过程中遇到的任何服务。这包括以下场景。

- 异步调用 – Lambda 会针对函数错误重试两次。如果该函数没有足够的容量来处理所有传入请求，则事件可能会在队列中等待数小时或数天才能发送到该函数。您可以在函数上配置死信队列以捕获未成功处理的事件。有关更多信息，请参阅 [异步调用 \(p. 83\)](#)。
- 事件源映射 – 从流中读取的事件源映射将重试整个批次的项。重复错误会阻止受影响的分片的处理，直到错误得到解决或项失效为止。要检测停顿的分片，您可以监控[迭代器期限 \(p. 368\)](#)指标。

对于从队列中读取的事件源映射，您可以通过在源队列上配置可见性超时和重新驱动策略来确定失败事件的重试次数以及目标之间的时间长度。有关更多信息，请参阅 [AWS Lambda 事件源映射 \(p. 90\)](#) 以及 [将 AWS Lambda 与其他服务结合使用 \(p. 138\)](#)下方特定于服务的主题。

- AWS 服务 – AWS 服务可能[同步 \(p. 81\)](#)或异步调用您的函数。对于同步调用，服务将决定是否重试。API 网关 和 Elastic Load Balancing 等服务（处理来自上游用户或客户端的代理请求）也可以选择将错误响应中继回请求者。

对于异步调用，行为与您异步调用函数时的行为相同。有关更多信息，请参阅 [将 AWS Lambda 与其他服务结合使用 \(p. 138\)](#) 以及调用服务文档下的特定于服务的主题。

- 其他账户和客户端 – 当您向其他账户授予访问权限时，可以使用[基于资源的策略 \(p. 33\)](#)来限制可配置为调用您的函数的服务或资源。为了保护您的函数不发生过载情况，请考虑使用 [Amazon API Gateway \(p. 141\)](#) 在您的函数前面放置一个 API 层。

为了帮助您处理 Lambda 应用程序中的错误，Lambda 集成了 Amazon CloudWatch 和 AWS X-Ray 等服务。您可以结合使用日志、指标、警报和跟踪来快速检测和识别函数代码，API 或支持您的应用程序的其他资源中的问题。有关更多信息，请参阅[对基于 Lambda 的应用程序进行监控和问题排查 \(p. 367\)](#)。

有关使用 CloudWatch Logs 订阅，X-Ray 跟踪和 Lambda 函数来检测和处理错误的示例应用程序，请参阅 [AWS Lambda 错误处理器示例应用程序 \(p. 128\)](#)。

使用 适用于 Android 的 AWS 移动软件开发工具包 调用 Lambda 函数

您可以从移动应用程序调用 Lambda 函数。将业务逻辑放在函数中，以将其开发生命周期与前端客户端分开，从而降低移动应用程序的开发和维护复杂性。通过 适用于 Android 的 移动软件开发工具包，您可以[使用 Amazon Cognito 对用户进行身份验证并授权请求 \(p. 100\)](#)。

从移动应用程序调用函数时，您可以选择事件结构、[调用类型 \(p. 81\)](#)和权限模型。您可以使用[别名 \(p. 65\)](#)来实现函数代码的无缝更新，但这样做会使函数与应用程序紧密耦合。在添加更多函数时，您可以创建 API 层来将函数代码与前端客户端解耦，从而提高性能。

要为您的移动和 Web 应用程序创建功能齐全的 Web API，请使用 Amazon API Gateway。通过 API 网关，您可以为所有函数添加自定义授权方、限制请求和缓存结果。有关更多信息，请参阅[配合使用 AWS Lambda 和 Amazon API Gateway \(p. 141\)](#)。

主题

- 教程：将 AWS Lambda 与适用于 Android 的移动软件开发工具包结合使用 (p. 100)
- 示例函数代码 (p. 105)

教程：将 AWS Lambda 与适用于 Android 的移动软件开发工具包结合使用

在本教程中，您将创建一个简单的 Android 移动应用程序，该应用程序使用 Amazon Cognito 获取凭证和调用 Lambda 函数。



该移动应用程序从 Amazon Cognito 身份池检索 AWS 凭证，并使用它们通过包含请求数据的事件调用 Lambda 函数。该函数处理请求并向前端返回响应。

先决条件

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command  
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以安装[Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

创建执行角色

创建[执行角色 \(p. 30\)](#)，向您的函数授予访问 AWS 资源的权限。

创建执行角色

1. 打开 IAM 控制台中的“[角色](#)”页面。
2. 选择 Create role (创建角色)。
3. 创建具有以下属性的角色。
 - 可信任的实体 – AWS Lambda。
 - 权限 – AWSLambdaBasicExecutionRole。
 - 角色名称 (角色名称) – **lambda-android-role**。

AWSLambdaBasicExecutionRole 策略具有函数将日志写入 CloudWatch Logs 所需的权限。

创建函数

以下示例使用数据来生成字符串响应。

Note

有关使用其他语言的示例代码，请参阅[示例函数代码 \(p. 105\)](#)。

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);
}
```

创建函数

1. 将示例代码复制到名为 `index.js` 的文件中。
2. 创建部署程序包。

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

3. 使用 `create-function` 命令创建 Lambda 函数。

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

测试 Lambda 函数。

使用示例事件数据手动调用函数。

测试 Lambda 函数 (AWS CLI)

1. 将下面的示例事件 JSON 保存到文件 `input.txt` 中。

```
{ "firstName": "first-name", "lastName": "last-name" }
```

2. 执行下面的 invoke 命令：

```
$ aws lambda invoke --function-name AndroidBackendLambdaFunction \
--payload file://file-path/input.txt outfile.txt
```

创建 Amazon Cognito 身份池

在这一部分，您将创建一个 Amazon Cognito 身份池。该身份池有两个 IAM 角色。您将更新未经身份验证的用户的 IAM 角色，并授予执行 AndroidBackendLambdaFunction Lambda 函数的权限。

有关 IAM 角色的详细信息，请参阅 IAM 用户指南 中的 [IAM 角色](#)。有关 Amazon Cognito 服务的更多信息，请参阅 [Amazon Cognito](#) 产品详细信息页面。

创建身份池

1. 打开 [Amazon Cognito 控制台](#)。
2. 新建一个名为 JavaFunctionAndroidEventHandlerPool 的身份池。在按照过程创建身份池前，请注意以下几点：
 - 您创建的身份池必须允许访问未经身份验证的身份，因为我们的示例移动应用程序不要求用户登录。因此，请确保选择 Enable access to unauthenticated identities (启用未经验证的身份的访问权限) 选项。
 - 将以下语句添加到与未经身份验证的身份关联的权限策略。

```
{
    "Effect": "Allow",
    "Action": [
        "lambda:InvokeFunction"
    ],
    "Resource": [
        "arn:aws:lambda:us-
east-1:123456789012:function:AndroidBackendLambdaFunction"
    ]
}
```

得到的策略应如下所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "mobileanalytics:PutEvents",
                "cognito-sync:*"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:invokefunction"
            ],
            "Resource": [
                "arn:aws:lambda:us-
east-1:123456789012:function:AndroidBackendLambdaFunction"
            ]
        }
    ]
}
```

```
    "Resource": [
        "arn:aws:lambda:us-east-1:account-id:function:AndroidBackendLambdaFunction"
    ]
}
```

有关如何创建身份池的说明，请登录 [Amazon Cognito 控制台](#)，然后根据 New Identity Pool (新建身份池) 向导的指示进行操作。

- 记下身份池 ID。您将在创建于下一节的移动应用程序中指定此 ID。应用程序在向 Amazon Cognito 请求临时安全凭证时将使用此 ID。

创建 Android 应用程序

创建一个简单的 Android 移动应用程序来生成事件并通过以参数形式传递事件数据调用 Lambda 函数。

以下操作不受已用 Android studio 进行了验证。

- 使用下面的配置创建名为 `AndroidEventGenerator` 的新 Android 项目：
 - 选择 Phone and Tablet 平台。
 - 选择 Blank Activity。
- 在 `build.gradle (Module:app)` 文件的 `dependencies` 部分中添加以下内容：

```
compile 'com.amazonaws:aws-android-sdk-core:2.2.+'
compile 'com.amazonaws:aws-android-sdk-lambda:2.2.+'
```

- 构建项目，以便根据需要下载必需的依赖项。
- 在 `AndroidManifest.xml` 中，添加以下权限，以使您的应用程序能够连接 Internet。可以将它们添加在 `</manifest>` 结束标签之前。

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

- 在 `MainActivity` 中，添加以下导入：

```
import com.amazonaws.mobileconnectors.lambdainvoker.*;
import com.amazonaws.auth.CognitoCachingCredentialsProvider;
import com.amazonaws.regions.Regions;
```

- 在 `package` 部分中，添加以下两个类 (`RequestClass` 和 `ResponseClass`)。请注意，此 POJO 与您在上一节中的 Lambda 函数中创建的 POJO 相同。

• `RequestClass`。此类的实例将充当包含名字和姓氏的事件数据的 POJO (Plain Old Java Object)。如果您使用的是在上一节中创建的 Lambda 函数的 Java 示例，则此 POJO 与您在 Lambda 函数代码中创建的 POJO 相同。

```
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. 在相同的程序包中，创建名为 MyInterface 的接口，以用于调用 AndroidBackendLambdaFunction Lambda 函数。

```
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);

}
```

代码中的 `@LambdaFunction` 注释将该特定的客户端方法映射到同名的 Lambda 函数。

8. 为使应用程序保持简单，我们将在 `onCreate()` 事件处理程序中添加调用 Lambda 函数的代码。在 `MainActivity` 中，向 `onCreate()` 代码末尾处添加以下代码。

```
// 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. 运行代码并按以下方式验证它：

- `Toast.makeText()` 显示返回的响应。
- 验证 CloudWatch Logs 能够显示 Lambda 函数创建的日志。它应显示事件数据（名字和姓氏）。您也可以在 AWS Lambda 控制台中对此进行验证。

示例函数代码

示例代码具有以下语言。

主题

- [Node.js \(p. 106\)](#)
- [Java \(p. 106\)](#)

Node.js

以下示例使用数据来生成字符串响应。

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);
}
```

压缩示例代码以创建部署程序包。有关说明，请参阅[Node.js 中的 AWS Lambda 部署程序包 \(p. 268\)](#)。

Java

以下示例使用数据来生成字符串响应。

在该代码中，`handler (myHandler)` 使用 `RequestClass` 和 `ResponseClass` 类型进行输入和输出。该代码为这些类型提供了实现。

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);  
}  
}
```

附属物

- [aws-lambda-java-core](#)

使用 Lambda 库依赖项构建代码以创建部署程序包。有关说明，请参阅[Java 中的 AWS Lambda 部署程序包 \(p. 302\)](#)。

AWS Lambda 运行时

AWS Lambda 通过使用运行时支持多种语言。您可以在创建函数时选择运行时，并且可以通过更新函数的配置来更改运行时。底层执行环境提供了您可通过函数代码访问的额外的库和环境变量 ([p. 49](#))。

Amazon Linux

- 映像 – [amzn-ami-hvm-2018.03.0.20181129-x86_64-gp2](#)
- Linux 内核 – 4.14.154-99.181.amzn1.x86_64

Amazon Linux 2

- 映像 – 自定义
- Linux 内核 – 4.14.138-99.102.amzn2.x86_64

在调用您的函数时，Lambda 会尝试重新使用上一个调用中的执行环境（如果可用）。这将节省执行环境的准备时间，并可让您将数据库连接和临时文件等资源保存在[执行上下文 \(p. 109\)](#)中，从而使函数无需在每次运行时都创建这些资源。

运行时可以支持一种语言的单个版本、一种语言的多个版本或多种语言。特定于语言或框架版本的运行时会在版本的使用寿命结束时被[弃用 \(p. 110\)](#)。

Node.js 运行时

名称	标识符	适用于 JavaScript 的 AWS 开发工具包	操作系统	
Node.js 12	nodejs12.x	2.631.0	Amazon Linux 2	
Node.js 10	nodejs10.x	2.631.0	Amazon Linux 2	

Python 运行时

名称	标识符	适用于 Python 的 AWS 开发工具包	操作系统
Python 3.8	python3.8	boto3-1.12.22 botocore-1.15.22	Amazon Linux 2
Python 3.7	python3.7	boto3-1.12.22 botocore-1.15.22	Amazon Linux
Python 3.6	python3.6	boto3-1.12.22 botocore-1.15.22	Amazon Linux
Python 2.7	python2.7	boto3-1.12.22 botocore-1.15.22	Amazon Linux

Ruby 运行时

名称	标识符	适用于 Ruby 的 AWS 开发工具包	操作系统
Ruby 2.7	ruby2.7	3.0.1	Amazon Linux 2
Ruby 2.5	ruby2.5	3.0.1	Amazon Linux

Java 运行时

名称	标识符	JDK	操作系统
Java 11	java11	amazon-corretto-11	Amazon Linux 2
Java 8	java8	java-1.8.0-openjdk	Amazon Linux

Go 运行时

名称	标识符	操作系统
Go 1.x	go1.x	Amazon Linux

.NET 运行时

名称	标识符	操作系统
.NET Core 3.1	dotnetcore3.1	Amazon Linux 2
.NET Core 2.1	dotnetcore2.1	Amazon Linux

要在 Lambda 中使用其他语言，您可以实施[自定义运行时 \(p. 111\)](#)。Lambda 执行环境提供[运行时接口 \(p. 112\)](#)来获取调用事件并发送响应。您可以将自定义运行时与函数代码一起部署，也可以在[层 \(p. 68\)](#)中部署它。

自定义运行时

名称	标识符	操作系统
自定义运行时	provided	Amazon Linux

主题

- [AWS Lambda 执行上下文 \(p. 109\)](#)
- [运行时支持策略 \(p. 110\)](#)
- [自定义 AWS Lambda 运行时 \(p. 111\)](#)
- [AWS Lambda 运行时接口 \(p. 112\)](#)
- [教程 – 发布自定义运行时 \(p. 115\)](#)

AWS Lambda 执行上下文

当 AWS Lambda 执行您的 Lambda 函数时，它会预置和管理要运行您的 Lambda 函数所需的资源。在您创建 Lambda 函数时，您指定配置信息，例如您希望允许 Lambda 函数使用的内存量和最长执行时间。调用

Lambda 函数时，AWS Lambda 根据您提供的配置设置启动执行上下文。执行上下文是一个临时运行时环境，它会初始化您的 Lambda 函数代码的任何外部依赖项，例如数据库连接或 HTTP 终端节点。这将为后续调用提供更好的性能，因为无需“冷启动”或初始化这些外部依赖项，如下所述。

设置执行上下文需要一些时间并进行必需的“引导”，这会在每次调用 Lambda 函数时增加一些延迟。通常，在首次调用 Lambda 函数或者在更新该函数后会看到此延迟，因为 AWS Lambda 尝试重用执行上下文进行 Lambda 函数的后续调用。

在执行 Lambda 函数之后，AWS Lambda 会保持执行上下文一段时间，预期用于另一次 Lambda 函数调用。其效果是，服务在 Lambda 函数完成后冻结执行上下文，如果再次调用 Lambda 函数时 AWS Lambda 选择重用上下文，则解冻上下文供重用。此执行上下文重用方法的意义在于：

- 在该函数的处理程序方法的外部声明的对象保持已初始化的状态，再次调用函数时提供额外的优化功能。例如，如果您的 Lambda 函数建立数据库连接，而不是重新建立连接，则在后续调用中使用原始连接。建议您在代码中添加逻辑，以便在创建新连接之前检查是否存在连接。
- 每个执行上下文在 /tmp 目录中提供 512 MB 大小的额外磁盘空间。冻结执行上下文时，目录内容会保留，提供可用于多次调用的暂时性缓存。您可以添加额外的代码来检查缓存中是否有您存储的数据。有关部署限制的更多信息，请参阅 [AWS Lambda 限制 \(p. 27\)](#)。
- 如果 AWS Lambda 选择重用执行上下文，而您的 Lambda 函数在结束时有未完成的后台进程或者回调，则函数将继续它们。您应确保代码中的任何后台进程或回调在代码退出前已完成。

在编写 Lambda 函数代码时，不会假定 AWS Lambda 自动为后续函数调用重用执行上下文。其他因素可能指示需要 AWS Lambda 以创建新的执行上下文，这可能导致意外结果（例如，数据库连接失败）。

运行时支持策略

[AWS Lambda 运行时 \(p. 108\)](#)是围绕不断进行维护和安全更新的操作系统、编程语言和软件库的组合构建的。当安全更新不再支持某个运行时组件时，Lambda 将弃用该运行时。

弃用过程分两个阶段进行。在第一阶段，您无法再创建使用已弃用的运行时的函数。您可以继续更新使用已弃用的运行时的现有函数达至少 30 天。该期限过后，将永久禁用函数创建和更新。但是，该函数可继续用于处理调用事件。

Note

Python 2.7 于 2020 年 1 月 1 日使用寿命到期。但是，Python 2.7 运行时仍然受支持，并且目前不计划弃用。有关详细信息，请参阅 [在 AWS Lambda 上继续支持 Python 2.7](#)。

在大多数情况下，语言版本或操作系统的寿命结束日期是事先已知的。如果您的函数运行在 60 天内将弃用的运行时上，Lambda 将通过电子邮件通知您将函数迁移到支持的运行时，以做好准备。在某些情况下，例如需要不向后兼容的更新的安全问题，或者不支持长期支持 (LTS) 计划的软件，可能无法提前通知。

语言和框架支持政策

- Node.js – github.com
- Python – devguide.python.org
- Ruby – www.ruby-lang.org
- Java – www.oracle.com 和 aws.amazon.com/corretto
- Go – golang.org
- .NET Core – dotnet.microsoft.com

弃用某运行时后，Lambda 可能随时通过禁用调用来彻底停用该运行时。我们不向已弃用的运行时提供安全更新或技术支持。在停用运行时之前，Lambda 会向受影响的客户发送额外的通知。目前，我们没有停用任何运行时的计划。

自定义 AWS Lambda 运行时

您可以采用任何编程语言实施 AWS Lambda 运行时。运行时是一个程序，当调用一个 Lambda 函数时，它将运行该函数的处理程序方法。您可以采用可执行文件（名为 `bootstrap`）的形式将运行时包含在函数的部署程序包中。

运行时负责运行函数的设置代码、从环境变量读取处理程序名称以及从 Lambda 运行时 API 读取调用事件。运行时会将事件数据传递到函数处理程序，并将来自处理程序的响应发布回 Lambda。

您的自定义运行时在标准 Lambda 执行环境 (p. 108) 中运行。它可以是 Shell 脚本（采用包含在 Amazon Linux 中的语言），也可以是在 Amazon Linux 中编译的二进制可执行文件。

要开始使用自定义运行时，请参阅教程 – 发布自定义运行时 (p. 115)。您还可在 GitHub 的 [awslabs/aws-lambda-cpp](#) 上了解以 C++ 实现的自定义运行时。

主题

- [使用自定义运行时 \(p. 111\)](#)
- [构建自定义运行时 \(p. 111\)](#)

使用自定义运行时

要使用自定义运行时，请将函数的运行时设置为 `provided`。该运行时可包含在函数的部署程序包中，或包含在层 (p. 68) 中。

Example function.zip

```
.\n### bootstrap\n### function.sh
```

如果部署程序包中存在一个名为 `bootstrap` 的文件，Lambda 将执行该文件。如果不存在，Lambda 将在函数的层中寻找运行时。如果引导文件未找到或不是可执行文件，您的函数在调用后将返回错误。

构建自定义运行时

自定义运行时的入口点是一个名为 `bootstrap` 的可执行文件。引导文件可以是运行时，也可以调用创建运行时的另一个文件。以下示例使用捆绑版本的 Node.js 在名为 `runtime.js` 的单独文件中执行 JavaScript 运行时。

Example 引导

```
#!/bin/sh\ncd $LAMBDA_TASK_ROOT\n./node-v11.1.0-linux-x64/bin/node runtime.js
```

您的运行时代码负责完成一些初始化任务。然后，它将在一个循环中处理调用事件，直到它被终止。初始化任务将对 [函数的每个实例 \(p. 109\)](#) 运行一次以准备用于处理调用的环境。

初始化任务

- 检索设置 – 读取环境变量以获取有关函数和环境的详细信息。
 - `_HANDLER` – 处理程序的位置（来自函数的配置）。标准格式为 `file.method`，其中 `file` 是没有表达式的文件的名称，`method` 是在文件中定义的方法或函数的名称。
 - `LAMBDA_TASK_ROOT` – 包含函数代码的目录。
 - `AWS_LAMBDA_RUNTIME_API` – 运行时 API 的主机和端口。

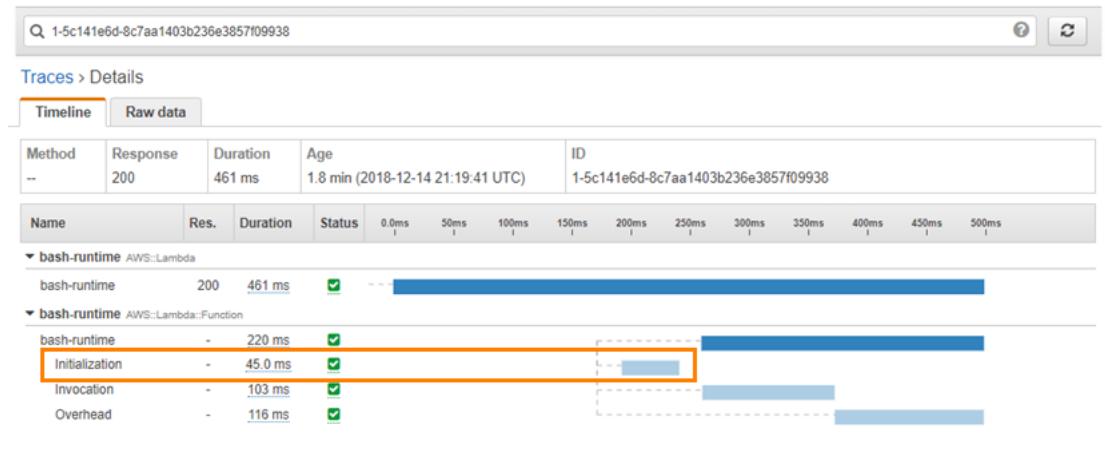
有关可用变量的完整列表，请参阅[运行时环境变量 \(p. 51\)](#)。

- 初始化函数 – 加载处理程序文件并运行它包含的任何全局或静态代码。函数应该创建静态资源（如开发工具包客户端和数据库连接）一次，然后将它们重复用于多个调用。
- 处理错误 – 如果出现错误，请调用[初始化错误 \(p. 114\)](#) API 并立即退出。

计入收费的执行时间和超时的初始化计数。当执行触发您的函数的新实例的初始化时，您可以在日志和[AWS X-Ray 跟踪 \(p. 371\)](#)中看到初始化时间。

Example 日志

```
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
```



当它运行时，运行时将使用[Lambda 运行时界面 \(p. 112\)](#)来管理传入事件和报告错误。完成初始化任务后，运行时将在一个循环中处理传入事件。在运行时代码中，按顺序执行下面的步骤。

处理任务

- 获取事件 – 调用[下一个调用 \(p. 113\)](#) API 来获取下一个事件。响应正文包含事件数据。响应标头包含请求 ID 和其他信息。
- 传播跟踪标头 – 从 API 响应中的 Lambda-Runtime-Trace-Id 标头获取 X-Ray 跟踪标头。使用与要使用的 X-Ray 开发工具包相同的值设置_X_AMZN_TRACE_ID 环境变量。
- 创建上下文对象 – 创建一个对象，该对象包含来自环境变量的上下文信息和 API 响应中的标头。
- 调用函数处理程序 – 将事件和上下文对象传递到处理程序。
- 处理响应 – 调用[调用响应 \(p. 114\)](#) API 以发布来自处理程序的响应。
- 处理错误 – 如果出现错误，请调用[调用错误 \(p. 114\)](#) API。
- 清理 – 释放未使用的资源，将数据发送到其他服务，或在获取下一个事件之前执行其他任务。

您可以在函数的部署程序包中包含运行时，也可以在函数层中单独分配运行时。有关示例演练的信息，请参阅[教程 – 发布自定义运行时 \(p. 115\)](#)。

AWS Lambda 运行时接口

AWS Lambda 提供了用于[自定义运行时 \(p. 111\)](#)的 HTTP API 来接收来自 Lambda 的调用事件并在 Lambda 执行环境 (p. 108) 中发送回响应数据。

运行时 API 版本 2018-06-01 的 OpenAPI 规范在此处提供：[runtime-api.zip](#)

运行时将从 AWS_LAMBDA_RUNTIME_API 环境变量获取终端节点，添加 API 版本，并使用以下资源路径与 API 进行交互。

Example 请求

```
curl "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invocation/next"
```

资源

- [下一个调用 \(p. 113\)](#)
- [调用响应 \(p. 114\)](#)
- [调用错误 \(p. 114\)](#)
- [初始化错误 \(p. 114\)](#)

下一个调用

路径 – /runtime/invocation/next

方法 – GET

检索调用事件。响应主体包含来自调用的有效负载，该负载是包含来自函数触发器的事件数据的 JSON 文档。响应标头包含有关调用的额外数据。

响应标头

- Lambda-Runtime-Aws-Request-Id – 请求 ID，用于标识触发了函数调用的请求。
例如：8476a536-e9f4-11e8-9739-2dfe598c3fc0。
- Lambda-Runtime-Deadline-Ms – 函数超时的日期（Unix 时间形式，以毫秒为单位）。
例如：1542409706888。
- Lambda-Runtime-Invoked-Function-Arn – Lambda 函数的 ARN、版本或在调用中指定的别名。
例如：arn:aws:lambda:us-east-2:123456789012:function:custom-runtime。
- Lambda-Runtime-Trace-Id – [AWS X-Ray 跟踪标头](#)。
例如：Root=1-5bef4de7-ad49b0e87f6ef6c87fc2e700;Parent=9a9197af755a6419;Sampled=1。
- Lambda-Runtime-Client-Context – 对于来自 AWS 移动开发工具包的调用，为有关客户端应用程序和设备的数据。
- Lambda-Runtime-Cognito-Identity – 对于来自 AWS 移动开发工具包的调用，为有关 Amazon Cognito 身份提供商的数据。

调用 /runtime/invocation/next 获取调用事件，并将其传递给函数处理程序进行处理。不要对 GET 调用设置超时。在 Lambda 引导运行时和运行时返回事件之间，运行时进程可能会冻结几秒钟。

请求 ID 用于跟踪 Lambda 中的调用。使用它可在您发送响应时指定调用。

跟踪标头包含跟踪 ID、父 ID 和采样决策。如果对请求进行采样，则由 Lambda 或某个上游服务对请求进行采样。运行时应设置具有标头的值的 _X_AMZN_TRACE_ID。X-Ray 开发工具包将读取此项以获取 ID 并确定是否要跟踪请求。

调用响应

路径 – /runtime/invocation/*AwsRequestId*/response

方法 – POST

将调用响应发送到 Lambda。在运行时调用函数处理程序后，它会将来自函数的响应发布到调用响应路径。对于同步调用，Lambda 随后会将响应发送回客户端。

Example 成功请求

```
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"
```

调用错误

路径 – /runtime/invocation/*AwsRequestId*/error

方法 – POST

如果函数返回了错误，则运行时会将错误格式化到一个 JSON 文档中，并将它发布到调用错误路径。

Example 请求正文

```
{
  "errorMessage" : "Error parsing event data.",
  "errorType" : "InvalidEventDataException"
}
```

Example 错误请求

```
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"
```

初始化错误

路径 – /runtime/init/error

方法 – POST

如果运行时在初始化期间遇到错误，它会将错误消息发布到初始化错误路径。

Example 初始化错误请求

```
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"
```

教程 – 发布自定义运行时

在本教程中，您将创建一个具有自定义运行时的 Lambda 函数。首先，您在函数的部署程序包中包含运行时。然后，您将其迁移到一个您独立于函数管理的层。最后，您通过更新运行时层的基于资源的权限策略来将运行时层与全球共享。

先决条件

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command  
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以[安装 Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

您需要 IAM 角色才能创建 Lambda 函数。该角色需要将日志发送到 CloudWatch Logs 并访问您的函数使用的 AWS 服务的权限。如果您还没有函数开发的角色，请立即创建一个。

创建执行角色

1. 打开 IAM 控制台中的“[角色](#)”页面。
2. 选择 Create role (创建角色)。
3. 创建具有以下属性的角色。
 - 可信任的实体 – Lambda。
 - 权限 – AWSLambdaBasicExecutionRole。
 - 角色名称 (角色名称) – **lambda-role**。

AWSLambdaBasicExecutionRole 策略具有函数将日志写入 CloudWatch Logs 所需的权限。

创建函数

使用自定义运行时创建 Lambda 函数。此示例包含两个文件 - 一个运行时 bootstrap 文件和一个函数处理程序。两个文件都在 Bash 中实施。

运行时将从部署程序包加载函数脚本。它使用两个变量来查找脚本。LAMBDA_TASK_ROOT 向它告知在何处提取程序包，_HANDLER 包含脚本的名称。

Example 引导

```
#!/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 -L "$HEADERS" -X GET "http://${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)

    # Execute the handler function from the script
    RESPONSE=$(($(_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
```

加载脚本后，运行时将在一个循环中处理事件。它使用运行时 API 从 Lambda 检索调用事件，将事件传递到处理程序，并将响应发布回 Lambda。为了获取请求 ID，运行时会将来自 API 响应的标头保存到临时文件，并从该文件读取 Lambda-Runtime-Aws-Request-Id 标头。

Note

运行时还具有其他职责（包括错误处理），并向处理程序提供上下文信息。有关详细信息，请参阅[构建自定义运行时 \(p. 111\)](#)。

该脚本将定义一个处理程序函数，该函数将选取事件数据，将该数据记录到 `stderr`，然后返回它。

Example function.sh

```
function handler () {
    EVENT_DATA=$1
    echo "$EVENT_DATA" 1>&2;
    RESPONSE="Echoing request: '$EVENT_DATA'

    echo $RESPONSE
}
```

将这两个文件都保存在名为 `runtime-tutorial` 的项目目录中。

```
runtime-tutorial
# bootstrap
# function.sh
```

使文件可执行并将其添加到 ZIP 文档。

```
runtime-tutorial$ chmod 755 function.sh bootstrap
runtime-tutorial$ zip function.zip function.sh bootstrap
adding: function.sh (deflated 24%)
adding: bootstrap (deflated 39%)
```

创建名为 `bash-runtime` 的函数。

```
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",
...
}
```

调用函数并验证响应。

```
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"}'
```

创建层

要将运行时代码与函数代码分开，请创建一个仅包含运行时的层。层可让您单独开发函数的各个依赖项，而且，通过对多个函数使用相同的层，还可以减少存储使用。

创建一个包含 `bootstrap` 文件的层存档。

```
runtime-tutorial$ zip runtime.zip bootstrap
adding: bootstrap (deflated 39%)
```

使用 `publish-layer-version` 命令创建层。

```
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
}
```

这将创建第一个版本的层。

更新函数

要将运行时层与函数一起使用，请将函数配置为使用该层，并从函数中删除运行时代码。

更新函数配置以拉入到层。

```
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
        }
    ]
    ...
}
```

这会将运行时添加到 /opt 目录中的函数。Lambda 将使用此运行时，但仅当您从函数的部署程序包中删除它时才如此。更新函数代码以仅包含处理程序脚本。

```
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
        }
    ]
    ...
}
```

调用函数以验证它是否适用于运行时层。

```
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"}'
```

更新运行时

要记录有关执行环境的信息，请更新运行时脚本以输出环境变量。

Example 引导

```
#!/bin/sh

set -euo pipefail

echo "## Environment variables:"
env
```

```
# Initialization - load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"
...
```

使用新代码创建另一个版本的层。

```
runtime-tutorial$ zip runtime.zip bootstrap
updating: bootstrap (deflated 39%)
runtime-tutorial$ aws lambda publish-layer-version --layer-name bash-runtime --zip-file
fileb://runtime.zip
```

配置函数以使用新版本的层。

```
runtime-tutorial$ aws lambda update-function-configuration --function-name bash-runtime \
--layers arn:aws:lambda:us-west-2:123456789012:layer:bash-runtime:2
```

共享层

将权限语句添加到运行时层以将其与其他账户共享。

```
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"
}
```

您可以添加多个语句，每个语句都会向单个账户、组织中的账户或所有账户授予权限。

清除

删除每个版本的层。

```
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
```

由于函数包含对版本 2 的层的引用，因此该层仍然存在于 Lambda 中。函数可继续工作，但无法再被配置为使用删除的版本。如果您随后修改了函数的层的列表，则必须指定新版本或忽略已删除的层。

使用 `delete-function` 命令删除教程函数。

```
runtime-tutorial$ aws lambda delete-function --function-name bash-runtime
```

AWS Lambda 应用程序

AWS Lambda 应用程序是 Lambda 函数、事件源以及共同执行任务的其他资源的组合。您可以使用 AWS CloudFormation 和其他工具来将您的应用程序组件收集到单个程序包中，作为一个资源进行部署和管理。应用程序可以使您的 Lambda 项目具备可移植性，并让您集成其他开发人员工具，如 AWS CodePipeline、AWS CodeBuild 和 AWS 无服务器应用程序模型命令行界面 (SAM CLI)。

[AWS Serverless Application Repository](#) 提供了一组 Lambda 应用程序，只需单击几次即可在您的账户中部署。该存储库包含随时可用的应用程序和示例，您可以用作自己项目的起点。您也可以提交自己的项目以包括在其中。

[AWS CloudFormation](#) 可以让您创建一个模板来定义您的应用程序资源，并让您将应用程序作为堆栈进行管理。您可以在您的应用程序堆栈中更安全地添加或修改资源。如果更新的任何部分失败，AWS CloudFormation 会自动回滚到之前的配置。利用 AWS CloudFormation 参数，您可以从同一模板为应用程序创建多个环境。[AWS SAM \(p. 26\)](#) 使用侧重于 Lambda 应用程序开发的简化语法来扩展 AWS CloudFormation。

[AWS CLI \(p. 26\)](#) 和 [SAM CLI \(p. 26\)](#) 是用于管理 Lambda 应用程序堆栈的命令行工具。除了通过 AWS CloudFormation API 管理应用程序堆栈的命令，AWS CLI 还支持高级命令，可简化诸如上传部署程序包和更新模板等任务。AWS SAM CLI 提供了额外的功能，包括验证模板和本地测试。

主题

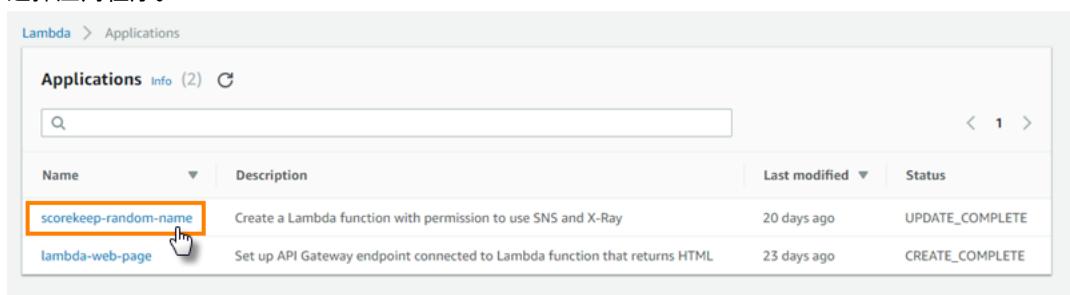
- [在 AWS Lambda 控制台中管理应用程序 \(p. 120\)](#)
- [在 Lambda 控制台中创建具有持续交付功能的应用程序 \(p. 122\)](#)
- [AWS Lambda 错误处理器示例应用程序 \(p. 128\)](#)
- [Lambda 函数的滚动部署 \(p. 132\)](#)
- [常见 Lambda 应用程序类型和使用案例 \(p. 133\)](#)
- [使用 AWS Lambda 函数的最佳实践 \(p. 135\)](#)

在 AWS Lambda 控制台中管理应用程序

AWS Lambda 控制台可帮助您监控和管理您的 [Lambda 应用程序 \(p. 120\)](#)。Applications (应用程序) 菜单列出了包含 Lambda 函数的 AWS CloudFormation 堆栈。该菜单包含使用 AWS CloudFormation 控制台、AWS Serverless Application Repository、AWS CLI 或 AWS SAM CLI 在 AWS CloudFormation 中启动的堆栈。

查看 Lambda 应用程序

1. 打开 Lambda 控制台的“[应用程序](#)”页面。
2. 选择应用程序。



概述显示有关应用程序的以下信息。

- AWS CloudFormation template (CFN 模板) 或 SAM template (SAM 模板) – 定义应用程序的模板。
- Resources (资源) – 应用程序模板中定义的 AWS 资源。要管理应用程序的 Lambda 函数，请从列表中选择一个函数名称。

监控应用程序

Monitoring (监控) 选项卡显示了 Amazon CloudWatch 控制面板，其中包含应用程序中的资源的聚合指标。

监控 Lambda 应用程序

1. 打开 Lambda 控制台的“[应用程序](#)”页面。
2. 选择 Monitoring。

默认情况下，Lambda 控制台会显示基本控制面板。您可以在应用程序模板中定义自定义控制面板，来自定义此页面。当您的模板包含一个或多个控制面板时，此页面会显示您的控制面板，而不是默认控制面板。您可以使用页面右上角的下拉菜单切换控制面板。

自定义监控控制面板

通过使用 [AWS::CloudWatch::Dashboard](#) 资源类型，在应用程序模板中添加一个或多个 Amazon CloudWatch 控制面板，以自定义您的应用程序监控页面。以下示例创建了包含单个小部件的控制面板，该小部件显示了名为 my-function 的函数的调用次数。

Example 函数控制面板模板

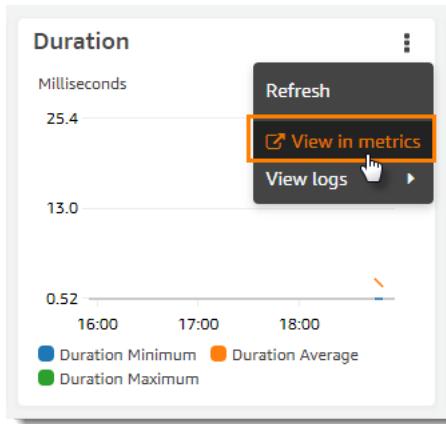
```
Resources:  
MyDashboard:  
  Type: AWS::CloudWatch::Dashboard  
  Properties:  
    DashboardName: my-dashboard  
    DashboardBody: |  
      {  
        "widgets": [  
          {  
            "type": "metric",  
            "width": 12,  
            "height": 6,  
            "properties": {  
              "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
    }
}
]
```

您可以从 CloudWatch 控制台了解默认监控控制面板中的任何小部件的定义。

查看小部件定义

1. 打开 Lambda 控制台的“[应用程序](#)”页面。
2. 选择具有标准控制面板的一个应用程序。
3. 选择 Monitoring。
4. 在任意小部件中，从下拉菜单中选择 View in metrics (在指标中查看)。



5. 选择源。

有关编写 CloudWatch 控制面板和小部件的更多信息，请参阅 Amazon CloudWatch API 参考 中的[控制面板主体结构和语法](#)。

在 Lambda 控制台中创建具有持续交付功能的应用程序

您可以使用 Lambda 控制台创建具有集成式持续交付管道的应用程序。通过持续交付，您推送到源代码控制存储库的每项更改都会触发一个自动构建和部署应用程序的管道。Lambda 控制台提供了用于带 Node.js 示例代码的通用应用程序类型的初学者项目以及用于创建支持性资源的模板。

在本教程中，您将创建以下资源。

- 应用程序 – Node.js Lambda 函数、构建规范和 AWS 无服务器应用程序模型 (AWS SAM) 模板。
- 管道 – 一个 AWS CodePipeline 管道，该管道连接其他资源以实现持续交付。
- 存储库 – AWS CodeCommit 中的 Git 存储库。当您推送更改时，管道将源代码复制到 Amazon S3 存储桶并将其传递给构建项目。
- 触发器 – 一个 Amazon CloudWatch Events 规则，该规则监视存储库的主分支并触发管道。
- 构建项目 – 从管道获取源代码并打包应用程序的 AWS CodeBuild 构建。源包括构建规范，其中包含用于安装依赖项和为部署准备应用程序模板的命令。

- 部署配置 – 管道的部署阶段定义了一组操作，这些操作从构建输出中获取经过处理的 AWS SAM 模板，并使用 AWS CloudFormation 部署新版本。
- 存储桶 – 用于部署构件存储的 Amazon Simple Storage Service (Amazon S3) 存储桶。
- 角色 – 管道的源、构建和部署阶段具有的 IAM 角色可让其管理 AWS 资源。应用程序的函数具有一个[执行角色 \(p. 30\)](#)，此角色可让其上传日志并扩展以访问其他服务。

您的应用程序和管道资源是在可自定义和扩展的 AWS CloudFormation 模板中定义的。您的应用程序存储库包含一个模板，可以修改该模板来添加 Amazon DynamoDB 表、Amazon API Gateway API 和其他应用程序资源。持续交付管道是在源代码控制之外的单独模板中定义的，并具有自己的堆栈。

管道将存储库中的单个分支映射到单个应用程序堆栈。您可以创建其他管道来为同一存储库中的其他分支添加环境。此外，您还可以向管道添加阶段，以进行测试、暂存和手动审批。有关 AWS CodePipeline 的更多信息，请参阅[什么是 AWS CodePipeline](#)。

小目录

- [先决条件 \(p. 123\)](#)
- [创建应用程序 \(p. 123\)](#)
- [调用函数 \(p. 124\)](#)
- [添加 AWS 资源 \(p. 124\)](#)
- [更新权限边界 \(p. 125\)](#)
- [更新函数代码 \(p. 126\)](#)
- [后续步骤 \(p. 127\)](#)
- [故障排除 \(p. 127\)](#)
- [清除 \(p. 128\)](#)

先决条件

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command  
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以[安装 Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

本教程将 CodeCommit 用于源代码控制。要设置本地计算机以访问和更新应用程序代码，请参阅 AWS CodeCommit 用户指南 中的[设置](#)。

创建应用程序

在 Lambda 控制台中创建应用程序。

创建应用程序

1. 打开 Lambda 控制台的[“应用程序”页面](#)。
2. 选择 Create application。

3. 选择 Author from scratch。
4. 配置应用程序设置。
 - Application name (应用程序名称) – **my-app**。
 - Application description (应用程序描述) – **my application**。
 - Runtime (运行时) – Node.js 10.x。
 - 源代码控制服务 – CodeCommit。
 - 存储库名称 – **my-app-repo**。
 - Permissions (权限) – Create roles and permissions boundary (创建角色和权限边界)。
5. 选择 Create。

Lambda 创建管道和相关资源，并将示例应用程序代码提交到 Git 存储库。在创建资源时，它们将显示在概览页面上。

调用函数

调用函数以验证它是否有效。

调用应用程序的函数

1. 打开 Lambda 控制台的“[应用程序](#)”页面。
2. 选择 my-app。
3. 在 Resources (资源) 下，选择 helloFromLambdaFunction。
4. 选择 Test。
5. 配置测试事件。
 - Event name (事件名称) – **test**
 - Body (正文) – {}
6. 选择 Create。
7. 选择 Test。

Lambda 控制台执行您的函数并显示结果。展开结果下的 Details (详细信息) 部分以查看输出和执行详细信息。

添加 AWS 资源

在创建应用程序时，Lambda 控制台会创建一个包含示例应用程序的 Git 存储库。要获取本地计算机上的应用程序代码副本，请克隆项目存储库。

克隆项目存储库

1. 打开 Lambda 控制台的“[应用程序](#)”页面。
2. 选择 my-app。
3. 选择 Code (代码)。
4. 在 Repository details (存储库详细信息) 下，复制 HTTP 或 SSH 存储库 URI，具体取决于您在[设置 \(p. 123\)](#)期间配置的身份验证模式。
5. 克隆存储库。

```
~$ git clone ssh://git-codecommit.us-east-2.amazonaws.com/v1/repos/my-app-repo
```

存储库包含应用程序模板、构建规范和代码。向应用程序模板添加 DynamoDB 表。

添加 DynamoDB 表

1. 在文本编辑器中打开 `template.yml`。
2. 添加一个表资源、一个将表名传递到函数的环境变量以及一个允许函数对其进行管理的权限策略。

Example template.yml - 资源

```
...
Resources:
  ddbTable:
    Type: AWS::Serverless::SimpleTable
    Properties:
      PrimaryKey:
        Name: id
        Type: String
      ProvisionedThroughput:
        ReadCapacityUnits: 1
        WriteCapacityUnits: 1
  helloFromLambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: src/handlers/hello-from-lambda.helloFromLambdaHandler
      Runtime: nodejs10.x
      MemorySize: 128
      Timeout: 100
      Description: This is a hello from Lambda example.
    Environment:
      Variables:
        DDB_TABLE: !Ref ddbTable
    Policies:
      - DynamoDBCrudPolicy:
          TableName: !Ref ddbTable
      - AWSLambdaBasicExecutionRole
```

3. 提交并推送更改。

```
~/my-app-repo$ git commit -am "Add DynamoDB table"
~/my-app-repo$ git push
```

在推送更改时，它会触发应用程序的管道。使用应用程序屏幕的 Deployments (部署) 选项卡在更改流经管道时进行跟踪。在部署完成后，继续执行下一步。

更新权限边界

示例应用程序将权限边界 应用于其函数的执行角色。权限边界将限制可添加到函数角色的权限。在没有边界的情况下，对项目存储库具有写访问权的用户可以修改项目模板，以向函数授予访问示例应用程序范围之外的资源和服务的权限。

要让函数使用您在上一步中添加到其执行角色的 DynamoDB 权限，您必须扩展权限边界以允许其他权限。Lambda 控制台将检测不在权限边界内的资源，并提供可用于更新资源的更新后的策略。

更新应用程序的权限边界

1. 打开 Lambda 控制台的“[应用程序](#)”页面。
2. 选择您的应用程序。
3. 在 Resources (资源) 下，选择 Edit permissions boundary (编辑权限边界)。

- 按照显示的说明操作来更新边界以允许访问新表。

有关权限边界的更多信息，请参阅[使用 AWS Lambda 应用程序的权限边界 \(p. 45\)](#)。

更新函数代码

接下来，更新函数代码以使用表。以下代码使用表来追踪函数的每个实例所处理的调用的次数。它将日志流 ID 用作唯一标识符。在更新函数时以及要处理多个并发调用时，将创建新实例。

更新函数代码

- 将名为 `index.js` 的新处理程序添加到包含以下内容的 `src/handlers` 文件夹中。

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;
}
```

- 打开应用程序模板，然后将处理程序值更改为 `src/handlers/index.handler`。

Example `template.yml`

```
...
helloFromLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: src/handlers/index.handler
    Runtime: nodejs10.x
```

- 提交并推送更改。

```
~/my-app-repo$ git add . && git commit -m "Use DynamoDB table"
~/my-app-repo$ git push
```

在部署代码更改后，调用函数几次以更新 DynamoDB 表。

查看 DynamoDB 表

1. 打开 [DynamoDB 控制台的“Tables \(表\)”页面。](#)
2. 选择以 my-app 开头的表。
3. 选择 Items (项目)。
4. 选择 Start search (开始搜索)。

后续步骤

定义应用程序资源的 AWS CloudFormation 模板使用 AWS 无服务器应用程序模型 转换来简化资源定义的语法，并自动上传部署包和其他构件。AWS SAM 还提供了一个命令行界面 (AWS SAM CLI)，该界面具有与 AWS CLI 相同的打包和部署功能，并具有特定于 Lambda 应用程序的其他功能。使用 AWS SAM CLI 可在模拟 Lambda 执行环境的 Docker 容器中本地测试您的应用程序。

- [安装 AWS SAM CLI](#)
- [测试和调试无服务器应用程序](#)

AWS Cloud9 提供了一个包含 Node.js、AWS SAM CLI 和 Docker 的在线开发环境。利用 AWS Cloud9，您可以快速开始开发并从任何计算机访问您的开发环境。有关说明，请参阅 AWS Cloud9 用户指南 中的[入门](#)。

对于本地开发，适用于集成开发环境 (IDE) 的 AWS 工具包可让您在将函数推送到存储库之前对函数进行测试和调试。

- [AWS Toolkit for JetBrains](#) – 适用于 PyCharm (Python) 和 IntelliJ (Java) IDE 的插件。
- [AWS Toolkit for Eclipse](#) – 适用于 Eclipse IDE 的插件 (多种语言)。
- [AWS Toolkit for Visual Studio Code](#) – 适用于 Visual Studio Code IDE 的插件 (多种语言)。
- [AWS Toolkit for Visual Studio](#) – 适用于 Visual Studio IDE 的插件 (多种语言)。

故障排除

在您开发应用程序时，可能会遇到以下类型的错误。

- 构建错误 – 在构建阶段出现的问题，包括编译、测试和打包错误。
- 部署错误 – 在 AWS CloudFormation 无法更新应用程序堆栈时出现的问题。其中包括权限错误、账户限制、服务问题或模板错误。
- 调用错误 – 由函数的代码或运行时返回的错误。

对于构建和部署错误，您可以在 Lambda 控制台中确定导致错误的原因。

纠正应用程序错误

1. 打开 Lambda 控制台的[“应用程序”页面。](#)
2. 选择应用程序。
3. 选择 Deployments (部署)。
4. 要查看应用程序的管道，请选择 Deployment pipeline (部署管道)。
5. 确定遇到了错误的操作。
6. 要在上下文中查看错误，请选择 Details (详细信息)。

对于在 ExecuteChaneSet 操作期间出现的部署错误，管道将链接到 AWS CloudFormation 控制台中的堆栈事件列表。搜索状态为 UPDATE_FAILED 的事件。由于 AWS CloudFormation 在出现错误后回滚，因此相

关事件在列表中位于其他几个事件的下方。如果 AWS CloudFormation 无法创建更改集，则错误将显示在 Change sets (更改集) 而非 Events (事件) 下。

导致部署和调用错误的一个常见原因是缺少一个或多个角色中的权限。管道具有用于部署的角色 (CloudFormationRole)，该角色与将用于直接更新 AWS CloudFormation 堆栈的 [用户权限 \(p. 37\)](#) 具有相同的效用。如果向应用程序添加资源或启用需要用户权限的 Lambda 功能，则将使用部署角色。您可以在应用程序概述中的 Infrastructure (基础设施) 下找到指向部署角色的链接。

如果您的函数访问其他 AWS 服务或资源，或者如果您启用要求该函数具有额外权限的功能，则将使用该函数的 [执行角色 \(p. 30\)](#)。在应用程序模板中创建的所有执行角色也受应用程序权限边界的约束。此边界要求您在向模板中的执行角色添加权限后，明确授予对 IAM 中的其他服务和资源的访问权限。

例如，要将函数连接到 [Virtual Private Cloud \(p. 72\)](#) (VPC)，您需要用户权限才能描述 VPC 资源。执行角色需要权限才能管理网络接口。这需要执行以下步骤。

1. 将所需的用户权限添加到 IAM 中的部署角色。
2. 将执行角色权限添加到 IAM 中的权限边界。
3. 将执行角色权限添加到应用程序模板中的执行角色。
4. 提交并推送以部署更新后的执行角色。

在纠正权限错误后，请在管道概述中选择 Release change (发布更改) 以重新运行构建和部署。

清除

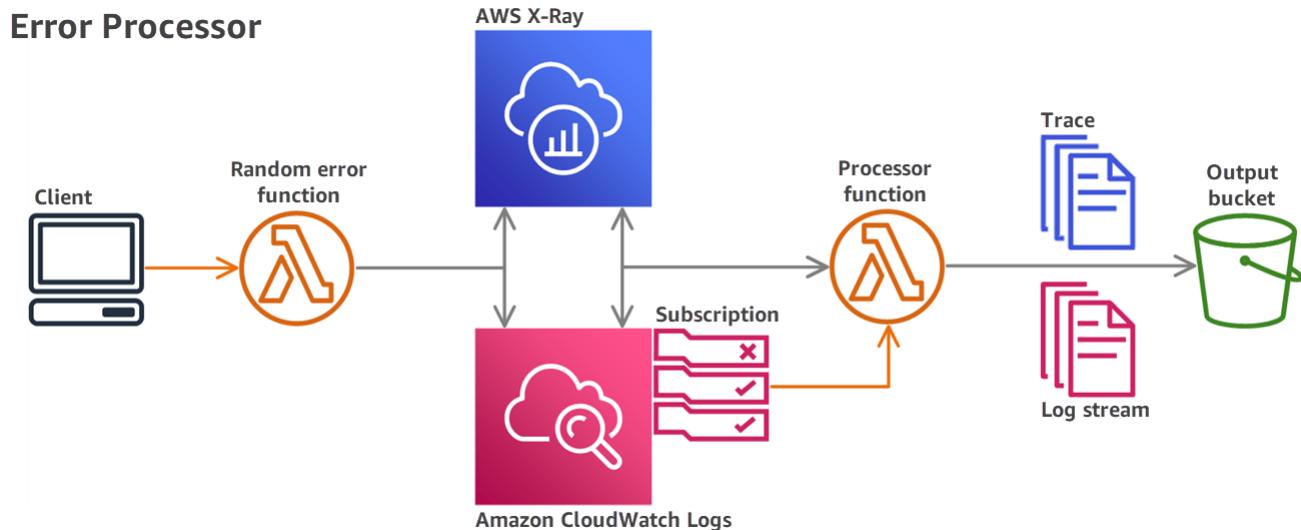
您可以继续进行修改并使用示例来开发您自己的应用程序。如果已使用示例完成此操作，请删除应用程序以避免为管道、存储库和存储付费。

删除应用程序

1. 打开 [AWS CloudFormation 控制台](#)。
2. 删除应用程序堆栈 – my-app。
3. 打开 [Amazon S3 控制台](#)。
4. 删除构件存储桶 – aws-**us-east-2-123456789012-my-app-pipe**。
5. 返回到 AWS CloudFormation 控制台并删除基础设施堆栈 – serverlessrepo-my-app-toolchain。

AWS Lambda 错误处理器示例应用程序

本错误处理器示例应用程序演示如何使用 AWS Lambda 处理来自 [Amazon CloudWatch Logs 订阅 \(p. 172\)](#) 的事件。CloudWatch Logs 允许您在日志条目与模式匹配时调用 Lambda 函数。该应用程序中的订阅监控函数的日志组来查找包含单词 ERROR 的条目。它调用处理器 Lambda 函数作为响应。处理器函数检索导致错误的请求的完整日志流和跟踪数据，并将其存储下来以备后用。



函数代码包含在以下文件中：

- 随机错误 – [random-error/index.js](#)
- 处理器 – [processor/index.js](#)

使用 AWS CLI 和 AWS CloudFormation 可在数分钟内部署本示例。请按照 README 中的说明在您的账户中下载、配置和部署它。

小目录

- 架构和事件结构 (p. 129)
- 使用 AWS X-Ray 进行检测 (p. 130)
- AWS CloudFormation 模板和其他资源 (p. 131)

架构和事件结构

本示例应用程序使用以下 AWS 服务。

- AWS Lambda – 运行函数代码，将日志发送到 CloudWatch Logs，并将跟踪数据发送到 X-Ray。
- Amazon CloudWatch Logs – 收集日志，并在日志条目与筛选器模式匹配时调用函数。
- AWS X-Ray – 收集跟踪数据，对跟踪建立索引以方便搜索，并生成服务地图。
- Amazon Simple Storage Service (Amazon S3) – 存储部署构件和应用程序输出。
- AWS CloudFormation – 创建应用程序资源并部署函数代码。

本应用程序中的 Lambda 函数将随机生成错误。当 CloudWatch Logs 在函数的日志中检测到单词 ERROR 时，即向处理器函数发送一个事件以进行处理。

Example – CloudWatch Logs 消息事件

```
{
  "awslogs": {
    "data": "H4sIAAAAAAAHWQT0/DMAzFv0vEkbLYcdJkt4qVXmCDteIAm1DbZKjS
+kdpB0Jo350MhsQFyVLsZ+unl/fJWje05asrPgbH5..."
  }
}
```

```
}
```

解码后，数据包含有关日志事件的详细信息。该函数使用这些详细信息来标识日志流，并解析日志消息以获取导致错误的请求的 ID。

Example – 解码的 CloudWatch Logs 事件数据

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

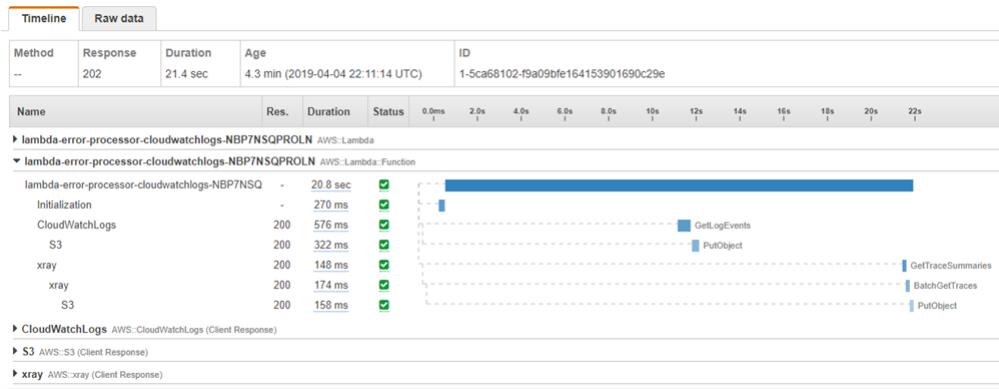
处理器函数使用来自 CloudWatch Logs 事件的信息下载导致错误的请求的完整日志流和 X-Ray 跟踪。它将这些信息存储在 Amazon S3 存储桶中。为了最终确定日志流和跟踪时间，函数在访问数据前将等待一小段时间。

使用 AWS X-Ray 进行检测

该应用程序使用 [AWS X-Ray \(p. 371\)](#) 跟踪函数调用以及函数对 AWS 服务的调用。X-Ray 使用从函数接收的跟踪数据创建服务地图，以帮助您确定错误。以下服务地图显示为部分请求生成错误的随机错误函数。它还显示调用 X-Ray、CloudWatch Logs 和 Amazon S3 的处理器函数。



两个 Node.js 函数在模板中配置为进行活动跟踪，并在代码中使用适用于 Node.js 的 AWS X-Ray 开发工具包对它们进行检测。通过活动跟踪，Lambda 标签会向传入请求添加跟踪标头，并将带有计时详细信息的跟踪发送到 X-Ray。此外，随机错误函数使用 X-Ray 开发工具包在注释中记录请求 ID 和用户信息。注释附加到跟踪，您可用它们来查找特定请求的跟踪。



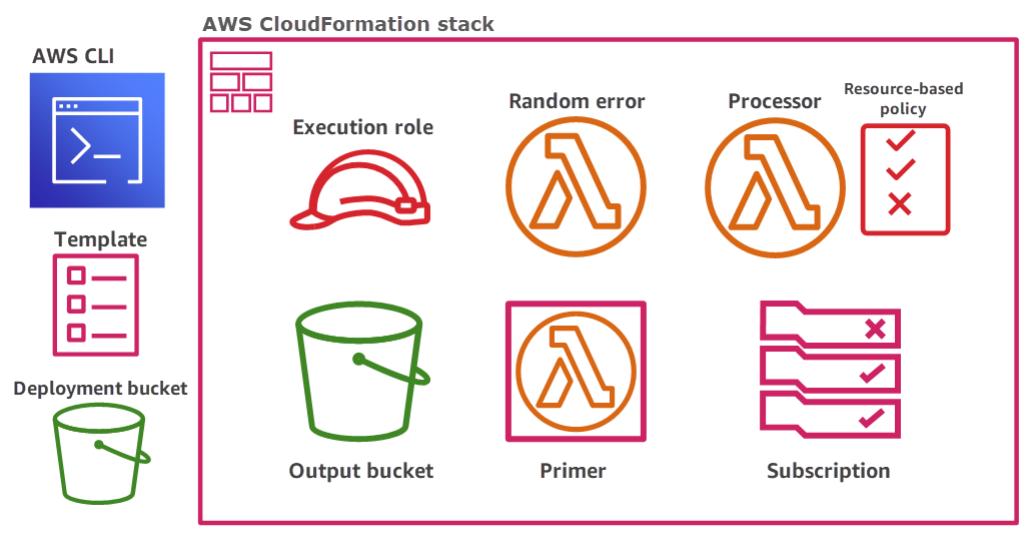
处理器函数从 CloudWatch Logs 事件获取请求 ID，并使用 AWS SDK for JavaScript 在 X-Ray 中搜索该请求。它使用 AWS 开发工具包客户端（使用 X-Ray 开发工具包进行检测）下载跟踪和日志流。然后将它们存储在输出存储桶中。X-Ray 开发工具包记录这些调用，它们在跟踪中显示为子分段。

AWS CloudFormation 模板和其他资源

本应用程序在两个 Node.js 模块（一个 AWS CloudFormation 模板和提供支持的 shell 脚本）中实现。模板创建处理器函数、随机错误函数和以下支持资源。

- 执行角色 – 一个 IAM 角色，授予函数访问其他 AWS 服务的权限。
- 初级函数 – 一个附加函数，它调用随机错误函数来创建日志组。
- 自定义资源 – 一个 AWS CloudFormation 自定义资源，在部署期间调用初级函数来确保日志组存在。
- CloudWatch Logs 订阅 – 一个日志流订阅，在记录单词 ERROR 时触发处理器函数。
- 基于资源的策略 – 关于处理器函数的权限声明，以允许 CloudWatch Logs 调用该函数。
- Amazon S3 存储桶 – 处理器函数输出的存储位置。

在 GitHub 上查看 [template.yml](#) 模板。



为了解决 Lambda 与 AWS CloudFormation 集成的限制，模板创建了一个在部署期间运行的额外函数。所有 Lambda 函数都附带一个 CloudWatch Logs 日志组，用于存储函数执行的输出。但是，在第一次调用函数之前，不会创建日志组。

为了创建订阅（它取决于日志组是否存在），应用程序使用第三个 Lambda 函数来调用随机错误函数。模板包含内联初级函数的代码。AWS CloudFormation 自定义资源在部署期间调用它。DependsOn 属性确保在订阅之前创建日志流和基于资源的策略。

Lambda 函数的滚动部署

使用滚动部署来控制与引入 Lambda 函数的新版本相关的风险。在滚动部署中，系统会自动部署函数的新版本，并逐步向新版本发送数量不断增加的流量。您可以配置的参数包括流量和增长率。

您可以使用 AWS CodeDeploy 和 AWS SAM 配置滚动部署。CodeDeploy 是一项服务，可自动将应用程序部署到 Amazon 计算平台（例如 Amazon EC2 和 AWS Lambda）。有关更多信息，请参阅[什么是 CodeDeploy？](#)。通过使用 CodeDeploy 部署您的 Lambda 函数，您可以轻松地监控部署状态，并在检测到任何问题时启动回滚。

AWS SAM 是一个开源框架，用于构建无服务器应用程序。您可以创建 AWS SAM 模板（以 YAML 格式），以便指定滚动部署所需的组件的配置。AWS SAM 使用模板来创建和配置组件。有关更多信息，请参阅[什么是 AWS 无服务器应用程序模型？](#)

在滚动部署中，AWS SAM 会执行以下任务：

- 它会配置您的 Lambda 函数并创建别名。
别名路由配置是实施滚动部署的基础功能。
- 它会创建一个 CodeDeploy 应用程序和一个部署组。
部署组可管理滚动部署和回滚（如果需要）。
- 它会检测您何时创建 Lambda 函数的新版本。
- 它会触发 CodeDeploy 以启动部署的新版本。

示例 AWS SAM Lambda 模板

下面的示例演示了用于简单滚动部署的 [AWS SAM 模板](#)。

```
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
```

此模板定义一个名为 myDateTimeFunction 的 Lambda 函数，其中包含以下属性。

AutoPublishAlias

`AutoPublishAlias` 属性创建一个名为 `live` 的别名。此外，AWS SAM 框架会自动检测您何时为函数保存新代码。然后，框架发布新的函数版本并更新 `live` 别名以便指向新版本。

DeploymentPreference

`DeploymentPreference` 属性决定 CodeDeploy 应用程序将流量从 Lambda 函数的原始版本转移到新版本的速率。`Linear10PercentEvery2Minutes` 值每两分钟将额外 10% 的流量转移到新版本。

有关预定义部署配置的列表，请参阅[部署配置](#)。

有关如何将 Lambda 函数与 CodeDeploy 结合使用的详细教程，请参阅[使用 CodeDeploy 部署更新的 Lambda 函数](#)。

常见 Lambda 应用程序类型和使用案例

在 AWS Lambda 上构建应用程序时，核心组件是 Lambda 函数和触发器。触发器是调用函数的 AWS 服务或应用程序，而 Lambda 函数是处理事件的代码和运行时。为了清晰起见，请考虑以下情景：

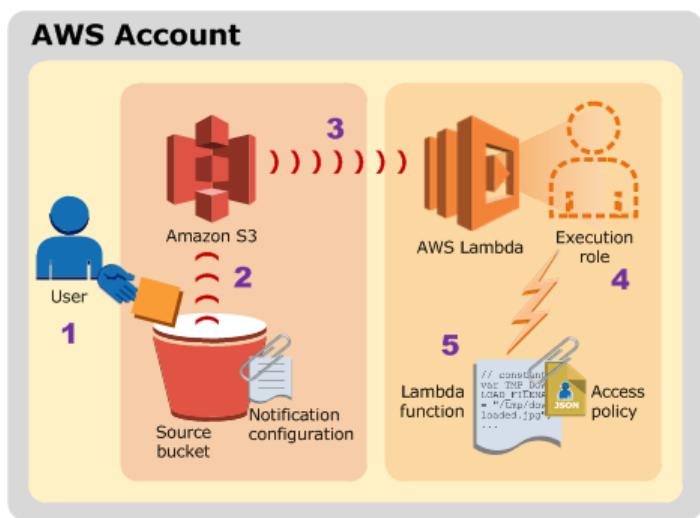
- **文件处理** – 假设您有一个照片共享应用程序。人们使用您的应用程序上传照片，应用程序将这些用户照片存储到 Amazon S3 存储桶中。然后，您的应用程序创建每个用户照片的缩略图版本，并在用户的资料页面上显示这些缩略图。在这种情景下，您可以选择创建 Lambda 函数来自动创建缩略图。Amazon S3 是支持的 AWS 事件源之一，可以发布对象创建的事件并调用您的 Lambda 函数。您的 Lambda 函数代码可以从 S3 存储桶读取照片对象、创建缩略图版本，然后将其保存到其他 S3 存储桶中。
- **数据和分析** – 假设您在构建分析应用程序并将原始数据存储到 DynamoDB 表中。在您编写、更新或删除表中的项目时，DynamoDB 流可以将项目更新事件发布到与表关联的流。在这种情况下，事件数据提供项目键、事件名称（例如插入、更新和删除）以及其他相关详细信息。您可以编写 Lambda 函数，通过聚合原始数据来生成自定义指标。
- **网站** – 假设您在创建网站并且希望在 Lambda 上托管后端逻辑。您可以在 HTTP 上调用 Lambda 函数，使用 Amazon API Gateway 作为 HTTP 终端节点。现在，您的 Web 客户端可以调用 API，然后 API 网关可将请求路由到 Lambda。
- **移动应用程序** – 假设您有生成事件的自定义移动应用程序。您可创建 Lambda 函数来处理由自定义应用程序发布的事件。例如，在此情景中，您可以配置 Lambda 函数来处理自定义移动应用程序中的点击。

AWS Lambda 支持将多种 AWS 服务作为事件源。有关更多信息，请参阅[将 AWS Lambda 与其他服务结合使用 \(p. 138\)](#)。当您配置了这些事件源触发 Lambda 函数时，Lambda 函数在出现事件时自动调用。您可以定义事件源映射，这是您如何确定要跟踪的事件以及要调用的 Lambda 函数。

以下为事件源和端到端体验工作方式的介绍性示例。

示例 1：Amazon S3 推送事件并调用 Lambda 函数

Amazon S3 可以在存储桶上发布不同类型的事件，例如 PUT、POST、COPY 和 DELETE 对象事件。使用存储桶通知功能，您可以配置事件源映射，引导 Amazon S3 在出现特定事件类型时调用 Lambda 函数，如下图中所示。



图中说明了以下序列：

1. 用户在存储桶中创建对象。
2. Amazon S3 检测到对象创建事件。
3. Amazon S3 使用由[执行角色 \(p. 30\)](#)提供的权限来调用 Lambda 函数。
4. AWS Lambda 执行 Lambda 函数，指定事件作为参数。

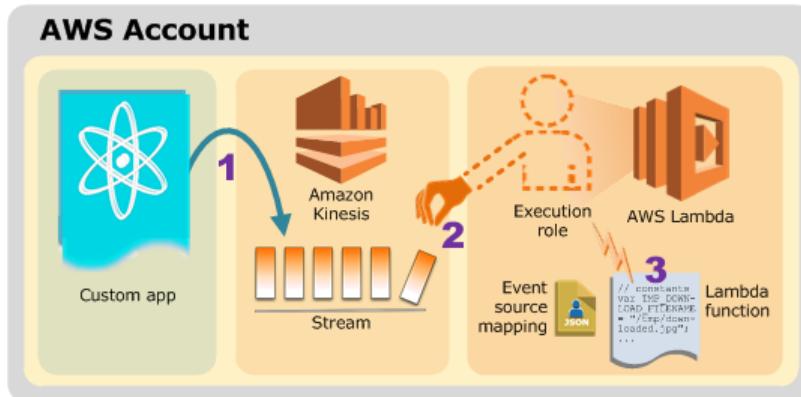
配置 Amazon S3 来将您的函数作为存储桶通知操作调用。要授予 Amazon S3 调用该函数的权限，请更新函数的[基于资源的策略 \(p. 33\)](#)。

示例 2 : AWS Lambda 从 Kinesis 流中拉取事件并调用 Lambda 函数

对于基于轮询的事件源，AWS Lambda 轮询源，然后在源上检测到记录时调用 Lambda 函数。

- [CreateEventSourceMapping \(p. 415\)](#)
- [UpdateEventSourceMapping \(p. 547\)](#)

下图显示了自定义应用程序如何将记录写入 Kinesis 流中。



图中说明了以下序列：

1. 自定义应用程序将记录写入 Kinesis 流。
2. AWS Lambda 持续轮询流，在服务检测到流上的新记录时调用 Lambda 函数。AWS Lambda 根据您在 Lambda 中创建的事件源映射来确定要轮询的流以及调用的 Lambda 函数。
3. 使用传入事件调用 Lambda 函数。

使用基于流的事件源时，您可以在 AWS Lambda 中创建事件源映射。Lambda 以同步方式调用该函数来从流读取项目。您不必授予 Lambda 调用该函数的权限，但它需要读取流的权限。

使用 AWS Lambda 函数的最佳实践

以下是推荐的使用 AWS Lambda 的最佳实践：

主题

- [函数代码 \(p. 135\)](#)
- [函数配置 \(p. 136\)](#)
- [警报与指标 \(p. 136\)](#)
- [流事件调用 \(p. 136\)](#)

函数代码

- 从核心逻辑中分离 Lambda 处理程序。这样您可以创建更容易进行单元测试的函数。在 Node.js 中可能如下所示：

```
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
}
```

- 利用执行上下文重用来提高函数性能。初始化开发工具包客户端和函数处理程序之外的数据库连接，并在 /tmp 目录中本地缓存静态资产。由函数的同一实例处理的后续调用可重用这些资源。这将节省执行时间和成本。

为了避免调用之间潜在的数据泄露，请不要使用执行上下文来存储用户数据、事件或其他具有安全影响的信息。。如果您的函数依赖于无法存储在处理程序的内存中的可变状态，请考虑为每个用户创建单独的函数或单独的函数版本。

- 使用[环境变量 \(p. 49\)](#)将操作参数传递给函数。例如，您在写入 Amazon S3 存储桶时，不应对要写入的存储桶名称进行硬编码，而应将存储桶名称配置为环境变量。
- 控制函数部署程序包中的依赖关系。AWS Lambda 执行环境中包括若干库，例如适用于 Node.js 和 Python 运行时的 AWS 开发工具包（完整列表位于此处：[AWS Lambda 运行时 \(p. 108\)](#)）。Lambda 会定期更新这些库，以支持最新的功能组合和安全更新。这些更新可能会使 Lambda 函数的行为发生细微变化。要完全控制您的函数所用的依赖项，请使用部署程序包来打包所有依赖项。
- 将部署程序包大小精简为只包含运行时必要的部分。这样会减少调用前下载和解压缩部署程序包所需的时间。对于用 Java 或 .NET Core 编写的函数，请不要将整个 AWS 开发工具包库作为部署程序包的一部分上传，而是要根据所需的模块有选择地挑选开发工具包中的组件（例如 DynamoDB、Amazon S3 开发工具包模块和 [Lambda 核心库](#)）。

- 将依赖关系 .jar 文件置于单独的 /lib 目录中，可减少 Lambda 解压缩部署程序包（用 Java 编写）所需的时间。这样比将函数的所有代码置于具有大量 .class 文件的同一 jar 中要快。有关说明，请参阅[Java 中的 AWS Lambda 部署程序包 \(p. 302\)](#)。
- 将依赖关系的复杂性降至最低。首选[执行上下文 \(p. 109\)](#)启动时可以快速加载的更简单的框架。例如，首选更简单的 Java 依赖关系注入 (IoC) 框架，如 [Dagger](#) 或 [Guice](#)，而不是更复杂的 [Spring Framework](#)。
- 避免在 Lambda 函数中使用递归代码，因为如果使用递归代码，函数便会自动调用自身，直到满足某些任意条件为止。这可能会导致意想不到的函数调用用量和升级成本。如果您意外地执行此操作，请立即将函数并发执行数限制设置为 0 来限制对函数的所有调用，同时更新代码。

函数配置

- 对您的 Lambda 函数进行性能测试是确保选择最佳内存大小配置的关键环节。增加内存大小会触发函数可用 CPU 的同等水平的增加。函数的内存使用率是根据调用情况确定的，可以在 [AWS CloudWatch 日志](#) 中查看。每次调用都将生成一个 REPORT: 条目，如下所示：

```
REPORT RequestId: 3604209a-e9a3-11e6-939a-754dd98c7be3 Duration: 12.34 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 18 MB
```

分析 Max Memory Used: 字段能够确定函数是否需要更多内存，或函数的内存大小是否过度配置。

- 对您的 Lambda 函数进行加载测试，确定最佳超时值。分析函数的运行时间很重要，这样更容易确定依赖关系服务是否有问题，这些问题可能导致并发函数的增加超出您的预期。如果您的 Lambda 函数进行网络调用的资源无法处理 Lambda 扩展，这就更加重要。
- 设置 IAM 策略时使用最严格的权限。了解您的 Lambda 函数所需的资源和操作，并限制这些权限的执行角色。有关更多信息，请参阅[AWS Lambda 权限 \(p. 30\)](#)。
- 熟悉[AWS Lambda 限制 \(p. 27\)](#)。在确定运行时资源限制时，负载大小、文件描述符和 /tmp 空间通常会被忽略。
- 删除不再使用的 Lambda 函数。这样，未使用的函数就不会不必要地占用有限的部署程序包空间。
- 如果您使用 Amazon Simple Queue Service 作为事件源，请确保该函数的预计执行时间值不超过队列上的[可见性超时值](#)。这适用于 [CreateFunction \(p. 421\)](#) 和 [UpdateFunctionConfiguration \(p. 560\)](#)。
 - 对于 CreateFunction，AWS Lambda 将使函数创建流程失败。
 - 对于 UpdateFunctionConfiguration，它可能会导致该函数的重复调用。

警报与指标

- 使用[使用 AWS Lambda 函数指标 \(p. 368\)](#) 和 [CloudWatch 警报](#)，不要在您的 Lambda 函数代码中创建和更新指标。跟踪 Lambda 函数的运行状况是更加有效的方式，这样您就可以在开发过程的早期发现问题。例如，您可以根据 Lambda 函数执行时间的预计持续时间配置警报，以解决函数代码引起的瓶颈或延迟。
- 利用您的日志记录库和[AWS Lambda 指标和维度](#)捕捉应用程序错误（例如，ERR、ERROR、WARNING 等）

流事件调用

- 测试不同批处理和记录的大小，这样每个事件源的轮询频率都会根据函数完成任务的速度进行调整。[BatchSize \(p. 416\)](#) 控制每次调用可向您的函数发送记录的最大数量。批处理大小如果较大，通常可以更有效地吸收大量记录的调用开销，从而增加吞吐量。

默认情况下，只要流中有记录，Lambda 就会调用您的函数。如果从流中读取的批处理中只有一条记录，则 Lambda 只向该函数发送一条记录。为避免在少量记录的情况下调用该函数，您可以通过配置批处理时

段让事件源缓冲记录，最多可达 5 分钟。在调用该函数之前，Lambda 会继续从流中读取记录，直到收集了完整批次，或者直到批处理时段到期。

- 通过增加分片提高 Kinesis 流处理吞吐量。一个 Kinesis 流由一个或多个分片组成。Lambda 轮询每个分片时最多会使用一个并发调用。例如，如果您的流有 100 个活跃分片，则最多可以并发运行 100 个 Lambda 函数调用。增加分片数量会直接增加 Lambda 函数并发调用的最大数量，还可增加 Kinesis 流处理的吞吐量。如果您增加 Kinesis 流中的分片数量，请确保您已为数据选择了合适的分区键（请参阅[分区键](#)），这样相关记录将会位于同一分片中，而且您的数据也可合理分配。
- 在 `IteratorAge` 上使用 [Amazon CloudWatch](#)，确定是否正在处理您的 Kinesis 流。例如，将 CloudWatch 警报的最大值设置配置为 30000（30 秒）。

将 AWS Lambda 与其他服务结合使用

AWS Lambda 与其他 AWS 服务集成以调用函数。您可以配置触发器来调用函数以响应资源生命周期事件、响应传入的 HTTP 请求、使用队列中的事件或[按计划运行 \(p. 167\)](#)。

与 Lambda 集成的每个服务都将数据作为事件以 JSON 形式发送给您的函数。对于每种事件类型，事件文档的结构各不相同，并且包含关于触发函数的资源或请求的数据。Lambda 运行时将事件转换为对象并将其传递给您的函数。

以下示例显示了一个来自[应用程序负载均衡器 \(p. 139\)](#)的测试事件，该事件表示对 /lambda?query=1234ABCD 的 GET 请求。

Example 来自应用程序负载均衡器的事件

```
{
  "requestContext": {
    "elb": {
      "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:123456789012:targetgroup/lambdata-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

Lambda 运行时将事件文档转换为一个对象，并将该对象传递给[函数处理程序 \(p. 16\)](#)。对于编译型语言，Lambda 在库中提供事件类型的定义。请参阅以下主题了解更多信息。

- [使用 Java 构建 Lambda 函数 \(p. 301\)](#)
- [使用 Go 构建 Lambda 函数 \(p. 330\)](#)
- [使用 C# 构建 Lambda 函数 \(p. 342\)](#)
- [使用 PowerShell 构建 Lambda 函数 \(p. 359\)](#)

对于生成队列或数据流的服务，您可以在 Lambda 中创建[事件源映射 \(p. 90\)](#)，并授予 Lambda 权限来访问[执行角色 \(p. 30\)](#)中的其他服务。Lambda 从其他服务读取数据、创建事件和调用您的函数。

Lambda 从其读取事件的服务

- [Amazon Kinesis \(p. 212\)](#)
- [Amazon DynamoDB \(p. 184\)](#)
- [Amazon Simple Queue Service \(p. 255\)](#)

其他服务直接调用您的函数。您在函数的[基于资源的策略 \(p. 33\)](#)中授予其他服务权限，并配置其他服务来生成事件和调用您的函数。取决于服务，调用可以是同步的，也可以是异步的。对于同步调用，其他服务等待来自您的函数的响应，并可能在[遇到错误时重试 \(p. 98\)](#)。

同步调用 Lambda 函数的服务

- [Elastic Load Balancing \(应用程序负载均衡器 \) \(p. 139\)](#)
- [Amazon Cognito \(p. 183\)](#)
- [Amazon Lex \(p. 226\)](#)
- [Amazon Alexa \(p. 141\)](#)
- [Amazon API Gateway \(p. 141\)](#)
- [Amazon CloudFront \(Lambda@Edge\) \(p. 174\)](#)
- [Amazon Kinesis Data Firehose \(p. 226\)](#)
- [AWS Step Functions](#)
- [Amazon Simple Storage Service 批处理 \(p. 246\)](#)

对于异步调用，Lambda 先将事件排队，然后再将事件传递给您的函数。一旦事件进入队列，其他服务就会获得成功响应，它们不知道之后会发生什么。如果出现错误，Lambda 负责重试 (p. 98)，并可能将失败的事件发送到您配置的[死信队列 \(p. 88\)](#)。

异步调用 Lambda 函数的服务

- [Amazon Simple Storage Service \(p. 233\)](#)
- [Amazon Simple Notification Service \(p. 250\)](#)
- [Amazon Simple Email Service \(p. 248\)](#)
- [AWS CloudFormation \(p. 172\)](#)
- [Amazon CloudWatch Logs \(p. 172\)](#)
- [Amazon CloudWatch Events \(p. 167\)](#)
- [AWS CodeCommit \(p. 176\)](#)
- [AWS Config \(p. 184\)](#)
- [AWS IoT \(p. 210\)](#)
- [AWS IoT 事件 \(p. 211\)](#)
- [AWS CodePipeline \(p. 176\)](#)

有关每项服务的详细信息以及可用于测试函数的示例事件，请参阅本章中的主题。

将 AWS Lambda 与 应用程序负载均衡器 结合使用

您可使用 Lambda 函数处理来自 应用程序负载均衡器 的请求。Elastic Load Balancing 支持 Lambda 函数作为 应用程序负载均衡器 的目标。使用负载均衡器规则，基于路径或标头值将 HTTP 请求路由到一个函数。处理请求并从 Lambda 函数返回 HTTP 响应。

Elastic Load Balancing 将使用包含请求正文和元数据的事件同步调用 Lambda 函数。

Example 应用程序负载均衡器 请求事件

```
{  
    "requestContext": {  
        "elb": {  
            "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-2:123456789012:targetgroup/lambd-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=1-5c36348-3d683b8b04734faae651f476",  
        "x-forwarded-for": "72.12.164.125",  
        "x-forwarded-port": "80",  
        "x-forwarded-proto": "http",  
        "x-imforwards": "20"  
    },  
    "body": "",  
    "isBase64Encoded": false  
}
```

您的函数将处理事件并以 JSON 格式返回对负载均衡器的响应文档。Elastic Load Balancing 会将文档转换为 HTTP 成功或错误响应并将其返回给用户。

Example 响应文档格式

```
{  
    "statusCode": 200,  
    "statusDescription": "200 OK",  
    "isBase64Encoded": False,  
    "headers": {  
        "Content-Type": "text/html"  
    },  
    "body": "<h1>Hello from Lambda!</h1>"  
}
```

要将 应用程序负载均衡器 配置为函数触发器，请为 Elastic Load Balancing 授予执行函数的权限，创建将请求路由到函数的目标组，并将规则添加到将请求发送到目标组的负载均衡器。

使用 add-permission 命令将权限语句添加到函数的基于资源的策略。

```
$ aws lambda add-permission --function-name alb-function \  
--statement-id load-balancer --action "lambda:InvokeFunction" \  
--principal elasticloadbalancing.amazonaws.com  
{  
    "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\"}"  
}
```

有关配置 应用程序负载均衡器 倾听器和目标组的说明，请参阅 Application Load Balancer 用户指南 中的 [作为目标组的 Lambda 函数](#)。

将 AWS Lambda 与 Alexa 结合使用

您可以使用 Lambda 函数构建服务，例如，赋予 Alexa 新的技能、在 Amazon Echo 上增加语音辅助功能。Alexa Skills Kit 提供了创建此类新技能（由以 Lambda 函数形式运行的您自己的服务提供）的 API、工具和文档。Amazon Echo 用户可通过询问 Alexa 问题或发出请求来访问这些新技能。

可在 GitHub 上找到 Alexa Skills Kit。

- [Alexa Skills Kit SDK for Node.js](#)
- [Alexa Skills Kit SDK for Java](#)

Example Alexa Smart Home 事件

```
{  
  "header": {  
    "payloadVersion": "1",  
    "namespace": "Control",  
    "name": "SwitchOnOffRequest"  
  },  
  "payload": {  
    "switchControlAction": "TURN_ON",  
    "appliance": {  
      "additionalApplianceDetails": {  
        "key2": "value2",  
        "key1": "value1"  
      },  
      "applianceId": "sampleId"  
    },  
    "accessToken": "sampleAccessToken"  
  }  
}
```

有关更多信息，请参阅 [Alexa Skills Kit 入门](#)。

配合使用 AWS Lambda 和 Amazon API Gateway

您可以使用 Amazon API Gateway 为 Lambda 函数创建带有 HTTP 终端节点的 Web API。API 网关 提供了相关工具用于创建和记录将 HTTP 请求传送到 Lambda 函数的 Web API。您可以使用身份验证和授权控制来保护对 API 的访问。您的 API 可以通过互联网传输流量，也可以仅允许在您的 VPC 内访问。

向 Lambda 函数添加公有终端节点

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Designer 下方，选择 Add trigger (添加触发器)。
4. 选择 API Gateway (API 网关)。
5. 对于 API，请选择 Create an API (创建 API)。
6. 对于 Security (安全性)，请选择 Open (打开)。
7. 选择 Add。

在设计器中选择 API Gateway (API 网关) 触发器后，选择要使用 API 网关 调用函数的终端节点。

The screenshot shows the AWS Lambda trigger configuration interface. At the top, there is a search bar and a 'Triggers' button. Below that, a list of triggers is shown, with 'API Gateway' selected. A modal window titled 'API Gateway' is open, displaying a single item named 'nodejs-apig'. The item has an 'Enabled' status and a 'Delete' button. It includes an 'ARN' field: 'arn:aws:execute-api:us-east-2:123456789012:5hqaxmpl4f/*:GET/'. Below this, it shows the 'API endpoint' as '<https://5hqaxmpl4f.execute-api.us-east-2.amazonaws.com/api/>' with a cursor icon over it. It also specifies 'API type: rest' and 'Authorization: NONE'.

API 网关 API 由阶段、资源、方法和集成组成。阶段和资源决定终端节点的路径：

API 路径格式

- /prod/ – prod 阶段和根资源。
- /prod/user – prod 阶段和 user 资源。
- /dev/{proxy+} – dev 阶段中的任何路线。
- / – (HTTP API) 默认阶段和根资源。

Lambda 集成将路径和 HTTP 方法的组合映射到 Lambda 函数。您可以将 API 网关 配置为按原样 (自定义集成) 传递 HTTP 请求的正文，或者将请求正文封装在包含所有请求信息 (包括标头、资源、路径和方法) 的文档中。

Amazon API Gateway 会使用包含 HTTP 请求的 JSON 表示形式的事件 [同步 \(p. 81\)](#) 调用您的函数。对于自定义集成，该事件为请求的正文。对于代理集成，该事件具有已确定的结构。以下示例显示了来自 API 网关 REST API 的代理事件。

Example `event.json` API 网关 代理事件 (REST API)

```
{  
    "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-7491f09xmp179d18acf3d050",  
        ...  
    },  
    "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
}

```

此示例显示了用于获取 REST API Prod 阶段根路径的 GET 请求所对应的事件。事件形状和内容因 [API 类型 \(p. 146\)](#) 和配置而异。

API 网关会等待函数响应并将结果转发给调用方。对于自定义集成，您可以定义集成响应和方法响应，以将函数的输出转换为 HTTP 响应。对于代理集成，函数必须以特定格式的响应形式来做出响应。

以下示例显示了来自 Node.js 函数的响应对象。该响应对象表示包含 JSON 文档的成功 HTTP 响应。

Example `index.js` – 代理集成响应对象 (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}"
}

```

Lambda 运行时会将响应对象序列化为 JSON 并将其发送给 API。此 API 会解析该响应并用它来创建 HTTP 响应，然后再将其发送到发出原始请求的客户端。

Example HTTP 响应

```

< 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-cccecxmplbae116148e52f036
<
{
    "TotalCodeSize": 104330022,
    "FunctionCount": 26
}

```

API 中的资源会定义一个或多个方法，如 GET 或 POST。方法具有将请求传送给 Lambda 函数或其他集成类型的集成。您可以单独定义每个资源和方法，也可以使用特定的资源和方法类型来匹配属于特定模式的所有请求。代理资源会捕获某个资源下的所有路径。ANY 方法会捕获所有 HTTP 方法。

小节目录

- [权限 \(p. 144\)](#)
- [利用 API 网关 API 处理错误 \(p. 145\)](#)
- [选择 API 类型 \(p. 146\)](#)
- [示例应用程序 \(p. 147\)](#)
- [教程：将 AWS Lambda 与 Amazon API Gateway 结合使用 \(p. 147\)](#)
- [示例函数代码 \(p. 155\)](#)
- [使用 Lambda 和 API 网关 创建简单的微服务 \(p. 158\)](#)
- [API 网关 应用程序的 AWS SAM 模板 \(p. 159\)](#)

权限

Amazon API Gateway 将从函数的[基于资源的策略 \(p. 33\)](#)获取调用函数的权限。您可以授予对整个 API 的调用权限，也可以仅授予对某个阶段、资源或方法的有限访问权限。

当您使用 Lambda 控制台、API 网关 控制台或 AWS SAM 模板向函数添加 API 时，会自动更新函数的基于资源的策略。以下示例显示了一个函数策略，其中包含通过 AWS SAM 模板添加的语句。

Example 函数策略

```
{  
    "Version": "2012-10-17",  
    "Id": "default",  
    "Statement": [  
        {  
            "Sid": "nodejs-apig-functiongetEndpointPermissionProd-BWDBXMPLXE2F",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "apigateway.amazonaws.com"  
            },  
            "Action": "lambda:InvokeFunction",  
            "Resource": "arn:aws:lambda:us-east-2:123456789012:function:nodejs-apig-  
function-1G3MXMPLXVXYI",  
            "Condition": {  
                "ArnLike": {  
                    "AWS:SourceArn": "arn:aws:execute-api:us-east-2:123456789012:ktyvxmpls1/*/GET/"  
                }  
            }  
        }  
    ]  
}
```

在 Lambda 控制台的 Permissions (权限) 选项卡中[确认函数策略 \(p. 33\)](#)。

您可以通过以下 API 操作手动管理函数策略权限：

- [AddPermission \(p. 407\)](#)
- [RemovePermission \(p. 537\)](#)
- [GetPolicy \(p. 477\)](#)

使用 add-permission 命令，可授予对现有 API 的调用权限。

```
$ 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/"
```

```
{  
    "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:mmh1xmpli7/default/GET  
    \"/\"}}}"  
}
```

Note

如果您的函数和 API 位于不同的区域，则源 ARN 中的区域标识符必须与函数的区域（而不是 API 的区域）匹配。当 API 网关 调用函数时，它会使用基于 API 的 ARN 的资源 ARN，但该资源 ARN 已被修改为与函数的区域相匹配。

此示例中的源 ARN 授予对 API 默认阶段根资源 GET 方法的集成的权限，其 ID 为 `mmh1xmpli7`。您可以在源 ARN 中使用星号授予对多个阶段、方法或资源的权限。

资源模式

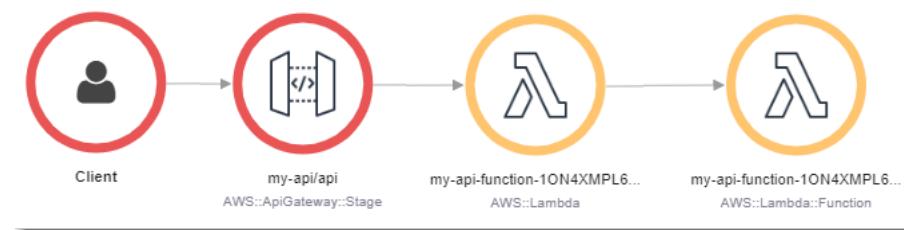
- `mmh1xmpli7/*/GET/*` – 对所有阶段中的所有资源调用 GET 方法。
- `mmh1xmpli7/prod/ANY/user` – 对 prod 阶段中的 user 资源调用 ANY 方法。
- `mmh1xmpli7/*/*/*` – 对所有阶段中的所有资源调用 ANY 方法。

有关查看策略和删除语句的详细信息，请参阅[清除基于资源的策略 \(p. 37\)](#)。

利用 API 网关 API 处理错误

API 网关 将所有调用和函数错误视为内部错误。如果 Lambda API 拒绝调用请求，则 API 网关 返回 500 错误代码。如果函数运行但返回错误，或返回格式错误的响应，则 API 网关 返回 502。在这两种情况下，来自 API 网关 的响应的正文都是 `{"message": "Internal server error"}`。

以下示例显示导致 API 网关 中出现函数错误和 502 的请求的 X-Ray 跟踪映射。客户端收到通用错误消息。

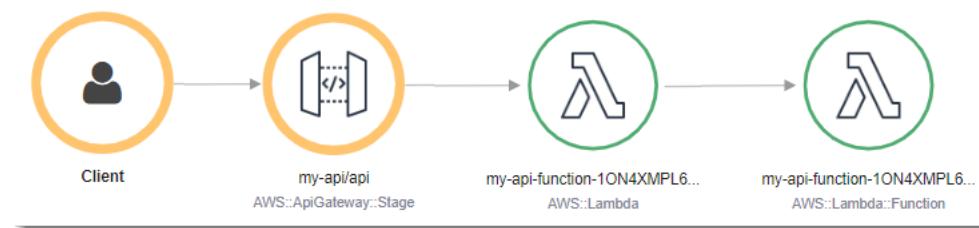


要自定义错误响应，您必须捕获代码中的错误并以所需格式设置响应的格式。

Example `index.js` – 错误格式设置

```
var formatError = function(error){  
    var response = {  
        "statusCode": error.statusCode,  
        "headers": {  
            "Content-Type": "text/plain",  
            "x-amzn-ErrorType": error.code  
        },  
        "isBase64Encoded": false,  
        "body": error.code + ": " + error.message  
    }  
    return response  
}
```

API 网关 将此响应转换为带有自定义状态代码和正文的 HTTP 错误。在跟踪映射中，函数节点为绿色，因为它处理了错误。



选择 API 类型

API 网关 支持三种可调用 Lambda 函数的 API：

- HTTP API – 一种轻型的低延迟 RESTful API。
- REST API – 一种功能丰富的可定制 RESTful API。
- WebSocket API – 一种 Web API，可与客户端保持持久连接并进行全双工通信。

HTTP API 和 REST API 都是用于处理 HTTP 请求并返回响应的 RESTful API。HTTP API 后推出，是使用 API 网关 版本 2 API 构建的。以下是 HTTP API 的新功能：

HTTP API 功能

- 自动部署 – 当您修改路线或集成时，更改会自动部署到启用了自动部署的阶段。
- 默认阶段 – 您可以创建默认阶段 (`$default`)，以便在 API URL 的根路径处提供请求。对于具名阶段，必须在路径的开头包含阶段名称。
- CORS 配置 – 您可以配置 API，使其将 CORS 标头添加到传出响应中，而不是在函数代码中手动添加。

REST API 是 API 网关 自发布起就支持的典型 RESTful API。REST API 现在具有更多的自定义、集成和管理功能。

REST API 功能

- 集成类型 – REST API 支持自定义 Lambda 集成。使用自定义集成，您可以直接将请求正文发送到函数，也可以将其应用转换模板，然后再发送到函数。
- 访问控制 – REST API 支持更多身份验证和授权选项。
- 监控和跟踪 – REST API 支持 AWS X-Ray 跟踪和其他日志记录选项。

要查看详细的比较，请参阅 API 网关 开发人员指南中的[在 HTTP API 和 REST API 之间选择](#)。

WebSocket API 也使用 API 网关 版本 2 API 并支持类似的功能集。对于受益于客户端和 API 之间的持久连接的应用程序，请使用 WebSocket API。WebSocket API 提供全双工通信，这意味着客户端和 API 都可以持续发送消息，而无需等待响应。

WebSocket API 和 HTTP API 可支持简化的事件格式（2.0 版）。以下示例显示了来自 HTTP API 的事件。

Example `event-v2.json` API 网关 代理事件 (HTTP API)

```
{  
  "version": "2.0",  
  "routeKey": "ANY /nodejs-apig-function-1G3XMPLZXVXYI",  
  "rawPath": "/default/nodejs-apig-function-1G3XMPLZXVXYI",  
  "rawQueryString": "",  
  "stage": "prod",  
  "requestContext": {  
    "accountId": "123456789012",  
    "apiId": "1G3XMPLZXVXYI",  
    "stage": "prod",  
    "requestId": "12345678901234567890123456789012",  
    "resourceId": "1G3XMPLZXVXYI",  
    "resourcePath": "/nodejs-apig-function-1G3XMPLZXVXYI",  
    "httpMethod": "GET",  
    "path": "/nodejs-apig-function-1G3XMPLZXVXYI",  
    "domainName": "apigateway.us-east-1.amazonaws.com",  
    "region": "us-east-1",  
    "identity": {  
      "principal": "arn:aws:iam::123456789012:root",  
      "accessType": "AWS_IAM",  
      "isGateway": true,  
      "cognitoPool": null,  
      "cognitoIdentity": null,  
      "cognitoAuthentication": null,  
      "apiKey": null,  
      "ip": "128.119.144.101",  
      "userAgent": "curl/7.54.0",  
      "accountId": "123456789012"  
    },  
    "apiStage": "prod"  
  },  
  "headers": {  
    "host": "apigateway.us-east-1.amazonaws.com",  
    "x-amzn-trace-id": "Root=12345678901234567890123456789012; SampledAt=1",  
    "x-amz-apigw-id": "1G3XMPLZXVXYI",  
    "x-amz-date": "20230112T123456Z",  
    "x-amz-apigw-region": "us-east-1",  
    "x-amz-svcname": "lambda",  
    "x-amz-throttle-reason": "None",  
    "x-amz-throttle-type": "Rate",  
    "x-amz-throttle-count": 0,  
    "x-amz-throttle-time": 0, "x-amz-throttle-time-until": null  
  },  
  "body": "Hello, World!",  
  "isBase64Encoded": false  
}
```

```
"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",
    "domainPrefix": "r3pmxmplak",
    "http": {
        "method": "GET",
        "path": "/default/nodejs-apig-function-1G3XMPLZXVXYI",
        "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": "JKJaXmPLvHcEsha=",
    "routeKey": "ANY /nodejs-apig-function-1G3XMPLZXVXYI",
    "stage": "default",
    "time": "10/Mar/2020:05:16:23 +0000",
    "timeEpoch": 1583817383220
},
"isBase64Encoded": true
}
```

有关更多信息，请参阅《API 网关 开发人员指南》中的 [AWS Lambda 集成](#)。

示例应用程序

本指南的 GitHub 存储库中提供了 API 网关的以下示例应用程序。

- 带有 [Node.js 的 API Gateway](#) – 具有 AWS SAM 模板的函数，该模板可用于创建启用了 AWS X-Ray 跟踪的 REST API。它包含用于部署和调用函数、测试 API 以及执行清理的脚本。

Lambda 还提供了[蓝图 \(p. 25\)](#)和[模板 \(p. 25\)](#)，您可以使用这些蓝图和模板在 Lambda 控制台中创建 API Gateway 应用程序。

教程：将 AWS Lambda 与 Amazon API Gateway 结合使用

在本示例中，您将使用 Amazon API Gateway 创建一个简单 API。Amazon API Gateway 是资源和方法的集合。在本教程中，您将创建一个资源 (`DynamoDBManager`) 并在其上定义一种方法 (`POST`)。该方法由 Lambda 函数 (`LambdaFunctionOverHttps`) 支持。也就是说，当您通过 HTTPS 终端节点调用 API 时，Amazon API Gateway 会调用 Lambda 函数。

`DynamoDBManager` 资源上的 `POST` 方法支持以下 DynamoDB 操作：

- 创建、更新和删除项目。
- 读取项目。
- 扫描项目。
- 与 DynamoDB 不相关且可用于测试的其他操作 (`echo`、`ping`)。

您在 POST 请求中发送的请求负载可标识 DynamoDB 操作并提供必需数据。例如：

- 下面是 DynamoDB 创建项目操作的示例请求负载：

```
{  
    "operation": "create",  
    "tableName": "lambda-apigateway",  
    "payload": {  
        "Item": {  
            "id": "1",  
            "name": "Bob"  
        }  
    }  
}
```

- 下面是 DynamoDB 读取项目操作的示例请求负载：

```
{  
    "operation": "read",  
    "tableName": "lambda-apigateway",  
    "payload": {  
        "Key": {  
            "id": "1"  
        }  
    }  
}
```

- 下面是 echo 操作的示例请求负载。您使用请求正文中的以下数据将一个 HTTP POST 请求发送到终端节点。

```
{  
    "operation": "echo",  
    "payload": {  
        "somekey1": "somevalue1",  
        "somekey2": "somevalue2"  
    }  
}
```

Note

API 网关 提供高级功能，例如：

- 传递整个请求 – Lambda 函数可以接收整个 HTTP 请求（而不仅仅是请求正文），并可以使用 AWS_PROXY 集成类型设置 HTTP 响应（而不仅仅是响应正文）。
- “捕获全部”方法 – 使用 ANY“捕获全部”方法将 API 资源的所有方法映射到具有单个映射的单个 Lambda 函数。
- “捕获全部”资源 – 使用新路径参数 ({proxy+}) 将资源的所有子路径映射到 Lambda 函数，而无需任何额外配置。

要了解有关这些 API 网关功能的更多信息，请参阅[为代理资源配置代理集成](#)。

先决条件

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以 [安装 Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

创建执行角色

[创建执行角色 \(p. 30\)](#)，向您的函数授予访问 AWS 资源的权限。

创建执行角色

1. 打开 IAM 控制台中的[“角色”页面](#)。
2. 选择 Create role (创建角色)。
3. 创建具有以下属性的角色。
 - Trusted entity (可信任的实体) – Lambda.
 - 角色名称 (角色名称) – **lambda-apigateway-role**.
 - 权限 – 具有对 DynamoDB 和 CloudWatch Logs 的权限的自定义策略。

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

自定义策略具有函数要将数据写入 DynamoDB 并上传日志所需的权限。记下角色的 Amazon 资源名称 (ARN) 以便稍后使用。

创建函数

以下示例代码接收 API 网关 事件输入并对其所包含的消息进行处理。为了展示这个过程，代码会将一些传入的事件数据写入 CloudWatch Logs。

Note

有关使用其他语言的示例代码，请参阅 [示例函数代码 \(p. 155\)](#)。

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}');
    }
};
```

创建函数

1. 将示例代码复制到名为 `index.js` 的文件中。
2. 创建部署程序包。

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

3. 使用 `create-function` 命令创建 Lambda 函数。

```
$ aws lambda create-function --function-name LambdaFunctionOverHttps \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs12.x \
--role arn:aws:iam::123456789012:role/service-role/lambda-apigateway-role
```

测试 Lambda 函数。

使用示例事件数据手动调用函数。建议您使用控制台来调用函数，因为控制台 UI 提供了用于查看执行结果（包括执行摘要、代码写入的日志和函数返回的结果）的用户友好型界面（因为控制台始终执行同步执行—使用 RequestResponse 调用类型来调用 Lambda 函数）。

测试 Lambda 函数

1. 将以下 JSON 复制到文件中并将其保存为 `input.txt`。

```
{  
    "operation": "echo",  
    "payload": {  
        "somekey1": "somevalue1",  
        "somekey2": "somevalue2"  
    }  
}
```

2. 执行下面的 `invoke` 命令：

```
$ aws lambda invoke --function-name LambdaFunctionOverHttps \
--payload fileb://input.txt outfile.txt
```

使用 Amazon API Gateway 创建 API

在本步骤中，您会将 Lambda 函数与您使用 Amazon API Gateway 创建的 API 中的方法关联并测试端到端体验。也就是说，在将 HTTP 请求发送到 API 方法时，Amazon API Gateway 将调用您的 Lambda 函数。

首先，通过将 Amazon API Gateway 与一种资源 (`DynamoDBManager`) 和一种方法 (`POST`) 结合使用来创建 API (`DynamoDBOperations`)。将 `POST` 方法与您的 Lambda 函数关联。然后，测试端到端体验。

创建 API

在本教程中，运行以下 `create-rest-api` 命令来创建 `DynamoDBOperations` API。

```
$ aws apigateway create-rest-api --name DynamoDBOperations  
{  
    "id": "bs8fqo6bp0",  
    "name": "DynamoDBOperations",  
    "createdDate": 1539803980,  
    "apiKeySource": "HEADER",  
    "endpointConfiguration": {  
        "types": [  
            "EDGE"  
        ]  
    }  
}
```

保存 API ID 以供在后续命令中使用。您还需要 API 根资源的 ID。要获取该 ID，请运行 `get-resources` 命令。

```
$ API=bs8fqo6bp0
```

```
$ aws apigateway get-resources --rest-api-id $API
{
    "items": [
        {
            "path": "/",
            "id": "e8kitthgdb"
        }
    ]
}
```

此时，您仅具有根资源，但您可在下一步中添加更多资源。

在 API 中创建资源

运行以下 `create-resource` 命令以在您在前一节中创建的 API 中创建资源 (`DynamoDBManager`)。

```
$ aws apigateway create-resource --rest-api-id $API --path-part DynamoDBManager \
--parent-id e8kitthgdb
{
    "path": "/DynamoDBManager",
    "pathPart": "DynamoDBManager",
    "id": "iuig5w",
    "parentId": "e8kitthgdb"
}
```

记下响应中的 ID。这是您创建的 `DynamoDBManager` 资源的 ID。

在资源上创建 POST 方法

运行以下 `put-method` 命令可在您的 API 中的 `DynamoDBManager` 资源上创建 POST 方法。

```
$ RESOURCE=iuig5w
$ aws apigateway put-method --rest-api-id $API --resource-id $RESOURCE \
--http-method POST --authorization-type NONE
{
    "apiKeyRequired": false,
    "httpMethod": "POST",
    "authorizationType": "NONE"
}
```

我们为 `--authorization-type` 参数指定了 `NONE`，这意味着针对此方法的未验证的请求受支持。此方法很适合用于测试，但在生产中，您应使用基于密钥或基于角色的身份验证。

将 Lambda 函数设置为 POST 方法的目标

运行以下命令可将 Lambda 函数设置为 POST 方法的集成点。这是在您对 POST 方法终端节点发出 HTTP 请求时，Amazon API Gateway 所调用的方法。此命令及其他命令使用包含您的账户 ID 和区域的 ARN。将这些命令保存到变量中（您可以在用于创建函数的角色 ARN 中找到您的账户 ID）。

```
$ 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
{
    "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": []
}
```

--integration-http-method 是 API 网关用于与 AWS Lambda 通信的方法。--uri 是 Amazon API Gateway 可将请求发送到的终端节点的唯一标识符。

设置对 JSON 的 POST 方法响应和集成响应的 content-type，如下所示：

- 运行以下命令以设置对 JSON 的 POST 方法响应。这是您的 API 方法返回的响应类型。

```
$ aws apigateway put-method-response --rest-api-id $API \
--resource-id $RESOURCE --http-method POST \
--status-code 200 --response-models application/json=Empty
{
    "statusCode": "200",
    "responseModels": {
        "application/json": "Empty"
    }
}
```

- 运行以下命令以设置对 JSON 的 POST 方法集成响应。这是 Lambda 函数返回的响应类型。

```
$ aws apigateway put-integration-response --rest-api-id $API \
--resource-id $RESOURCE --http-method POST \
--status-code 200 --response-templates application/json=""
{
    "statusCode": "200",
    "responseTemplates": {
        "application/json": null
    }
}
```

部署 API

在本步骤中，您会将您创建的 API 部署到名为 prod 的阶段。

```
$ aws apigateway create-deployment --rest-api-id $API --stage-name prod
{
    "id": "20vgpsz",
    "createdDate": 1539820012
}
```

向 API 授予调用权限

既然您已使用 Amazon API Gateway 创建并部署了一个 API，您便可以进行测试了。首先，您需要添加权限，以便 Amazon API Gateway 在您将 HTTP 请求发送到 POST 方法时能够调用 Lambda 函数。

为此，您需要向与 Lambda 函数关联的权限策略添加权限。运行以下 add-permission AWS Lambda 命令可为 Amazon API Gateway 服务委托人 (apigateway.amazonaws.com) 授予调用 Lambda 函数 (LambdaFunctionOverHttps) 的权限。

```
$ 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"
{
    "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\"}}\""
}
```

您必须授权此权限才能启用测试（如果您转到 Amazon API Gateway 并选择 Test 来测试 API 方法，则需要此权限）。请注意，`--source-arn` 将通配符 (*) 指定为了阶段值（仅指示测试）。这使您无需部署 API 即可进行测试。

Note

如果您的函数和 API 位于不同的区域，则源 ARN 中的区域标识符必须与函数的区域（而不是 API 的区域）匹配。

现在，再次运行同一命令，但这次您将向已部署的 API 授予调用 Lambda 函数的权限。

```
$ 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"
{
    "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\"}}\"}"
}
```

您授予此权限是为了让已部署的 API 有权调用 Lambda 函数。请注意，`--source-arn` 指定了 `prod`，这是我们在部署 API 时使用的阶段名称。

创建 Amazon DynamoDB 表

创建 Lambda 函数使用的 DynamoDB 表。

创建 DynamoDB 表

1. 打开 [DynamoDB 控制台](#)。
2. 选择 Create Table。
3. 使用以下设置创建表。
 - 表名称 – `lambda-apigateway`
 - 主键 – `id` (字符串)
4. 选择 Create。

使用 HTTP 请求触发函数

在本步骤中，您已准备好向 POST 方法终端节点发送 HTTP 请求。您可使用 Curl 或 Amazon API Gateway 提供的方法 (`test-invoker-method`)。

您可使用 Amazon API Gateway CLI 命令向资源 (DynamoDBManager) 终端节点发送 HTTP POST 请求。因为您已部署 Amazon API Gateway，所以可以使用 Curl 来调用相应的方法来实现同一操作。

Lambda 函数支持使用 `create` 操作在 DynamoDB 表中创建项目。要请求此操作，请使用以下 JSON：

Example `create-item.json`

```
{
```

```
"operation": "create",
"tableName": "lambda-apigateway",
"payload": {
    "Item": {
        "id": "1234ABCD",
        "number": 5
    }
}
```

将测试输入保存到一个名为 `create-item.json` 的文件中。运行 `test-invoke-method` Amazon API Gateway 命令可将 HTTP POST 方法请求发送到资源 (`DynamoDBManager`) 终端节点。

```
$ aws apigateway test-invoke-method --rest-api-id $API \
--resource-id $RESOURCE --http-method POST --path-with-query-string "" \
--body file://create-item.json
```

或者，您也可以使用以下 Curl 命令：

```
$ curl -X POST -d "{\"operation\":\"create\",\"tableName\":\"lambda-apigateway\",
\"payload\":{\"Item\":{\"id\":\"1\",\"name\":\"Bob\"}}}" https://$API.execute-api.
$REGION.amazonaws.com/prod/DynamoDBManager
```

要发送对您的 Lambda 函数支持的 `echo` 操作的请求，可使用以下请求负载：

Example `echo.json`

```
{
    "operation": "echo",
    "payload": {
        "somekey1": "somevalue1",
        "somekey2": "somevalue2"
    }
}
```

将测试输入保存到一个名为 `echo.json` 的文件中。运行 `test-invoke-method` Amazon API Gateway CLI 命令可使用请求正文中的 JSON 代码向资源 (`DynamoDBManager`) 终端节点发送 HTTP POST 方法请求。

```
$ aws apigateway test-invoke-method --rest-api-id $API \
--resource-id $RESOURCE --http-method POST --path-with-query-string "" \
--body file://echo.json
```

或者，您也可以使用以下 Curl 命令：

```
$ curl -X POST -d "{\"operation\":\"echo\",\"payload\":{\"somekey1\":\"somevalue1\",
\"somekey2\":\"somevalue2\"}}" https://$API.execute-api.$REGION.amazonaws.com/prod/
DynamoDBManager
```

示例函数代码

示例代码具有以下语言。

主题

- [Node.js \(p. 156\)](#)
- [Python 3 \(p. 156\)](#)

- [Go \(p. 157\)](#)

Node.js

以下示例根据请求方法处理来自 API 网关 的消息并管理 DynamoDB 文档。

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}');
    }
};
```

压缩示例代码以创建部署程序包。有关说明，请参阅[Node.js 中的 AWS Lambda 部署程序包 \(p. 268\)](#)。

Python 3

以下示例根据请求方法处理来自 API 网关 的消息并管理 DynamoDB 文档。

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))
```

压缩示例代码以创建部署程序包。有关说明，请参阅[Python 中的 AWS Lambda 部署程序包 \(p. 279\)](#)。

Go

以下示例处理来自 API 网关的消息，并记录有关请求的信息。

Example LambdaFunctionOverHttps.go

```
import (
    "strings"
    "github.com/aws/aws-lambda-go/events"
)

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
}
```

使用 `go build` 构建可执行文件并创建部署程序包。有关说明，请参阅[Go 中的 AWS Lambda 部署程序包 \(p. 330\)](#)。

使用 Lambda 和 API 网关 创建简单的微服务

在本教程中，您将使用 Lambda 控制台创建一个 Lambda 函数，并创建一个 Amazon API Gateway 终端节点来触发该函数。您可以使用任何方法（GET、POST、PATCH）调用该终端节点以触发您的 Lambda 函数。调用终端节点后，整个请求将会传递到您的 Lambda 函数。您的函数操作将取决于您调用终端节点时使用的方法：

- DELETE：从 DynamoDB 表中删除项目
- GET：扫描表并返回所有项目
- POST：创建项目
- PUT：更新项目

使用 Amazon API Gateway 创建 API

按照本部分的步骤创建新 Lambda 函数，并创建 API 网关 终端节点以触发该函数：

创建 API

1. 登录到 AWS 管理控制台，然后打开 AWS Lambda 控制台。
2. 选择 Create Lambda function。
3. 选择 Blueprint (蓝图)。
4. 在搜索栏中输入 **microservice**。选择 microservice-http-endpoint 蓝图，然后选择 Configure (配置)。
5. 配置以下设置。
 - 名称 – **lambda-microservice**。
 - 角色 – 从一个或多个模板创建新角色。
 - 角色名称 (角色名称) – **lambda-apigateway-role**。
 - 策略模板 – 简单微服务权限。
 - API – 创建新的 API。
 - 安全性 – 打开。

选择 Create function。

在完成向导并创建函数时，Lambda 会在所选 API 名称下自动创建名为 `lambda-microservice` 的代理资源。有关代理资源的更多信息，请参阅[为代理资源配置代理集成](#)。

代理资源具有 `AWS_PROXY` 集成类型和“捕获全部”方法 `ANY`。`AWS_PROXY` 集成类型会应用默认映射模板将整个请求传递到 Lambda 函数，并将 Lambda 函数的输出转换为 HTTP 响应。`ANY` 方法为支持的所有方法（包括 `GET`、`POST`、`PATCH`、`DELETE` 和其他方法）定义同样的集成设置。

测试发送 HTTPS 请求

在本步骤中，您将使用控制台来测试 Lambda 函数。此外，您还可以运行 `curl` 命令来测试端到端体验。也就是说，将 HTTPS 请求发送到您的 API 方法，并让 Amazon API Gateway 调用您的 Lambda 函数。为了

完成这些步骤，请确保您已创建了 DynamoDB 表并将其命名为“MyTable”。有关更多信息，请参阅 [创建一个 DynamoDB 表（启用流）\(p. 193\)](#)。

测试 API

- 让 MyLambdaMicroService 函数在控制台中保持打开状态，选择 Actions 选项卡，然后选择 Configure test event。
- 使用以下内容替换现有文本：

```
{  
    "httpMethod": "GET",  
    "queryStringParameters": {  
        "TableName": "MyTable"  
    }  
}
```

- 输入上面的文本后，选择 Save and test。

API 网关 应用程序的 AWS SAM 模板

下面是[教程 \(p. 147\)](#)中 Lambda 应用程序的示例 AWS SAM 模板。将以下文本复制到一个文件中，并将该文件保存到您之前创建的 ZIP 程序包旁。请注意，Handler 和 Runtime 参数值应与上一节中创建函数时所用的参数值匹配。

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
```

有关如何使用程序包和部署命令打包和部署无服务器应用程序的信息，请参阅 AWS 无服务器应用程序模型开发人员指南 中的[部署无服务器应用程序](#)。

配合使用 AWS Lambda 和 AWS CloudTrail

AWS CloudTrail 是一项服务，提供由用户、角色或 AWS 服务执行的操作的记录。CloudTrail 会将 API 调用作为事件捕获。要持续记录 AWS 账户中的事件，您可以创建跟踪。通过跟踪，CloudTrail 可将事件的日志文件传送至 Amazon S3 存储桶。

您可以利用 Amazon S3 的存储桶通知功能并指示 Amazon S3 将对象创建事件发布到 AWS Lambda。当 CloudTrail 将日志写入 S3 存储桶时，Amazon S3 随后可通过将 Amazon S3 对象创建事件作为参数传递来调用 Lambda 函数。S3 事件提供了信息，包括存储桶名称和 CloudTrail 创建的日志对象的键名称。Lambda 函数代码可读取日志对象并处理由 CloudTrail 记录的访问记录。例如，如果您的账户中发生了特定 API 调用，您可以写入 Lambda 函数代码来通知您。

在这种情况下，CloudTrail 会将访问日志写入 S3 存储桶。对于 AWS Lambda，Amazon S3 是事件源，因此 Amazon S3 会将事件发布到 AWS Lambda 并调用 Lambda 函数。

Example CloudTrail 日志

```
{  
    "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": "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-c5c98c29cbec",  
        }  
    ]  
}
```

```
        "eventID": "ec5bb073-8fa1-4d45-b03c-f07b9fc9ea18",
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
}
```

有关如何将 Amazon S3 配置为事件源的详细信息，请参阅[将 AWS Lambda 与 Amazon S3 事件结合使用 \(p. 233\)](#)。

主题

- 教程：使用 AWS CloudTrail 事件触发 Lambda 函数 (p. 161)
- 示例函数代码 (p. 165)

教程：使用 AWS CloudTrail 事件触发 Lambda 函数

您可以配置 Amazon S3 以在 AWS CloudTrail 存储 API 调用日志时将事件发布到 AWS Lambda。Lambda 函数可读取日志对象并处理由 CloudTrail 记录的访问记录。

使用以下说明创建一个 Lambda 函数，该函数在您的账户中进行特定的 API 调用时向您发送通知。该函数处理来自 Amazon S3 的通知事件，从存储桶中读取日志，并通过 Amazon SNS 主题发布警报。在本教程中，您将创建：

- 一个 CloudTrail 跟踪和一个用于保存日志的 S3 存储桶。
- 一个用于发布警报通知的 Amazon SNS 主题。
- 一个 IAM 用户角色，具有从 S3 存储桶读取项目并将日志写入 Amazon CloudWatch 的权限。
- 一个 Lambda 函数，用于处理 CloudTrail 日志并在创建 Amazon SNS 主题时发送通知。

要求

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

开始之前，请确保您具有以下工具：

- Node.js 8 (含 npm)。
- Bash shell。对于 Linux 和 macOS，默认情况下包含此项。在 Windows 10 中，您可以安装 [Windows Subsystem for Linux](#)，以获取 Windows 集成版本的 Ubuntu 和 Bash。
- AWS CLI。

步骤 1：在 CloudTrail 中创建跟踪

在创建跟踪时，CloudTrail 将 API 调用记录在日志文件中，并将这些文件存储在 Amazon S3 中。CloudTrail 日志是 JSON 格式的一系列无序事件。对于支持的 API 操作的每次调用，CloudTrail 会记录请求信息，以及发出请求的实体。日志事件包含操作名称、参数、响应值以及有关请求者的详细信息。

创建跟踪

1. 打开 [CloudTrail 控制台的 Trails \(跟踪\) 页面](#)。
2. 选择 Create trail (创建跟踪)。
3. 对于 Trail name (跟踪名称)，输入一个名称。
4. 对于 S3 bucket (S3 存储桶)，输入一个名称。

5. 选择 Create。
6. 保存存储桶 Amazon 资源名称 (ARN) 以将它添加到您稍后创建的 IAM 执行角色。

步骤 2：创建 Amazon SNS 主题

创建一个 Amazon SNS 主题，以便在发生新的对象事件时发出通知。

要创建主题，请执行以下操作

1. 打开 [Amazon SNS 控制台的 Topics \(主题\) 页](#)。
2. 选择 Create topic (创建主题)。
3. 对于 Topic name (主题名称)，输入一个名称。
4. 选择 Create topic (创建主题)。
5. 记录主题 ARN。您将在创建 IAM 执行角色和 Lambda 函数时需要它。

步骤 3：创建 IAM 执行角色

执行角色 ([p. 30](#)) 向您的函数授予访问 AWS 资源的权限。创建一个执行角色，该角色向函数授予访问 CloudWatch Logs、Amazon S3 和 Amazon SNS 的权限。

创建执行角色

1. 打开 IAM 控制台的 [Roles \(角色\) 页](#)。
2. 选择创建角色。
3. 创建具有以下属性的角色：
 - 对于 Trusted entity (可信实体)，选择 Lambda。
 - 对于 Role Name (角色名称)，输入 `lambda-cloudtrail-role`。
 - 对于 Permissions (权限)，使用以下语句创建自定义策略。将突出显示的值替换为存储桶和主题的名称。

```
{
  "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. 记录角色 ARN。您在创建 Lambda 函数时将需要它。

步骤 4：创建 Lambda 函数

以下 Lambda 函数处理 CloudTrail 日志，并在创建新的 Amazon SNS 主题时通过 Amazon SNS 发送通知。

创建函数

1. 创建一个文件夹，并为该文件夹提供一个名称来指示它是您的 Lambda 函数（例如，`lambda-cloudtrail`）。
2. 在该文件夹中，创建一个名为 `index.js` 的文件。
3. 将以下代码粘贴到 `index.js`。将 Amazon SNS 主题 ARN 替换为 Amazon S3 在您创建 Amazon SNS 主题时创建的 ARN。

```
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. 在 `lambda-cloudtrail` 文件夹中，运行以下脚本。它会创建一个 `package-lock.json` 文件和一个 `node_modules` 文件夹，处理所有依赖项。

```
$ npm install async
```

5. 运行以下脚本可创建部署程序包。

```
$ zip -r function.zip .
```

6. 使用 `create-function` 命令通过运行以下脚本来创建一个名为 CloudTrailEventProcessing 的 Lambda 函数。进行指示的替换。

```
$ 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
```

步骤 5：向 Lambda 函数策略添加权限

Lambda 函数的资源策略需要权限以允许 Amazon S3 调用函数。

向 Amazon S3 授予调用函数的权限

1. 运行以下 `add-permission` 命令。将 ARN 和账户 ID 替换为您自己的 ARN 和账户 ID。

```
$ 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
```

此命令向 Amazon S3 服务委托人 (`s3.amazonaws.com`) 授予执行 `lambda:InvokeFunction` 操作的权限。仅在满足以下条件时向 Amazon S3 授予调用权限：

- CloudTrail 将日志对象存储在指定的存储桶中。
 - 存储桶由指定的 AWS 账户拥有。如果存储桶拥有者删除一个存储桶，则其他 AWS 账户可以创建同名存储桶。该条件确保只有特定的 AWS 账户能调用您的 Lambda 函数。
2. 要查看 Lambda 函数的访问策略，请运行以下 `get-policy` 命令，并替换函数名称。

```
$ aws lambda get-policy --function-name function-name
```

步骤 6：在 Amazon S3 存储桶上配置通知

要请求 Amazon S3 将对象创建的事件发布到 Lambda，请将通知配置添加到 S3 存储桶。在配置中，指定以下内容：

- 事件类型 – 任何创建对象的事件类型。
- Lambda 函数 – 您希望 Amazon S3 调用的 Lambda 函数。

配置通知

1. 打开 [Amazon S3 控制台](#)。
2. 选择源存储桶。
3. 选择属性。
4. 在 Events (事件) 下，使用以下设置配置通知：
 - Name (名称) – **lambda-trigger**
 - Events (事件) – **All object create events**
 - Send to (发送到) – **Lambda function**
 - Lambda – **CloudTrailEventProcessing**

当 CloudTrail 将日志存储在存储桶中时，Amazon S3 会向函数发送一个事件。该事件提供了信息，包括存储桶名称和 CloudTrail 创建的日志对象的键名称。

示例函数代码

示例代码具有以下语言。

主题

- [Node.js \(p. 165\)](#)

Node.js

以下示例处理 CloudTrail 日志，并在创建 Amazon SNS 主题时发送通知。

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...');
            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);
            }
        }
    ], function (err) {
        if (err) {
            console.error('Failed to publish notifications: ', err);
        }
    });
}
```

```
        } else {
            console.log('Successfully published all notifications.');
        }
        callback(null,"message");
    });
};
```

压缩示例代码以创建部署程序包。有关说明，请参阅[Node.js 中的 AWS Lambda 部署程序包 \(p. 268\)](#)。

配合使用 AWS Lambda 和 Amazon CloudWatch Events

[Amazon CloudWatch 事件](#)可用于响应您的 AWS 资源的状态更改。当您的资源的状态发生变化时，会自动向事件流发送事件。您可以创建规则来匹配流中的选定事件并将它们发送到 AWS Lambda 函数以采取操作。例如，您可以自动调用 AWS Lambda 函数以记录 [EC2 实例](#)或 [AutoScaling 组](#)的状态。

您可使用规则目标定义在 Amazon CloudWatch 事件中保留事件源映射。有关更多信息，请参阅 Amazon CloudWatch Events API 参考 中的 [PutTargets](#) 操作。

您还可以创建一个 Lambda 函数并指示 AWS Lambda 定期执行此函数。您可以指定一个固定速率（例如，每小时或每 15 分钟执行一次 Lambda 函数），也可以指定一个 Cron 表达式。有关表达式计划的更多信息，请参阅[使用 Rate 或 Cron 来计划表达式 \(p. 171\)](#)。

Example CloudWatch Events 消息事件

```
{
    "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-1:123456789012:rule/my-schedule"
    ]
}
```

此功能仅在您使用 AWS Lambda 控制台或 AWS CLI 创建 Lambda 函数时可用。要使用 AWS CLI 配置它，请参阅[使用 AWS CLI 按计划运行 AWS Lambda 函数](#)。控制台提供 CloudWatch 事件作为事件源。创建 Lambda 函数时，选择此事件源并指定时间间隔。

如果您对函数的权限做出任何手动更改，可能需要将计划事件访问权限重新应用于您的函数。您可以使用下面的 CLI 命令执行这项操作。

```
$ aws lambda add-permission --function-name my-function \
    --action 'lambda:InvokeFunction' --principal events.amazonaws.com --statement-id
events-access \
    --source-arn arn:aws:events:*:123456789012:rule/*
```

每个 AWS 账户可以有最多 100 个 CloudWatch Events- 计划源类型的唯一事件源。其中每个事件源可以是最多五种 Lambda 函数的事件源。也就是说，您的 AWS 账户最多可以有 500 种能够按计划执行的 Lambda 函数。

控制台还提供了使用 CloudWatch Events - 计划源类型的蓝图 (lambda-canary)。利用此蓝图，您可以创建示例 Lambda 函数并测试此功能。蓝图提供的示例代码将检查特定网页和网页上的特定文本字符串是否存在。如果未找到网页或文本字符串，则 Lambda 函数会引发错误。

教程：将 AWS Lambda 用于计划的事件

在本教程中，您将执行以下操作：

- 使用 lambda-canary 蓝图创建 Lambda 函数。您将 Lambda 函数配置为每分钟运行一次。请注意，如果函数返回错误，则 AWS Lambda 会将错误指标记录到 CloudWatch。
- 将 Lambda 函数的 Errors 指标的 CloudWatch 警报配置为在 AWS Lambda 向 CloudWatch 发出错误指标时将消息发布到 Amazon SNS 主题。您将订阅 Amazon SNS 主题以接收电子邮件通知。在本教程中，您将执行以下操作来进行此设置：
 - 创建一个 Amazon SNS 主题。
 - 订阅主题以便在有新消息发布到主题时接收电子邮件通知。
 - 在 Amazon CloudWatch 中，将 Lambda 函数的 Errors 指标的警报设置为在出错时将消息发布到 SNS 主题。

先决条件

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

创建 Lambda 函数

1. 通过以下网址登录 AWS 管理控制台并打开 AWS Lambda 控制台 (<https://console.aws.amazon.com/lambda/>)。
2. 选择 Create function。
3. 选择 Blueprints (蓝图)。
4. 在搜索栏中输入 **canary**。选择 lambda-canary 蓝图，然后选择 Configure (配置)。
5. 配置以下设置。
 - 名称 – **lambda-canary**。
 - 角色 – 从一个或多个模板创建新角色。
 - 角色名称 (角色名称) – **lambda-apigateway-role**。
 - 策略模板 – 简单微服务权限。
 - 规则 – Create a new rule (创建新规则)。
 - 规则名称 – **CheckWebsiteScheduledEvent**。
 - 规则描述 – **CheckWebsiteScheduledEvent trigger**。
 - 计划表达式 – **rate(1 minute)**。
 - 已启用 – True (已选中)。
 - 环境变量
 - site (站点) – <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>。
 - expected (预期) – **What Is AWS Lambda?**。
6. 选择 Create function。

CloudWatch Events 根据计划表达式每分钟发出一个事件。该事件触发 Lambda 函数，从而验证指定页面上是否显示了预期字符串。有关表达式计划的更多信息，请参阅[使用 Rate 或 Cron 来计划表达式 \(p. 171\)](#)。

测试 Lambda 函数。

使用 Lambda 控制台所提供的示例事件测试函数。

1. 打开 Lambda 控制台 [函数页面](#)。

2. 选择 lambda-canary。
3. 在页面顶部的 Test (测试) 按钮旁，选择下拉菜单中的 Configure test events (配置测试事件)。
4. 使用 CloudWatch Events 事件模板创建新事件。
5. 选择 Create。
6. 选择 Test。

函数执行的输出将显示在页面顶部。

创建一个 Amazon SNS 主题并订阅此主题

创建一个 Amazon Simple Notification Service (Amazon SNS) 主题以在 canary 函数返回错误时收到通知。

要创建主题，请执行以下操作

1. 打开 [Amazon SNS 控制台](#)。
2. 选择 Create topic。
3. 使用以下设置创建主题。
 - 名称 – **lambda-canary-notifications**。
 - 显示名称 – **Canary**。
4. 选择 Create subscription。
5. 使用以下设置创建订阅。
 - 协议 – **Email**。
 - 终端节点 – 您的电子邮件地址。

Amazon SNS 将通过 **Canary <no-reply@sns.amazonaws.com>** 发送一封电子邮件，其中反映了主题的友好名称。使用此电子邮件中的链接确认您的地址。

配置警报

在 Amazon CloudWatch 中配置用来监控 Lambda 函数并在此函数失败时发送通知的警报。

创建警报

1. 打开 [CloudWatch 控制台](#)。
2. 选择 Alarms。
3. 选择 Create Alarm (创建警报)。
4. 选择 Alarms。
5. 使用以下设置创建警报。
 - Metrics (指标) – lambda-canary 错误。

搜索 **lambda canary errors** 以查找指标。

- 统计数据 – **Sum**。
- 从预览图上方的下拉菜单中选择统计数据。
- 名称 – **lambda-canary-alarm**。
 - Description (描述) – **Lambda canary alarm**。
 - 阈值 – 当错误为 **>=1**。

- Send notification to (发送通知到) – **lambda-canary-notifications**。

测试警报

更新函数配置以使函数返回错误，从而触发警报。

触发警报

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择 lambda-canary。
3. 在 Environment variables (环境变量) 下，选择 Edit (编辑)。
4. 将 expected (预期) 设置为 **404**。
5. 选择保存。

等待一分钟，然后在电子邮件中查看来自 Amazon SNS 的邮件。

CloudWatch Events 应用程序的 AWS SAM 模板

您可以使用 [AWS SAM](#) 构建此应用程序。要了解有关创建 AWS SAM 模板的更多信息，请参阅 AWS 无服务器应用程序模型 开发人员指南 中的 [AWS SAM 模板基础知识](#)。

下面是[教程 \(p. 168\)](#)中 Lambda 应用程序的示例 AWS SAM 模板。将以下文本复制到 .yaml 文件中，并将其保存到您之前创建的 ZIP 程序包旁。请注意，Handler 和 Runtime 参数值应与上一节中创建函数时所用的参数值匹配。

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'
```

有关如何使用程序包和部署命令打包和部署无服务器应用程序的信息，请参阅 AWS 无服务器应用程序模型开发人员指南 中的[部署无服务器应用程序](#)。

使用 Rate 或 Cron 来计划表达式

AWS Lambda 支持最高每分钟一次的频率的标准 rate 和 cron 表达式。CloudWatch Events rate 表达式具有以下格式。

```
rate(Value Unit)
```

其中，# 是一个正整数，## 可以是分钟、小时或天。对于奇异值，单位必须是单数（例如，`rate(1 day)`），而不是复数（例如，`rate(5 days)`）。

Rate 表达式示例

频率	表达式
每 5 分钟	<code>rate(5 minutes)</code>
每小时	<code>rate(1 hour)</code>
每 7 天	<code>rate(7 days)</code>

Cron 表达式具有以下格式。

```
cron(Minutes Hours Day-of-month Month Day-of-week Year)
```

Cron 表达式示例

频率	表达式
每天上午 10:15 (UTC)	<code>cron(15 10 * * ? *)</code>
星期一到星期五的下午 6:00	<code>cron(0 18 ? * MON-FRI *)</code>
每月第一天早上 8:00	<code>cron(0 8 1 * ? *)</code>
工作日每隔 10 分钟	<code>cron(0/10 * ? * MON-FRI *)</code>
工作日早上 8:00 到下午 5:55 期间每隔 5 分钟	<code>cron(0/5 8-17 ? * MON-FRI *)</code>
每月第一个星期一早上 9:00	<code>cron(0 9 ? * 2#1 *)</code>

请注意以下几点：

- 如果您使用的是 Lambda 控制台，请不要向您的表达式添加 cron 前缀。
- 日期值或星期几值之一必须是问号 (?)。

有关更多信息，请参阅 CloudWatch Events 用户指南 中的[规则的计划表达式](#)。

将 AWS Lambda 与 Amazon CloudWatch Logs 结合使用

您可以使用 Lambda 函数监视和分析来自 Amazon CloudWatch Logs 日志流的日志。为一个或多个日志流创建[订阅](#)，以在创建日志时调用函数或匹配可选模式。使用函数发送通知或将日志保存到数据库或存储。

CloudWatch Logs 通过包含日志数据的事件异步调用您的函数。数据字段的值是 Base64 编码的 ZIP 存档。

Example Amazon CloudWatch Logs 消息事件

```
{  
  "awslogs": {  
    "data"::  
      "ewogICAgIm1lc3NhZ2VUeXB1IjogIkRBVEFFTUVTUOFHRSIsCiAgICAib3duZXIIoiaIMTIzNDU2Nzg5MDEyIiwKICAgICJsb2dH  
  }  
}
```

在解码和解压缩后，日志数据为具有以下结构的 JSON 文档。

Example Amazon CloudWatch Logs 消息数据（已解码）

```
{  
  "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: 100 ms \\tMemory Size: 192 MB\\tMax Memory Used: 72 MB\\t\\n"  
    }  
  ]  
}
```

有关使用 CloudWatch Logs 作为触发器的示例应用程序，请参阅[AWS Lambda 错误处理器示例应用程序 \(p. 128\)](#)。

将 AWS Lambda 与 AWS CloudFormation 结合使用

在 AWS CloudFormation 模板中，您可以指定 Lambda 函数作为自定义资源的目标。使用自定义资源来处理参数、检索配置值或者在堆栈生命周期事件期间调用其他 AWS 服务。

以下示例调用在模板中的其他位置定义的函数。

Example – 自定义资源定义

```
Resources:  
  primerinvoke:  
    Type: AWS::CloudFormation::CustomResource  
    Version: "1.0"  
    Properties:  
      ServiceToken: !GetAtt primer.Arn  
      FunctionName: !Ref randomerror
```

服务令牌是在您创建、更新或删除堆栈时，AWS CloudFormation 所调用函数的 Amazon 资源名称 (ARN)。您还可以按原样包含 AWS CloudFormation 传递到您函数的其他属性，例如 `FunctionName`。

AWS CloudFormation 通过包含回调 URL 的事件 [异步 \(p. 83\)](#) 调用您的 Lambda 函数。

Example – AWS CloudFormation 消息事件

```
{  
  "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=28UijsZePE5I4dvukKQqM%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"  
  }  
}
```

函数负责将指示成功还是失败的响应返回到回调 URL。有关完整响应语法，请参阅[自定义资源响应对象](#)。

Example – AWS CloudFormation 自定义资源响应

```
{  
  "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"  
}
```

AWS CloudFormation 提供称为 `cfn-response` 的库来处理响应的发送。如果您在模板中定义函数，则可以按名称请求库。随后，AWS CloudFormation 将库添加到为函数创建的部署程序包。

以下示例函数调用第二个函数。如果调用成功，则函数发送成功响应到 AWS CloudFormation，并且堆栈更新继续。该模板使用 AWS 无服务器应用程序模型提供的 [AWS::Serverless::Function](#) 资源类型。

Example `error-processor/template.yml` – 自定义资源函数

```
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
```

如果模板中未定义自定义资源调用的函数，您可以从 AWS CloudFormation 用户指南 中的 [cfn-response 模块](#) 获取 `cfn-response` 的源代码。

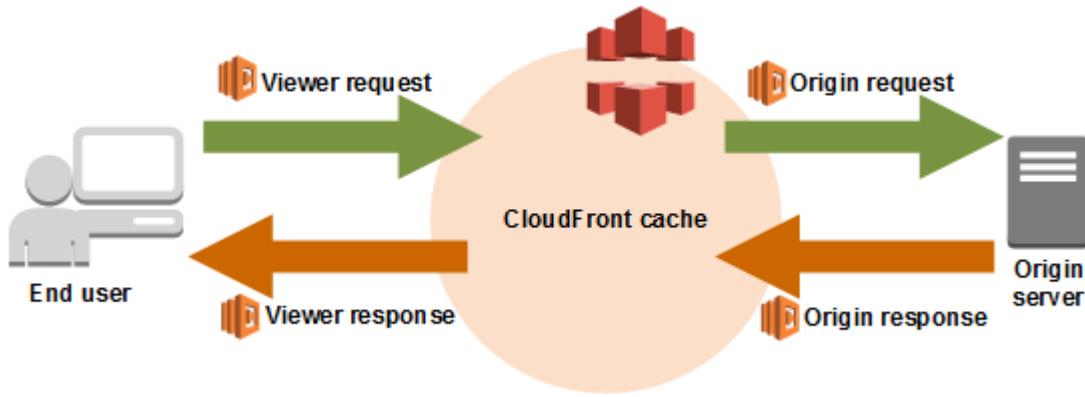
有关使用自定义资源确保首先创建函数的日志组（先于依赖于它的其他资源）的示例应用程序，请参阅[AWS Lambda 错误处理器示例应用程序 \(p. 128\)](#)。

有关自定义资源的更多信息，请参阅 AWS CloudFormation 用户指南 中的[自定义资源](#)。

将 AWS Lambda 与 CloudFront Lambda@Edge 结合使用

Lambda@Edge 允许您运行 Node.js 和 Python Lambda 函数来自定义 CloudFront 提供的内容，从而在更靠近查看器的 AWS 位置执行函数。这些函数在不提供或管理服务器的情况下运行，以响应 CloudFront 事件。您可以在以下时间点使用 Lambda 函数来更改 CloudFront 请求和响应：

- 在 CloudFront 收到查看器的请求之后 (查看器请求)
- 在 CloudFront 将请求转发到源之前 (源请求)
- 在 CloudFront 收到来自源的响应之后 (源响应)
- 在 CloudFront 将响应转发到查看器之前 (查看器响应)



Note

Lambda@Edge 支持有限的一组运行时和功能。有关详细信息，请参阅 Amazon CloudFront 开发人员指南中的 [Lambda 函数的要求和限制](#)。

您也可以生成对查看器的响应，而不必将请求发送到源。

Example CloudFront 消息事件

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

使用 Lambda@Edge，您可以构建各种解决方案，例如：

- 检查 Cookie，从而重写站点不同版本的 URL 以进行 A/B 测试。
- 根据 User-Agent 标头将不同的对象发送给您的用户，该标头包含有关提交请求的设备的信息。例如，您可以根据用户的设备向用户发送分辨率不同的图像。
- 检查标头或授权令牌，在将请求转发到源之前插入一个相应的标头并允许访问控制。
- 添加、删除和修改标头，然后重写 URL 路径，将用户定向到缓存中的不同对象。

- 生成新的 HTTP 响应，将未经身份验证的用户重定向到登录页面，直接从边缘创建和交付静态网页，或执行其他操作。有关更多信息，请参阅 Amazon CloudFront 开发人员指南 中的[使用 Lambda 函数生成对查看器和源请求的 HTTP 响应](#)。

有关使用 Lambda@Edge 的更多信息，请参阅[将 CloudFront 与 Lambda@Edge 结合使用](#)。

将 AWS Lambda 与 AWS CodeCommit 结合使用

您可以为 AWS CodeCommit 存储库创建触发器，以便存储库中的事件可以调用 Lambda 函数。例如，当创建分支或标签时，或者推送现有分支时，您可以调用 Lambda 函数。

Example AWS CodeCommit 消息事件

```
{  
    "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-a63741baddeb8",  
            "eventTriggerName": "index.handler",  
            "eventVersion": "1.0",  
            "userIdentityARN": "arn:aws:iam::123456789012:user/intern"  
        }  
    ]  
}
```

有关更多信息，请参阅[管理 AWS CodeCommit 存储库的触发器](#)。

将 AWS Lambda 与 AWS CodePipeline 结合使用

AWS CodePipeline 是一项使您能够为在 AWS 上运行的应用程序创建连续交付管道的服务。您可以创建管道来部署 Lambda 应用程序。您还可以将管道配置为在管道运行时调用 Lambda 函数以执行任务。在 Lambda 控制台中[创建 Lambda 应用程序 \(p. 120\)](#)时，Lambda 创建包括源、构建和部署阶段的管道。

CodePipeline 使用包含作业详细信息的事件异步调用您的函数。以下示例显示了调用名为 my-function 的函数的管道中的事件。

Example CodePipeline 事件

```
{  
    "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": "aws-us-west-2-123456789012-my-pipeline",
                    "objectKey": "my-pipeline/test-api-2/TdOSFRV"
                }
            }
        }
    ],
    "outputArtifacts": [
        {
            "name": "invokeOutput",
            "revision": null,
            "location": {
                "type": "S3",
                "s3Location": {
                    "bucketName": "aws-us-west-2-123456789012-my-pipeline",
                    "objectKey": "my-pipeline/invokeOutp/DOYHsJn"
                }
            }
        }
    ],
    "artifactCredentials": {
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "secretAccessKey": "6CGtmAa3lzWtV7a...",
        "sessionToken": "IQoJb3JpZ2luX2VjEA...",
        "expirationTime": 1575493418000
    }
}
}
```

要完成作业，函数必须调用 CodePipeline API 来指示成功或失败。以下示例 Node.js 函数使用 `PutJobSuccessResult` 操作来指示成功。它从事件对象获取 API 调用的作业 ID。

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()
}
```

对于异步调用，Lambda 将消息排队，并在函数返回错误时重试 (p. 98)。将函数配置为带有目标 (p. 85)，以保留函数无法处理的事件。

有关配置管道以调用 Lambda 函数的详细信息，请参阅 AWS CodePipeline 用户指南 中的[在管道中调用 AWS Lambda 函数](#)。

小节目录

- [权限 \(p. 178\)](#)
- [使用 AWS CodePipeline 为 Lambda 应用程序构建持续交付管道 \(p. 178\)](#)

权限

要调用函数，CodePipeline 管道需要具有使用以下 API 操作的权限：

- [ListFunctions \(p. 498\)](#)
- [InvokeFunction \(p. 482\)](#)

默认管道服务角色包括这些权限。

要完成作业，函数需要在其[执行角色 \(p. 30\)](#)中具有以下权限。

- `codepipeline:PutJobSuccessResult`
- `codepipeline:PutJobFailureResult`

这些权限包括在 [AWSCodePipelineCustomActionAccess](#) 托管策略中。

使用 AWS CodePipeline 为 Lambda 应用程序构建持续交付管道

您可以使用 AWS CodePipeline 为 Lambda 应用程序创建持续交付管道。CodePipeline 结合了源代码控制、构建和部署资源，以创建一个在您更改应用程序源代码时运行的管道。

在本教程中，您将创建以下资源。

- 存储库 – AWS CodeCommit 中的 Git 存储库。当您推送更改时，管道将源代码复制到 Amazon S3 存储桶并将其传递给构建项目。
- 构建项目 – 从管道获取源代码并打包应用程序的 AWS CodeBuild 构建。源包括构建规范，其中包含用于安装依赖项和为部署准备 AWS 无服务器应用程序模型 (AWS SAM) 模板的命令。
- 部署配置 – 管道的部署阶段定义一组操作，这些操作从构建输出获取 AWS SAM 模板，在 AWS CloudFormation 中创建更改集，并执行更改集以更新应用程序的 AWS CloudFormation 堆栈。
- AWS CloudFormation 堆栈 – 部署阶段使用模板在 AWS CloudFormation 中创建堆栈。模板是 YAML 格式的文档，用于定义 Lambda 应用程序的资源。应用程序包括一个 Lambda 函数和一个调用它的 Amazon API Gateway API。
- 角色 – 管道、构建和部署各有一个允许其管理 AWS 资源的服务角色。在创建这些资源时，控制台会创建管道和构建角色。您负责创建允许 AWS CloudFormation 管理应用程序堆栈的角色。

管道将存储库中的单个分支映射到单个 AWS CloudFormation 堆栈。您可以创建其他管道来为同一存储库中的其他分支添加环境。此外，您还可以向管道添加阶段，以进行测试、暂存和手动审批。有关 AWS CodePipeline 的更多信息，请参阅[什么是 AWS CodePipeline](#)。

对于使用 AWS 无服务器应用程序模型 和 AWS CloudFormation 创建管道的替代方法，请观看 Amazon Web Services YouTube 频道上的[自动执行您的无服务器应用程序部署](#)。

小节目录

- 先决条件 (p. 179)
- 创建 AWS CloudFormation 角色 (p. 179)
- 设置存储库 (p. 180)
- 创建管道 (p. 181)
- 更新构建阶段角色 (p. 182)
- 完成部署阶段 (p. 182)
- 测试应用程序 (p. 183)

先决条件

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以[安装 Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

在构建阶段，构建脚本将构件上传到 Amazon Simple Storage Service (Amazon S3)。您可以使用现有存储桶，也可以为管道创建新的存储桶。使用 AWS CLI 创建存储桶。

```
$ aws s3 mb s3://lambda-deployment-artifacts-123456789012
```

创建 AWS CloudFormation 角色

创建一个角色，以授予 AWS CloudFormation 访问 AWS 资源的权限。

创建 AWS CloudFormation 角色

1. 打开 IAM 控制台中的“[角色](#)”页面。
2. 选择 Create role (创建角色)。
3. 创建具有以下属性的角色。
 - Trusted entity (可信任的实体) – AWS CloudFormation
 - Permissions (权限) – AWSLambdaExecute
 - Role name (角色名称) – **cfn-lambda-pipeline**
4. 打开角色。在 Permissions 选项卡下，选择 Add inline policy。
5. 在创建策略中，选择 JSON 选项卡，然后添加以下策略。

```
{
  "Statement": [
    {
      "Action": [
        "apigateway:*",
        "codedeploy:*",
        "lambda:*",
        "cloudformation:CreateChangeSet",
        "cloudformation:DescribeChangeSets",
        "cloudformation:GetStack",
        "cloudformation:ListStacks",
        "cloudformation:UpdateStack"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

```
    "iam:GetRole",
    "iam:CreateRole",
    "iam:DeleteRole",
    "iam:PutRolePolicy",
    "iam:AttachRolePolicy",
    "iam:DeleteRolePolicy",
    "iam:DetachRolePolicy",
    "iam:PassRole",
    "s3:GetObject",
    "s3:GetObjectVersion",
    "s3:GetBucketVersioning"
],
"Resource": "*",
"Effect": "Allow"
}
],
"Version": "2012-10-17"
}
```

设置存储库

创建 AWS CodeCommit 存储库来存储项目文件。有关更多信息，请参阅 CodeCommit 用户指南中的[设置](#)。

创建存储库

1. 打开[开发人员工具控制台](#)。
2. 在 Source (源) 下，选择 Repositories (存储库)。
3. 选择 Create repository。
4. 按照说明创建和克隆名为 **lambda-pipeline-repo** 的存储库。

在存储库文件夹中创建以下文件。

Example index.js

一个返回当前时间的 Lambda 函数。

```
var time = require('time');
exports.handler = (event, context, callback) => {
  var currentTime = new time.Date();
  currentTime.setTz("America/Los_Angeles");
  callback(null, {
    statusCode: '200',
    body: 'The time in Los Angeles is: ' + currentTime.toString(),
  });
};
```

Example template.yml

定义应用程序的 [AWS SAM 模板 \(p. 26\)](#)。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Outputs the time
Resources:
  TimeFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
```

```
Runtime: nodejs10.x
CodeUri: ./
Events:
  MyTimeApi:
    Type: Api
    Properties:
      Path: /TimeResource
      Method: GET
```

Example buildspec.yml

安装所需的包并将部署包上传到 Amazon S3 的 [AWS CodeBuild 构建规范](#)。请将突出显示的文本替换为您的存储桶的名称。

```
version: 0.2
phases:
  install:
    runtime-versions:
      nodejs: 10
  build:
    commands:
      - npm install time
      - export BUCKET=lambda-deployment-artifacts-123456789012
      - aws cloudformation package --template-file template.yml --s3-bucket $BUCKET --
        output-template-file outputtemplate.yml
  artifacts:
    type: zip
    files:
      - template.yml
      - outputtemplate.yml
```

提交并将文件推送到 CodeCommit。

```
~/lambda-pipeline-repo$ git add .
~/lambda-pipeline-repo$ git commit -m "project files"
~/lambda-pipeline-repo$ git push
```

创建管道

创建一个部署应用程序的管道。管道监视存储库更改，运行 AWS CodeBuild 构建来创建部署包，并使用 AWS CloudFormation 部署应用程序。在创建管道的过程中，您还可以创建 AWS CodeBuild 构建项目。

创建管道

1. 打开[开发人员工具控制台](#)。
2. 在 Pipeline (管道) 下，选择 Pipelines (管道)。
3. 选择 Create pipeline (创建管道)。
4. 配置管道设置，然后选择 Next (下一步)。
 - Pipeline name (管道名称) – **lambda-pipeline**
 - Service role (服务角色) – New service role
 - Artifact store (构件存储) – 默认位置
5. 配置源阶段设置，然后选择 Next (下一步)。
 - Source provider (源提供商) – AWS CodeCommit
 - Repository name (存储库名称) – lambda-pipeline-repo
 - Branch name (分支名称) – master

- Change detection options (更改检测选项) – Amazon CloudWatch Events
6. 对于 Build provider (构建提供商) , 选择 AWS CodeBuild , 然后选择 Create project (创建项目)。
7. 配置构建项目设置 , 然后选择 Continue to CodePipeline (前往 CodePipeline)。
- Project name (项目名称) – **lambda-pipeline-build**
 - Operating system (操作系统) – Ubuntu
 - Runtime (运行时) – Standard (标准)
 - Runtime version (运行时版本) – aws/codebuild/standard:2.0
 - Image version (映像版本) – Latest (最新)
 - Buildspec name (构建规范名称) – **buildspec.yml**
8. 选择 Next (下一步)。
9. 配置部署阶段设置 , 然后选择 Next (下一步)。
- Deploy provider (部署提供商) – AWS CloudFormation
 - Action mode (操作模式) – Create or replace a change set (创建或替换更改集)
 - Stack name (堆栈名称) – lambda-pipeline-stack
 - Change set name (更改集名称) – lambda-pipeline-changeset
 - Template (模板) – **BuildArtifact::outputtemplate.yml**
 - 功能 – CAPABILITY_IAM、CAPABILITY_AUTO_EXPAND
 - Role name (角色名称) – cfn-lambda-pipeline
10. 选择 Create pipeline (创建管道)。

管道首次运行将失败 , 因为它需要其他权限。在下一部分中 , 您将向为构建阶段生成的角色添加权限。

更新构建阶段角色

在构建阶段 , AWS CodeBuild 需要将构建输出上传到 Amazon S3 存储桶的权限。

更新角色

1. 打开 IAM 控制台中的“[角色](#)”页面。
2. 选择 code-build-lambda-pipeline-service-role。
3. 选择 Attach policies (附加策略)。
4. 附加 AmazonS3FullAccess。

完成部署阶段

部署阶段有一个操作 - 为管理 Lambda 应用程序的 AWS CloudFormation 堆栈创建更改集。更改集指定对堆栈所做的更改 , 例如添加新资源和更新现有资源。通过更改集 , 您可以在更改前预览所做的更改 , 并添加审批阶段。添加第二个操作 - 执行更改集以完成部署。

更新部署阶段

1. 在[开发人员工具控制台](#)中打开您的管道。
 2. 选择 Edit (编辑)。
 3. 然后转到 Deploy (部署) , 选择 Edit stage (编辑阶段)。
 4. 选择 Add action group (添加操作组)。
 5. 配置部署阶段设置 , 然后选择 Next (下一步)。
- Action name (操作名称) – **execute-changeset**

- Action provider (操作提供程序) – AWS CloudFormation
 - Input artifacts (输入构件) – BuildArtifact
 - Action mode (操作模式) – Execute a change set (执行更改集)
 - Stack name (堆栈名称) – lambda-pipeline-stack
 - Change set name (更改集名称) – lambda-pipeline-changeset
6. 选择完成。
7. 选择 Save。
8. 选择 Release change (发布更改) 以运行管道。

您的管道已就绪。将更改推送到主分支即可触发部署。

测试应用程序

应用程序包括 API 网关 API 以及返回当前时间的公共终端节点。使用 Lambda 控制台查看应用程序和访问该 API。

测试应用程序

1. 打开 Lambda 控制台的“[应用程序](#)”页面。
2. 选择 lambda-pipeline-stack。
3. 在 Resources (资源) 下，展开 ServerlessRestApi。
4. 选择 Prod API endpoint (Prod API 终端节点)。
5. 将 `/TimeResource` 添加到 URL 的末尾。例如：`https://1193nqxdjj.execute-api.us-east-2.amazonaws.com/Prod/TimeResource`。
6. 打开 URL。

API 以以下格式返回当前时间。

```
The time in Los Angeles is: Thu Jun 27 2019 16:07:20 GMT-0700 (PDT)
```

将 AWS Lambda 与 Amazon Cognito 结合使用

利用 Amazon Cognito 事件功能，您可以运行 Lambda 函数以响应 Amazon Cognito 中的事件。例如，您可为同步触发事件调用 Lambda 函数，每当同步数据集时，就会发布该事件。要了解更多信息并完成一个示例，请参阅“移动开发”博客中的 [Amazon Cognito 事件简介：同步触发](#)。

Example Amazon Cognito 消息事件

```
{  
  "datasetName": "datasetName",  
  "eventType": "SyncTrigger",  
  "region": "us-east-1",  
  "identityId": "identityId",  
  "datasetRecords": {  
    "SampleKey2": {  
      "newValue": "newValue2",  
      "oldValue": "oldValue2",  
      "op": "replace"  
    },  
    "SampleKey1": {  
      "newValue": "newValue1",  
      "oldValue": "oldValue1",  
      "op": "add"  
    }  
  }  
}
```

```
        "oldValue": "oldValue1",
        "op": "replace"
    },
    "identityPoolId": "identityPoolId",
    "version": 2
}
```

可使用 Amazon Cognito 事件订阅配置来配置事件源映射。有关事件源映射和示例事件的信息，请参阅 Amazon Cognito 开发人员指南 中的 [Amazon Cognito 事件](#)。

将 AWS Lambda 与 AWS Config 结合使用

您可以使用 AWS Lambda 函数评估 AWS 资源配置是否遵从自定义配置规则。在创建、删除或更改资源时，AWS Config 将记录这些更改并将信息发送到您的 Lambda 函数。随后，您的 Lambda 函数将评估更改，并将结果报告给 AWS Config。然后，您可以使用 AWS Config 评估整体资源合规性：您可以了解哪些资源不合规以及哪些配置属性是导致不合规的原因。

Example AWS Config 消息事件

```
{
    "invokingEvent": "{\"configurationItem\":{\"configurationItemCaptureTime\": \"2016-02-17T01:36:34.043Z\", \"awsAccountId\": \"000000000000\", \"configurationItemStatus\": \"OK\", \"resourceId\": \"i-00000000\", \"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\"}
}
```

有关更多信息，请参阅 [使用 AWS Config 规则评估资源](#)。

将 AWS Lambda 与 Amazon DynamoDB 结合使用

您可以使用 AWS Lambda 函数来处理 [Amazon DynamoDB 流](#) 中的记录。使用 DynamoDB 流，每次更新 DynamoDB 表时，您都可以触发 Lambda 函数以执行额外的工作。

Lambda 从流中读取记录，并使用包含流记录的事件 [同步 \(p. 81\)](#) 调用您的函数。Lambda 以批量方式读取记录并调用您的函数来处理批次中的记录。

Example DynamoDB 流 记录事件

```
{
    "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"
            }
        },
        "StreamViewType": "NEW_AND_OLD_IMAGES"
},
    "awsRegion": "us-west-2",
    "eventName": "MODIFY",
    "eventSourceARN": sourcearn,
    "eventSource": "aws:dynamodb"
}

```

Lambda 将针对记录轮询 DynamoDB 流中的分片（按照每秒 4 次的基本频率）。当记录可用时，Lambda 调用您的函数并等待结果。如果处理成功，Lambda 将恢复轮询，直到它收到更多记录。

默认情况下，只要流中有记录，Lambda 就会调用您的函数。如果从流中读取的批处理中只有一条记录，则 Lambda 只向该函数发送一条记录。为避免在少量记录的情况下调用该函数，您可以通过配置批处理时段让

事件源缓冲记录，最多可达 5 分钟。在调用该函数之前，Lambda 会继续从流中读取记录，直到收集了完整批次，或者直到批处理时段到期。

如果您的函数返回一个错误，则 Lambda 将重试批处理，直到处理成功或数据过期。为避免分片停滞，可以将事件源映射配置为以较小的批处理大小重试，限制重试次数或者丢弃太早的记录。要保留丢弃的事件，可以配置事件源映射，以将有关失败批处理的详细信息发送到 SQS 队列或 SNS 主题。

您还可以通过并行处理每个分片的多个批处理来提高并发性。在每个分片中，Lambda 最多可以同时处理 10 个批处理。如果您增加每个分片的并发批处理数量，则 Lambda 仍然需要确保在分区键级别进行有序处理。

小节目录

- [执行角色权限 \(p. 186\)](#)
- [将流配置为事件源 \(p. 186\)](#)
- [事件源映射 API \(p. 187\)](#)
- [错误处理 \(p. 189\)](#)
- [Amazon CloudWatch 指标 \(p. 190\)](#)
- [教程：将 AWS Lambda 与 Amazon DynamoDB 流结合使用 \(p. 190\)](#)
- [示例函数代码 \(p. 194\)](#)
- [DynamoDB 应用程序的 AWS SAM 模板 \(p. 197\)](#)

执行角色权限

Lambda 需要以下权限才能管理与您的 DynamoDB 流相关的资源。将这些权限添加到您的函数的执行角色中。

- [dynamodb:DescribeStream](#)
- [dynamodb:GetRecords](#)
- [dynamodb:GetShardIterator](#)
- [dynamodb>ListStreams](#)

`AWSLambdaDynamoDBExecutionRole` 托管策略包含这些权限。有关更多信息，请参阅 [AWS Lambda 执行角色 \(p. 30\)](#)。

要将失败批处理的记录发送到队列或主题，您的函数需要其他权限。每项目标服务均需要不同的权限，如下所示：

- Amazon SQS – [sns:SendMessage](#)
- Amazon SNS – [sns:Publish](#)

将流配置为事件源

创建事件源映射以指示 Lambda 将流中的记录发送到 Lambda 函数。您可以创建多个事件源映射，以使用多个 Lambda 函数处理相同的数据，或使用单个函数处理来自多个流的项目。

要在 Lambda 控制台中将您的函数配置为从 DynamoDB 流 读取，请创建 DynamoDB 触发器。

创建触发器

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。

3. 在 Designer 下方，选择 Add trigger (添加触发器)。
4. 选择触发器类型。
5. 配置所需选项，然后选择 Add (添加)。

Lambda 支持 DynamoDB 事件源的以下选项。

事件源选项

- DynamoDB 表 – 要从中读取记录的 DynamoDB 表。
- Batch size (批处理大小) – 每个批处理中发送到函数的记录的数量 (最多 1000 条)。Lambda 通过单个调用将批处理中的所有记录传递给函数，前提是事件的总大小未超出同步调用的 [负载限制 \(p. 27\)](#) (6 MB)。
- 批处理时段 – 指定在调用函数之前收集记录的最长时间 (以秒为单位)。
- 起始位置 – 仅处理新记录或所有现有记录。
 - 最新 – 处理已添加到流中的新记录。
 - 时间范围 – 处理流中的所有记录。

在处理任何现有记录后，函数将继续处理新记录。

- On-failure destination (故障目标) – SQS 队列或 SNS 主题，用于无法处理的记录。当 Lambda 由于时间太远或已用尽所有重试而丢弃一批记录时，它将有关该批处理的详细信息发送到队列或主题。
- Retry attempts (重试) – 函数返回错误时 Lambda 重试的最大次数。这不适用于批处理未到达该函数的服务错误或限制。
 - Maximum age of record (最长记录期限) – Lambda 发送到函数的记录的最长期限。
 - Split batch on error (出错时拆分批处理) – 当函数返回错误时，请在重试之前将批处理拆分为两部分。
 - Concurrent batches per shard (每个分片的并发批处理) – 并发处理来自同一分片的多个批处理。
 - 已启用 – 禁用事件源可停止处理记录。Lambda 将跟踪已处理的最后一条记录，并在重新启用映射后从停止位置重新开始处理。

之后，要管理事件源配置，请在设计器中选择触发器。

事件源映射 API

要使用 AWS CLI 或 AWS 开发工具包管理事件源映射，请使用以下 API 操作：

- [CreateEventSourceMapping \(p. 415\)](#)
- [ListEventSourceMappings \(p. 492\)](#)
- [GetEventSourceMapping \(p. 451\)](#)
- [UpdateEventSourceMapping \(p. 547\)](#)
- [DeleteEventSourceMapping \(p. 432\)](#)

以下示例使用 AWS CLI 将名为 my-function 的函数映射到由 Amazon 资源名称 (ARN) 指定的 DynamoDB 流 (批处理大小为 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
{
    "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  
}
```

配置其他选项，以自定义如何处理批处理，并指定何时丢弃无法处理的记录。以下示例更新事件源映射，以在两次重试之后或者如果失败记录已存在一个小时以上，将失败记录发送到 SQS 队列。

```
$ 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"}}'  
{  
    "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  
}
```

更新的设置是异步应用的，并且直到该过程完成才反映在输出中。使用 `get-event-source-mapping` 命令可查看当前状态。

```
$ aws lambda get-event-source-mapping --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b  
{  
    "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
```

}

要同时处理多个批处理，请使用 `--parallelization-factor` 选项。

```
$ aws lambda update-event-source-mapping --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b11284 \
--parallelization-factor 5
```

错误处理

从 DynamoDB 流中读取记录的事件源映射将同步调用函数并在出错时重试。如果函数受到限制，或者 Lambda 服务未调用该函数而返回错误，Lambda 将重试，直到记录到期或者超过您在事件源映射上配置的最长期限。

如果函数接收到记录但返回错误，Lambda 将重试，直到批处理中的记录到期、超过最大使用期限或者达到配置的重试限制。对于函数错误，您还可以配置事件源映射，以将失败的批处理拆分为两个批处理。重试较小的批处理可以隔离不良记录并解决超时问题。拆分批处理不计入重试限制。

如果错误处理措施失败，Lambda 将丢弃记录并继续处理流中的批处理。使用默认设置，这意味着不良记录最多可以将针对受影响分片的处理操作阻止 one day。为避免这种情况，请以合理的重试次数和适合您使用案例的最长记录期限来配置函数的事件源映射。

要保留废弃批处理的记录，请配置失败事件目标。Lambda 将文档和有关批处理的详细信息发送到目标队列或主题。

配置失败事件记录的目标

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Designer (设计器) 下，选择 Add destination (添加目标)。
4. 对于 Source (源)，选择 Stream invocation (流调用)。
5. 对于 Stream (流)，选择映射到函数的流。
6. 对于 Destination type (目标类型)，请选择接收调用记录的资源类型。
7. 对于 Destination (目标)，请选择一个资源。
8. 选择保存。

以下示例显示了 DynamoDB 流的调用记录。

Example 调用记录

```
{
    "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": "80000000003126276362",
        "endSequenceNumber": "80000000003126276362"
    }
}
```

```
        "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"
    }
}
```

您可以使用此信息从流中检索受影响的记录以进行故障排除。实际的记录不包括在内，因此您必须处理这些记录，并在它们到期并丢失之前从流中检索它们。

Amazon CloudWatch 指标

在您的函数处理完一批记录后，Lambda 将发出 `IteratorAge` 指标。该指标指示处理完成时批处理中最后一条记录的时间。如果您的函数正在处理新事件，则可使用迭代器期限来估算新记录的添加时间与函数处理新记录的时间之间的延迟。

迭代器期限中的上升趋势可以指示您的函数问题。有关更多信息，请参阅[使用 AWS Lambda 函数指标 \(p. 368\)](#)。

教程：将 AWS Lambda 与 Amazon DynamoDB 流结合使用

在本教程中，您将创建一个 Lambda 函数来处理来自 Amazon DynamoDB 流的事件。

先决条件

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以[安装 Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

创建执行角色

创建[执行角色 \(p. 30\)](#)，向您的函数授予访问 AWS 资源的权限。

创建执行角色

1. 打开 IAM 控制台中的“[角色](#)”页面。
2. 选择 `Create role` (创建角色)。
3. 创建具有以下属性的角色。
 - Trusted entity (可信任的实体) – Lambda.
 - 权限 – `AWSLambdaDynamoDBExecutionRole`.

- 角色名称 (角色名称) – **lambda-dynamodb-role**。

AWSLambdaDynamoDBExecutionRole 具有该函数从 DynamoDB 中读取项目并将日志写入 CloudWatch Logs 所需的权限。

创建函数

以下示例代码接收 DynamoDB 事件输入并对其所包含的消息进行处理。为了展示这个过程，代码会将一些传入的事件数据写入 CloudWatch Logs。

Note

有关使用其他语言的示例代码，请参阅 [示例函数代码 \(p. 194\)](#)。

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");
};
```

创建函数

1. 将示例代码复制到名为 `index.js` 的文件中。
2. 创建部署程序包。

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

3. 使用 `create-function` 命令创建 Lambda 函数。

```
$ 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
```

测试 Lambda 函数。

在本步骤中，您将使用 `invoke` AWS Lambda CLI 命令和以下示例 DynamoDB 事件手动调用您的 Lambda 函数。

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"  
}  
]  
}
```

执行下面的 `invoke` 命令。

```
$ aws lambda invoke --function-name ProcessDynamoDBRecords --payload file://input.txt  
outputfile.txt
```

函数在响应正文中返回字符串 `message`。

在 `outputfile.txt` 文件中验证输出。

创建一个 DynamoDB 表 (启用流)

创建一个启用了 Amazon DynamoDB 表。

创建 DynamoDB 表

1. 打开 [DynamoDB 控制台](#)。
2. 选择 Create Table。
3. 使用以下设置创建表。
 - 表名称 – **lambda-dynamodb-stream**
 - 主键 – **id** (字符串)
4. 选择 Create。

启用流

1. 打开 [DynamoDB 控制台](#)。
2. 选择表。
3. 选择 `lambda-dynamodb-stream` 表。
4. 在 Overview (概述) 选项卡下，选择 Manage stream (管理流)。
5. 选择 Enable。

记下流 ARN。在下一步中将该流与您的 Lambda 函数关联时，您将需要此类信息。有关启用流的更多信息，请参阅[使用 DynamoDB 流捕获表活动](#)。

在 AWS Lambda 中添加事件源。

在 AWS Lambda 中创建事件源映射。此事件源映射将 DynamoDB 流与您的 Lambda 函数关联。创建此事件源映射后，AWS Lambda 即开始轮询该流。

运行以下 AWS CLI `create-event-source-mapping` 命令。命令执行后，记下 UUID。在任何命令中，如删除事件源映射时，您都需要该 UUID 来引用事件源映射。

```
$ aws lambda create-event-source-mapping --function-name ProcessDynamoDBRecords \
--batch-size 100 --starting-position LATEST --event-source DynamoDB-stream-arn
```

这会在指定的 DynamoDB 流和 Lambda 函数之间创建映射。您可将一个 DynamoDB 流关联到多个 Lambda 函数，也可将同一个 Lambda 函数关联到多个流。但是，Lambda 函数将共享它们共享的流的读取吞吐量。

您可以通过运行以下命令获取事件源映射的列表。

```
$ aws lambda list-event-source-mappings
```

该列表返回您创建的所有事件源映射，而对于每个映射，它都显示 `LastProcessingResult` 等信息。该字段用于在出现任何问题时提供信息性消息。`No records processed`（指示 AWS Lambda 未开始轮询或流中没有任何记录）和 `OK`（指示 AWS Lambda 已成功读取流中的记录并调用了您的 Lambda 函数）等值表示未出现任何问题。如果出现问题，您将收到一条错误消息。

如果您有大量事件源映射，请使用函数名称参数缩窄结果范围。

```
$ aws lambda list-event-source-mappings --function-name ProcessDynamoDBRecords
```

测试设置

测试端到端体验。执行表更新时，DynamoDB 会将事件记录写入流。AWS Lambda 轮询该流时，它将在流中检测新记录并通过向该函数传递事件来代表您执行 Lambda 函数。

1. 在 DynamoDB 控制台中，在表中添加、更新和删除项目。DynamoDB 会将这些操作的记录写入流。
2. AWS Lambda 轮询该流，当检测到流有更新时，它会通过传递在流中发现的事件数据来调用您的 Lambda 函数。
3. 您的函数将执行并在 Amazon CloudWatch 中创建日志。您可以验证 Amazon CloudWatch 控制台中报告的日志。

示例函数代码

示例代码具有以下语言。

主题

- [Node.js \(p. 194\)](#)
- [Java 11 \(p. 195\)](#)
- [C# \(p. 195\)](#)
- [Python 3 \(p. 196\)](#)
- [Go \(p. 197\)](#)

Node.js

以下示例处理来自 DynamoDB 的消息并记录其内容。

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");
};
```

压缩示例代码以创建部署程序包。有关说明，请参阅[Node.js 中的 AWS Lambda 部署程序包 \(p. 268\)](#)。

Java 11

以下示例处理来自 DynamoDB 的消息，并记录其内容。`handleRequest` 是 AWS Lambda 调用并提供事件数据的处理程序。该处理程序使用了预定义的 `DynamodbEvent` 类（在 `aws-lambda-java-events` 库中定义）。

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.";
    }
}
```

如果该处理程序正常返回并且没有异常，则 Lambda 认为输入的记录批次得到成功处理并开始读取流中的新记录。如果该处理程序引发异常，则 Lambda 认为输入的记录批次未得到处理，并用相同的记录批次再次调用该函数。

附属物

- `aws-lambda-java-core`
- `aws-lambda-java-events`

使用 Lambda 库依赖项构建代码以创建部署程序包。有关说明，请参阅[Java 中的 AWS Lambda 部署程序包 \(p. 302\)](#)。

C#

以下示例处理来自 DynamoDB 的消息，并记录其内容。`ProcessDynamoEvent` 是 AWS Lambda 调用并提供事件数据的处理程序。该处理程序使用了预定义的 `DynamoDbEvent` 类（在 `Amazon.Lambda.DynamoDBEvents` 库中定义）。

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());
            }
        }
    }
}
```

使用以上示例替换.NET Core 中的 Program.cs 有关说明，请参阅[C# 中的 AWS Lambda 部署程序包 \(p. 342\)](#)。

Python 3

以下示例处理来自 DynamoDB 的消息并记录其内容。

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'])))
```

压缩示例代码以创建部署程序包。有关说明，请参阅[Python 中的 AWS Lambda 部署程序包 \(p. 279\)](#)。

Go

以下示例处理来自 DynamoDB 的消息并记录其内容。

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())
            }
        }
    }
}
```

压缩示例代码以创建部署程序包。有关说明，请参阅[Python 中的 AWS Lambda 部署程序包 \(p. 279\)](#)。

DynamoDB 应用程序的 AWS SAM 模板

您可以使用 [AWS SAM](#) 构建此应用程序。要了解有关创建 AWS SAM 模板的更多信息，请参阅 AWS 无服务器应用程序模型 开发人员指南 中的 [AWS SAM 模板基础知识](#)。

下面是[教程应用程序 \(p. 190\)](#)的示例 AWS SAM 模板。将以下文本复制到 .yaml 文件中，并将其保存到您之前创建的 ZIP 程序包旁。请注意，Handler 和 Runtime 参数值应与上一节中创建函数时所用的参数值匹配。

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
```

有关如何使用程序包和部署命令打包和部署无服务器应用程序的信息，请参阅 AWS 无服务器应用程序模型开发人员指南 中的[部署无服务器应用程序](#)。

教程：配置 Lambda 函数以访问 Amazon VPC 中的 Amazon ElastiCache

在本教程中，您将执行以下操作：

- 在您的默认 Amazon Virtual Private Cloud 中创建 Amazon ElastiCache 集群。有关 Amazon ElastiCache 的更多信息，请参阅[Amazon ElastiCache](#)。
- 创建 Lambda 函数以访问 ElastiCache 集群。在创建 Lambda 函数时，您需要提供 Amazon VPC 和 VPC 安全组中的子网 ID，以允许 Lambda 函数访问您的 VPC 中的资源。在本教程的图示中，Lambda 函数生成 UUID，将它写入到缓存，然后再从缓存中检索。
- 调用 Lambda 函数，并确保它访问了您的 VPC 中的 ElastiCache 集群。

有关将 Lambda 与 Amazon VPC 结合使用的详细信息，请参阅[配置 Lambda 函数以访问 VPC 中的资源 \(p. 72\)](#)。

先决条件

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command  
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以[安装 Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

创建执行角色

创建[执行角色 \(p. 30\)](#)，向您的函数授予访问 AWS 资源的权限。

创建执行角色

1. 打开 IAM 控制台中的[“角色”页面](#)。

2. 选择 Create role (创建角色)。
3. 创建具有以下属性的角色。
 - Trusted entity (可信任的实体) – Lambda.
 - 权限 – AWSLambdaVPCAccessExecutionRole.
 - 角色名称 (角色名称) – **lambda-vpc-role**.

AWSLambdaVPCAccessExecutionRole 具有函数管理与 VPC 的网络连接所需的权限。

创建 ElastiCache 集群

在您的默认 VPC 中创建 ElastiCache 集群。

1. 运行以下 AWS CLI 命令以创建 Memcached 集群。

```
$ aws elasticache create-cache-cluster --cache-cluster-id ClusterForLambdaTest \
--cache-node-type cache.m3.medium --engine memcached --num-cache-nodes 1 \
--security-group-ids sg-0897d5f549934c2fb
```

您可以在 VPC 控制台的 Security Groups 下查找默认 VPC 安全组。您的示例 Lambda 函数将在该集群中添加和检索项目。

2. 记下您启动的缓存集群的配置终端节点。您可通过 Amazon ElastiCache 控制台获取这一信息。在下一部分中，您将在自己的 Lambda 函数中指定此值。

创建部署程序包

以下 Python 代码示例在 ElastiCache 集群中读取和写入项目。

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")
```

附属物

- [pymemcache](#) – Lambda 函数代码使用此库创建 `HashClient` 对象，以便在内存缓存中设置和获取项目。
- [elasticache-auto-discovery](#) – Lambda 函数使用此库来获取 Amazon ElastiCache 集群中的节点。

安装 Pip 的依赖项并创建部署程序包。有关说明，请参阅[Python 中的 AWS Lambda 部署程序包 \(p. 279\)](#)。

创建 Lambda 函数

使用 `create-function` 命令创建 Lambda 函数。

```
$ 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
```

您可以从 VPC 控制台查找子网 ID 以及您的 VPC 的默认安全组 ID。

测试 Lambda 函数。

在此步骤中，您将使用 `invoke` 命令手动调用 Lambda 函数。当 Lambda 函数执行时，它会生成 UUID，并将它写入到在您的 Lambda 代码中指定的 ElastiCache 集群。然后，Lambda 函数将从缓存中检索项目。

1. 使用 `invoke` 命令调用 Lambda 函数。

```
$ aws lambda invoke --function-name AccessMemCache output.txt
```

2. 按以下过程验证 Lambda 函数是否已成功执行：

- 查看 `output.txt` 文件。
- 在 AWS Lambda 控制台中查看结果。
- 在 CloudWatch Logs 中验证结果。

您已经创建了 Lambda 函数来访问您的 VPC 中的 ElastiCache 集群，现在您可以调用该函数来响应事件。有关配置事件源和示例的信息，请参阅[将 AWS Lambda 与其他服务结合使用 \(p. 138\)](#)。

将 AWS Lambda 和 Amazon EC2 结合使用

您可以使用 AWS Lambda 处理来自 Amazon Elastic Compute Cloud 的生命周期事件并管理 Amazon EC2 资源。Amazon EC2 针对生命周期事件（例如，在实例更改状态时、在 Amazon Elastic Block Store 卷快照

完成时或在计划终止 Spot 实例时) 向 Amazon CloudWatch Events 发送事件。配置 CloudWatch Events 以将这些事件转发到 Lambda 函数来进行处理。

CloudWatch Events 通过来自 Amazon EC2 的事件文档异步调用 Lambda 函数。

Example 实例生命周期事件

```
{  
    "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"  
    }  
}
```

有关在 CloudWatch Events 中配置事件的详细信息 , 请参阅[配合使用 AWS Lambda 和 Amazon CloudWatch Events \(p. 167\)](#)。有关处理 Amazon EBS 快照通知的示例函数 , 请参阅 Amazon EC2 用户指南 (适用于 Linux 实例) 中的[适用于 Amazon EBS 的 Amazon CloudWatch Events](#)。

您还可以使用 AWS 开发工具包通过 Amazon EC2 API 管理实例和其他资源。有关用 C# 编写的示例应用程序的教程 , 请参阅[教程 : 使用适用于 .NET 的 AWS 开发工具包管理 Amazon EC2 Spot 实例 \(p. 202\)](#)。

权限

要处理来自 Amazon EC2 的生命周期事件 , CloudWatch Events 需要有权调用您的函数。此权限来自函数的[基于资源的策略 \(p. 33\)](#)。如果您使用 CloudWatch Events 控制台配置事件触发器 , 则该控制台将代表您更新基于资源的策略。否则 , 请添加如下所示的语句 :

Example Amazon EC2 生命周期通知的基于资源的策略语句

```
{  
    "Sid": "ec2-events",  
    "Effect": "Allow",  
    "Principal": {  
        "Service": "events.amazonaws.com"  
    },  
    "Action": "lambda:InvokeFunction",  
    "Resource": "arn:aws:lambda:us-east-2:12456789012:function:my-function",  
    "Condition": {  
        "ArnLike": {  
            "AWS:SourceArn": "arn:aws:events:us-east-2:12456789012:rule/*"  
        }  
    }  
}
```

要添加语句 , 请使用 add-permission AWS CLI 命令。

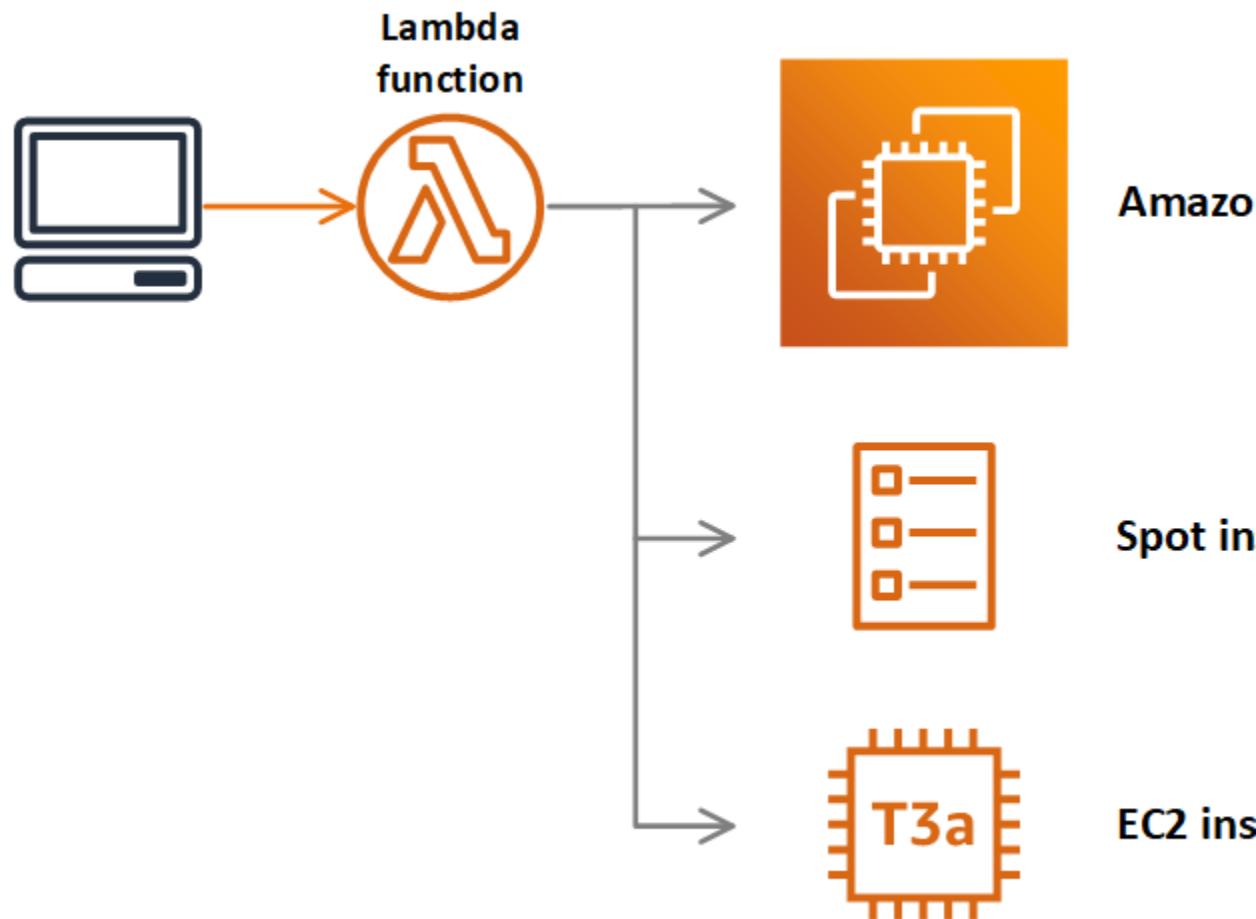
```
aws lambda add-permission --action lambda:InvokeFunction --statement-id ec2-events \  
--principal events.amazonaws.com --function-name my-function --source-arn  
'arn:aws:events:us-east-2:12456789012:rule/*'
```

如果您的函数使用 AWS 开发工具包来管理 Amazon EC2 资源，请向函数的[执行角色 \(p. 30\)](#)添加 Amazon EC2 权限。

教程：使用适用于 .NET 的 AWS 开发工具包 管理 Amazon EC2 Spot 实例

您可以使用 适用于 .NET 的 AWS 开发工具包 通过 C# 代码管理 Amazon EC2 Spot 实例。利用开发工具包，您可以使用 Amazon EC2 API 创建 Spot 实例请求、确定满足请求的时间、删除请求以及标识创建的实例。

本教程提供执行这些任务的代码以及可本地或在 AWS 上运行的示例应用程序。它包含一个示例项目，可将该项目部署到 AWS Lambda 的 .NET Core 2.1 运行时。



有关 Spot 实例使用情况和最佳实践的更多信息，请参阅《Amazon EC2 用户指南》中的 [Spot 实例](#)。

先决条件

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以 [安装 Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

本教程使用开发人员指南的 GitHub 存储库中的代码。该存储库还包含执行其过程所需的帮助程序脚本和配置文件。克隆 github.com/awsdocs/aws-lambda-developer-guide 上的存储库。

要使用示例代码，您需要以下工具：

- AWS CLI – 要将示例应用程序部署到 AWS，请安装 [AWS CLI](#)。在本地运行示例代码时，AWS CLI 还会向该代码提供凭证。
- .NET Core CLI – 要本地运行和测试代码，请安装 [.NET Core 开发工具包 2.1](#)。
- Lambda .NET Core Global Tool – 要为 Lambda 构建部署程序包，请安装带 .NET Core CLI 的 [.NET Core Global Tool](#)。

```
$ dotnet tool install -g Amazon.Lambda.Tools
```

本教程中的代码可管理启动 Amazon EC2 实例的 Spot 请求。要本地运行代码，您需要具有开发工具包凭证并有权使用以下 API。

- `ec2:RequestSpotInstance`
- `ec2:GetSpotRequestState`
- `ec2:CancelSpotRequest`
- `ec2:TerminateInstances`

要在 AWS 中运行示例应用程序，您需要[有权使用 Lambda \(p. 30\)](#) 和以下服务。

- [AWS CloudFormation \(定价 \)](#)
- [Amazon Elastic Compute Cloud \(定价 \)](#)

标准费用适用于每项服务。

查看代码

在 [sample-apps/ec2-spot](#) 下的指南存储库中查找示例项目。此目录包含 Lambda 函数代码、测试、项目文件、脚本和 AWS CloudFormation 模板。

Function 类包含一个 FunctionHandler 方法，该方法调用其他方法来创建 Spot 请求、检查其状态并进行清理。它在静态构造函数中使用 [适用于 .NET 的 AWS 开发工具包](#) 中创建 Amazon EC2 客户端，以便能够在整个类中使用它。

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: aws.amazon.com/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].SpotInstanceId;
```

RequestSpotInstance 方法可创建 Spot 实例请求。

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;
}
...
```

接下来，您需要等待直至 Spot 请求进入 Active 状态，然后继续到最后一步。要确定 Spot 请求的状态，请使用 [DescribeSpotInstanceRequests](#) 方法来获取要监视的 Spot 请求 ID 的状态。

```
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];
}
```

最后一步是清除您的请求和实例。重要的是，要取消所有未完成的请求并终止所有实例。只取消请求不会终止您的实例，这意味着您需要继续为它们支付费用。如果您终止了实例，那么 Spot 请求可能会被取消，但在某些情况下，例如，如果您使用的是持久请求，那么只终止实例是不足以停止请求的，以至于这些请求会重新执行。因此，最好的做法是取消所有活跃的请求并终止所有正在运行的实例。

您使用 [CancelSpotInstanceRequests](#) 方法来取消 Spot 请求。以下示例演示了如何取消 Spot 请求。

```
public async Task CancelSpotRequest(string spotRequestId)
{
    Console.WriteLine("Canceling request " + spotRequestId);
    var cancelRequest = new CancelSpotInstanceRequestsRequest();
    cancelRequest.SpotInstanceRequestIds.Add(spotRequestId);

    await ec2Client.CancelSpotInstanceRequestsAsync(cancelRequest);
}
```

您可使用 [TerminateInstances](#) 方法终止实例。

```
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;
        }
    }
}
```

本地运行代码

在本地计算机上运行代码以创建 Spot 实例请求。满足请求后，代码将删除请求并终止实例。

运行应用程序代码

1. 导航到 `ec2Spot.Tests` 目录。

```
$ cd test/ec2Spot.Tests
```

2. 使用 .NET CLI 运行项目的单元测试。

```
test/ec2Spot.Tests$ dotnet test
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
```

单元测试调用 `FunctionHandler` 方法来创建 Spot 实例请求、监控它并进行清理。它是在 [xUnit.net](#) 测试框架中实现的。

部署应用程序

在 Lambda 中运行代码作为创建无服务器应用程序的起点。

部署和测试应用程序

1. 将您的区域设置为 `us-east-2`。

```
$ export AWS_DEFAULT_REGION=us-east-2
```

2. 为部署构件创建存储桶。

```
$ ./create-bucket.sh
make_bucket: lambda-artifacts-63d5cbbf18fa5ecc
```

3. 创建部署程序包并部署应用程序。

```
$ ./deploy.sh
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 ebd38e401cedd7d676d05d22b76f0209 1305107 / 1305107.0 (100.00%)
Successfully packaged artifacts and wrote output template to file out.yaml.
Execute 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. 打开 Lambda 控制台的“[Applications](#)”(应用程序) 页面。

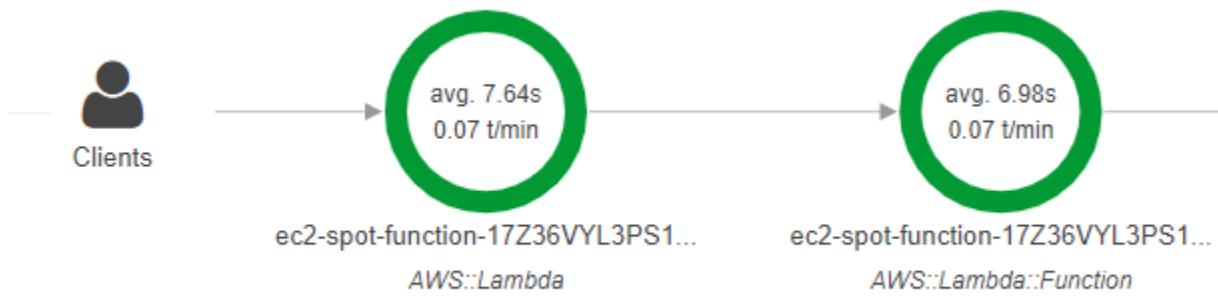
The screenshot shows the AWS Lambda console for the 'ec2-spot' function. The 'Overview' tab is active. A 'Getting started' section is present. The 'Resources' section displays two entries:

Logical ID	Physical ID	Type
function	ec2-spot-function-17Z36VYL3PS14	Lambda Function
role	ec2-spot-role-1TDCWJ2ZNNF1M	IAM Role

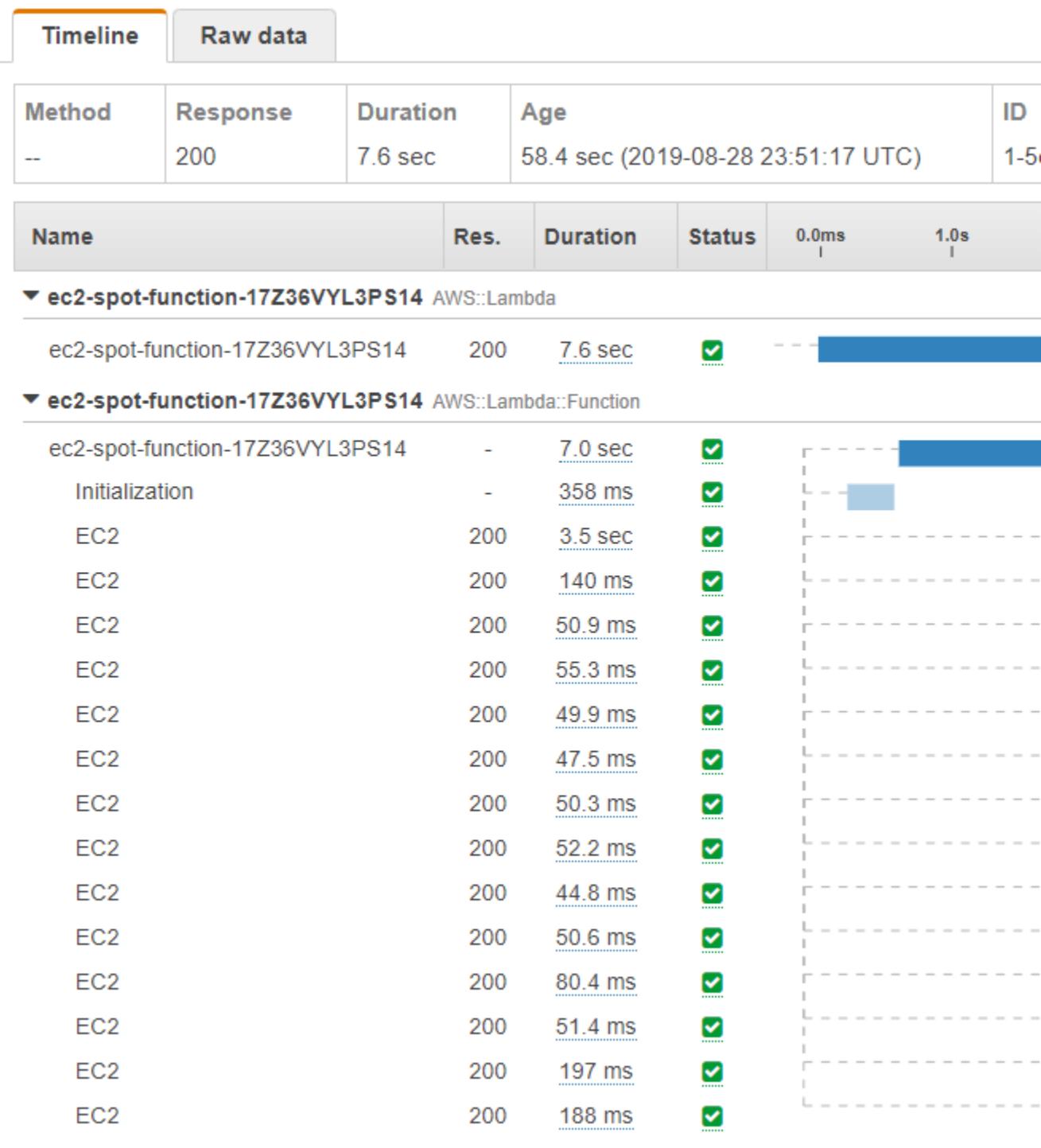
5. 在 Resources (资源) 下，选择 function (函数)。
6. 选择 Test (测试) 并从默认模板创建测试事件。
7. 再次选择 Test (测试) 以调用该函数。

查看日志和追踪信息以了解 Spot 请求 ID 以及对 Amazon EC2 的调用的顺序。

要查看服务映射，请打开 X-Ray 控制台中的“Service map (服务映射)”页面。



在服务映射中选择一个节点，然后选择 View traces (查看跟踪) 以查看跟踪列表。从列表中选择跟踪以查看函数对 Amazon EC2 进行的调用的时间线。



清除

本教程中提供的代码旨在创建和删除 Spot 实例请求，并终止它们启动的实例。但是，如果出现错误，可能不会自动清理请求和实例。在 Amazon EC2 控制台中查看 Spot 请求和实例。

确认已清理 Amazon EC2 资源

1. 在 Amazon EC2 控制台中打开“[Spot Requests \(Spot 请求\)](#)”页面。
2. 验证请求的状态是否为 Cancelled (已取消)。
3. 在 Capacity (容量) 列中选择实例 ID 以查看实例。
4. 验证实例的状态是 Terminated (已终止) 还是 Shutting down (正在关闭)。

要清理示例函数和支持资源，请删除其 AWS CloudFormation 堆栈以及您创建的构件存储桶。

```
$ ./cleanup.sh
Delete deployment artifacts and bucket (lambda-artifacts-63d5cbbf18fa5ecc)?y
delete: s3://lambda-artifacts-63d5cbbf18fa5ecc/ebd38e401cedd7d676d05d22b76f0209
remove_bucket: lambda-artifacts-63d5cbbf18fa5ecc
```

不会自动删除函数的日志组。您可以在 [CloudWatch Logs](#) 控制台中删除它。X-Ray 中的跟踪在几周后过期，并且会被自动删除。

将 AWS Lambda 与 AWS IoT 结合使用

AWS IoT 提供连接 Internet 的设备（如传感器）与 AWS 云之间的安全通信。这让您能够从多个设备收集、存储和分析遥测数据。

您可以为设备创建 AWS IoT 规则，以便与 AWS 服务进行交互。AWS IoT [规则引擎](#)提供基于 SQL 的语言，用于从消息负载中选择数据，并将数据发送到其他服务，如 Amazon S3、Amazon DynamoDB 和 AWS Lambda。当您想要调用其他 AWS 服务或第三方服务时，您可以定义规则以调用 Lambda 函数。

当传入的 IoT 消息触发规则时，AWS IoT [异步 \(p. 83\)](#)调用 Lambda 函数并将数据从 IoT 消息传递到函数。

以下示例显示了温室传感器的湿度读数。row 和 pos 值标识传感器的位置。此示例事件基于 [AWS IoT 规则教程](#)中的温室类型。

Example 示例 AWS IoT 消息事件

```
{
  "row" : "10",
  "pos" : "23",
  "moisture" : "75"
}
```

对于异步调用，Lambda 将消息排队，并在函数返回错误时[重试 \(p. 98\)](#)。将函数配置为带有[目标 \(p. 85\)](#)，以保留函数无法处理的事件。

您需要向 AWS IoT 服务授予调用 Lambda 函数的权限。使用 add-permission 命令将权限语句添加到函数的基于资源的策略。

```
$ aws lambda add-permission --function-name my-function \
--statement-id iot-events --action "lambda:InvokeFunction" principal
ioevents.amazonaws.com
{
  "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\""
}
```

有关如何将 Lambda 与 AWS IoT 结合使用的更多信息，请参阅[创建 AWS Lambda 规则](#)。

将 AWS Lambda 和 AWS IoT Events 结合使用

AWS IoT Events 监控来自多个 IoT 传感器和应用程序的输入以识别事件模式。然后，它会在事件发生时采取适当的操作。AWS IoT Events 从多个源接收其输入作为 JSON 负载。AWS IoT Events 支持简单事件（其中，每个输入触发一个事件）和复杂事件（其中，必须发生多个输入才能触发事件）。

要使用 AWS IoT Events，请定义一个检测器模型，该模型是设备或过程的状态机模型。除了状态之外，您还可以定义模型的输入和事件。您还可以定义在事件发生时要执行的操作。在需要调用其他 AWS 服务（如 Amazon Connect）或在外部应用程序（如企业资源规划（ERP）应用程序）中执行操作时，请对操作使用 Lambda 函数。

当事件发生时，AWS IoT Events 将异步调用您的 Lambda 函数。它提供有关检测器模型和触发操作的事件的信息。以下示例消息事件基于 AWS IoT Events [简单分步示例](#)中的定义。

Example 示例 AWS IoT Events 消息事件

```
{  
  "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": {}  
      }  
    }  
  }  
}
```

传递到 Lambda 函数的事件包括以下字段：

- **eventName** – 检测器模型中此事件的名称。
- **eventTime** – 事件的发生时间。
- **detector** – 检测器模型的名称和版本。
- **eventTriggerDetails** – 触发了事件的输入的描述。
- **actionExecutionId** – 操作的唯一执行标识符。
- **state** – 事件发生时检测器模型的状态。
 - **stateName** – 检测器模型中的状态名称。
 - **timers** – 在此状态下设置的任何计时器。

- variables – 在此状态下设置的任何变量值。

您需要向 AWS IoT Events 服务授予调用 Lambda 函数的权限。使用 add-permission 命令将权限语句添加到函数的基于资源的策略。

```
$ aws lambda add-permission --function-name my-function \
--statement-id iot-events --action "lambda:InvokeFunction" principal
ioevents.amazonaws.com
{
    "Statement": "{\"Sid\":\"iot-events\",\"Effect\":\"Allow\",\"Principal\":{\"Service\"
\":\"ioevents.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\"
:\"arn:aws:lambda:us-west-2:123456789012:function:my-function\"}"
}
```

有关将 Lambda 和 AWS IoT Events 结合使用的更多信息，请参阅[将 AWS IoT Events 和其他服务结合使用](#)。

将 AWS Lambda 与 Amazon Kinesis 结合使用

您可以使用 AWS Lambda 函数来处理 [Amazon Kinesis 数据流](#)中的记录。借助 Kinesis，您可以从多个源收集数据并与多个使用者一起处理这些数据。Lambda 支持标准数据流迭代器和 HTTP/2 流使用者。

Lambda 从数据流中读取记录，并使用包含流记录的事件[同步 \(p. 81\)](#)调用您的函数。Lambda 以批量方式读取记录并调用您的函数来处理批次中的记录。

Example Kinesis 记录事件

```
{
    "Records": [
        {
            "kinesis": {
                "kinesisSchemaVersion": "1.0",
                "partitionKey": "1",
                "sequenceNumber":
"49590338271490256608559692538361571095921575989136588898",
                "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
                "approximateArrivalTimestamp": 1545084650.987
            },
            "eventSource": "aws:kinesis",
            "eventVersion": "1.0",
            "eventID":
"shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
            "eventName": "aws:kinesis:record",
            "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
            "awsRegion": "us-east-2",
            "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream"
        },
        {
            "kinesis": {
                "kinesisSchemaVersion": "1.0",
                "partitionKey": "1",
                "sequenceNumber":
"49590338271490256608559692540925702759324208523137515618",
                "data": "VGhpcyBpcyBvbmx5IGEgdGVzdC4=",
                "approximateArrivalTimestamp": 1545084711.166
            },
            "eventSource": "aws:kinesis",
        }
    ]
}
```

```
        "eventVersion": "1.0",
        "eventID":
"shardId-000000000006:49590338271490256608559692540925702759324208523137515618",
        "eventName": "aws:kinesis:record",
        "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
        "awsRegion": "us-east-2",
        "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream"
    }
]
```

如果您有多个应用程序正在从同一个流中读取记录，则可以使用 Kinesis 流使用者而不是标准迭代器。使用者具有专用读取吞吐量，因此它们不必与相同数据的其他使用者竞争。对于使用者，Kinesis 通过 HTTP/2 连接将记录推送到 Lambda，这也减少添加记录和函数调用之间的延迟。

Note

HTTP/2 流使用者无法使用批处理窗口、错误处理和并发设置。

默认情况下，只要流中有记录，Lambda 就会调用您的函数。如果从流中读取的批处理中只有一条记录，则 Lambda 只向该函数发送一条记录。为避免在少量记录的情况下调用该函数，您可以通过配置批处理时段让事件源缓冲记录，最多可达 5 分钟。在调用该函数之前，Lambda 会继续从流中读取记录，直到收集了完整批次，或者直到批处理时段到期。

如果您的函数返回一个错误，则 Lambda 将重试批处理，直到处理成功或数据过期。为避免分片停滞，可以将事件源映射配置为以较小的批处理大小重试，限制重试次数或者丢弃太早的记录。要保留丢弃的事件，可以配置事件源映射，以将有关失败批处理的详细信息发送到 SQS 队列或 SNS 主题。

您还可以通过并行处理每个分片的多个批处理来提高并发性。在每个分片中，Lambda 最多可以同时处理 10 个批处理。如果您增加每个分片的并发批处理数量，则 Lambda 仍然需要确保在分区键级别进行有序处理。

小节目录

- [配置您的数据流和函数 \(p. 213\)](#)
- [执行角色权限 \(p. 214\)](#)
- [将流配置为事件源 \(p. 214\)](#)
- [事件源映射 API \(p. 215\)](#)
- [错误处理 \(p. 217\)](#)
- [Amazon CloudWatch 指标 \(p. 218\)](#)
- [教程：将 AWS Lambda 与 Amazon Kinesis 结合使用 \(p. 218\)](#)
- [示例函数代码 \(p. 221\)](#)
- [Kinesis 应用程序的 AWS SAM 模板 \(p. 224\)](#)

配置您的数据流和函数

您的 Lambda 函数是数据流的用户应用程序。对于每个分片，它一次处理一批记录。您可以将 Lambda 函数映射到数据流（标准迭代器），或映射到流的使用者（[增强型扇出功能](#)）。

对于标准迭代器，Lambda 将针对记录轮询 Kinesis 流中的每个分片（按照每秒一次的基本频率）。当有更多记录可用时，Lambda 会继续进行批处理，直到函数赶上流的速度。事件源映射与分片的其他使用者共享读取吞吐量。

要最大程度地降低延迟并最大程度地提高读取吞吐量，请创建数据流使用者。流使用者将获得与每个分片的专用连接，这不会影响从流中读取信息的其他应用程序。如果您有许多应用程序读取相同的数据，或者您正在重新处理具有大记录的流，则专用吞吐量可以提供帮助。

流使用者使用 HTTP/2 通过长期连接将记录推送到 Lambda 并压缩请求头来减少延迟。您可以使用 [Kinesis RegisterStreamConsumer API](#) 创建流使用者。

```
$ aws kinesis register-stream-consumer --consumer-name con1 \
--stream-arn arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream
{
    "Consumer": {
        "ConsumerName": "con1",
        "ConsumerARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream/
consumer/con1:1540591608",
        "ConsumerStatus": "CREATING",
        "ConsumerCreationTimestamp": 1540591608.0
    }
}
```

要提高函数处理记录的速度，请将分片添加到数据流中。Lambda 会按顺序处理每个分片中的记录。如果您的函数返回错误，它会停止处理分片中的其他记录。使用更多分片，可以同时处理更多批次，从而降低错误对并发性的影响。

如果您的函数无法纵向扩展以处理并发批处理的总数，请为您的函数[请求提高限额 \(p. 27\)](#)或[预留并发 \(p. 54\)](#)。

执行角色权限

Lambda 需要以下权限才能管理与您的 Kinesis 数据流相关的资源。将它们添加到您的函数的[执行角色 \(p. 30\)](#)。

- [kinesis:DescribeStream](#)
- [kinesis:DescribeStreamSummary](#)
- [kinesis:GetRecords](#)
- [kinesis:GetShardIterator](#)
- [kinesis>ListShards](#)
- [kinesis>ListStreams](#)
- [kinesis:SubscribeToShard](#)

[AWSLambdaKinesisExecutionRole](#) 托管策略包含这些权限。有关更多信息，请参阅[AWS Lambda 执行角色 \(p. 30\)](#)。

要将失败批处理的记录发送到队列或主题，您的函数需要其他权限。每项目标服务均需要不同的权限，如下所示：

- Amazon SQS – [sq:SendMessage](#)
- Amazon SNS – [sns:Publish](#)

将流配置为事件源

创建事件源映射以指示 Lambda 将数据流中的记录发送到 Lambda 函数。您可以创建多个事件源映射，以使用多个 Lambda 函数处理相同的数据，或使用单个函数处理来自多个数据流的项目。

要在 Lambda 控制台中将您的函数配置为从 Kinesis 读取，请创建 Kinesis 触发器。

创建触发器

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Designer 下方，选择 Add trigger (添加触发器)。
4. 选择触发器类型。

5. 配置所需选项，然后选择 Add (添加)。

Lambda 支持 Kinesis 事件源的以下选项。

事件源选项

- Kinesis 流 – 从中读取记录的 Kinesis 流。
- 使用者 (可选) – 使用流使用者通过专用连接从流中读取。
- Batch size (批处理大小) – 每个批处理中发送到函数的记录的数量 (最多 10000 条)。Lambda 通过单个调用将批处理中的所有记录传递给函数，前提是事件的总大小未超出同步调用的 [负载限制 \(p. 27\)](#) (6 MB)。
- 批处理时段 – 指定在调用函数之前收集记录的最长时间 (以秒为单位)。
- 开始位置 – 仅处理新记录、所有现有记录或在特定日期之后创建的记录。
 - 最新 – 处理已添加到流中的新记录。
 - 时间范围 – 处理流中的所有记录。
 - 位于时间戳 – 处理从特定时间开始的记录。

在处理任何现有记录后，函数将继续处理新记录。

- On-failure destination (故障目标) – SQS 队列或 SNS 主题，用于无法处理的记录。当 Lambda 由于时间太远或已用尽所有重试而丢弃一批记录时，它将有关该批处理的详细信息发送到队列或主题。
- Retry attempts (重试) – 函数返回错误时 Lambda 重试的最大次数。这不适用于批处理未到达该函数的服务错误或限制。
- Maximum age of record (最长记录期限) – Lambda 发送到函数的记录的最长期限。
- Split batch on error (出错时拆分批处理) – 当函数返回错误时，请在重试之前将批处理拆分为两部分。
- Concurrent batches per shard (每个分片的并发批处理) – 并发处理来自同一分片的多个批处理。
- 已启用 – 禁用事件源可停止处理记录。Lambda 将跟踪已处理的最后一条记录，并在重新启用后从停止位置重新开始处理。

之后，要管理事件源配置，请在设计器中选择触发器。

事件源映射 API

要使用 AWS CLI 或 AWS 开发工具包管理事件源映射，请使用以下 API 操作：

- [CreateEventSourceMapping \(p. 415\)](#)
- [ListEventSourceMappings \(p. 492\)](#)
- [GetEventSourceMapping \(p. 451\)](#)
- [UpdateEventSourceMapping \(p. 547\)](#)
- [DeleteEventSourceMapping \(p. 432\)](#)

要使用 AWS CLI 创建事件源映射，请使用 `create-event-source-mapping` 命令。以下示例使用 AWS CLI 将名为 `my-function` 的函数映射到 Kinesis 数据流。数据流由 Amazon 资源名称 (ARN) 指定，批处理大小为 500，从以 Unix 时间表示的时间戳开始。

```
$ 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
{
    "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
}
```

要使用一个使用者，请指定使用者的 ARN 而不是流的 ARN。

配置其他选项，以自定义如何处理批处理，并指定何时丢弃无法处理的记录。以下示例更新事件源映射，以在两次重试之后或者如果失败记录已存在一个小时以上，将失败记录发送到 SQS 队列。

```
$ 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"}}'
{
    "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
}
```

更新的设置是异步应用的，并且直到该过程完成才反映在输出中。使用 `get-event-source-mapping` 命令可查看当前状态。

```
$ aws lambda get-event-source-mapping --uuid f89f8514-cdd9-4602-9e1f-01a5b77d449b
{
    "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
}
```

要同时处理多个批处理，请使用 `--parallelization-factor` 选项。

```
$ aws lambda update-event-source-mapping --uuid 2b733gdc-8ac3-cdf5-af3a-1827b3b11284 \
--parallelization-factor 5
```

错误处理

从 Kinesis 流中读取记录的事件源映射将同步调用函数并在出错时重试。如果函数受到限制，或者 Lambda 服务未调用该函数而返回错误，Lambda 将重试，直到记录到期或者超过您在事件源映射上配置的最长期限。

如果函数接收到记录但返回错误，Lambda 将重试，直到批处理中的记录到期、超过最大使用期限或者达到配置的重试限制。对于函数错误，您还可以配置事件源映射，以将失败的批处理拆分为两个批处理。重试较小的批处理可以隔离不良记录并解决超时问题。拆分批处理不计入重试限制。

如果错误处理措施失败，Lambda 将丢弃记录并继续处理流中的批处理。使用默认设置，这意味着不良记录最多可以将针对受影响分片的处理操作阻止 one week。为避免这种情况，请以合理的重试次数和适合您使用案例的最长记录期限来配置函数的事件源映射。

要保留废弃批处理的记录，请配置失败事件目标。Lambda 将文档和有关批处理的详细信息发送到目标队列或主题。

配置失败事件记录的目标

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Designer (设计器) 下，选择 Add destination (添加目标)。
4. 对于 Source (源)，选择 Stream invocation (流调用)。
5. 对于 Stream (流)，选择映射到函数的流。
6. 对于 Destination type (目标类型)，请选择接收调用记录的资源类型。
7. 对于 Destination (目标)，请选择一个资源。
8. 选择保存。

以下示例显示了 Kinesis 流的调用记录。

Example 调用记录

```
{
  "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"
    }
}
```

您可以使用此信息从流中检索受影响的记录以进行故障排除。实际的记录不包括在内，因此您必须处理这些记录，并在它们到期并丢失之前从流中检索它们。

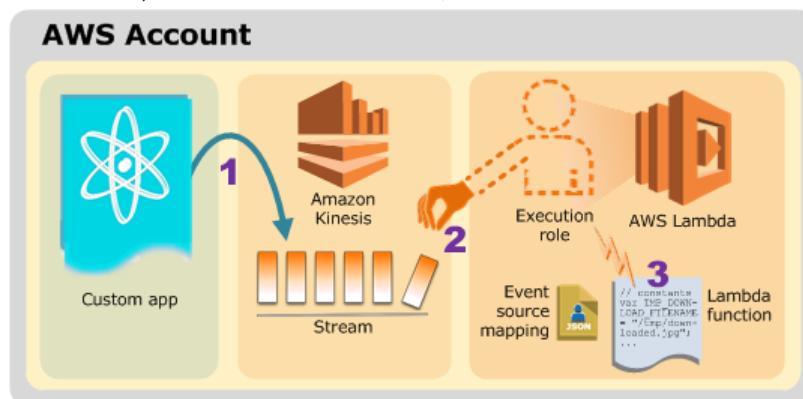
Amazon CloudWatch 指标

在您的函数处理完一批记录后，Lambda 将发出 `IteratorAge` 指标。该指标指示处理完成时批处理中最后一条记录的时间。如果您的函数正在处理新事件，则可使用迭代器期限来估算新记录的添加时间与函数处理新记录的时间之间的延迟。

迭代器期限中的上升趋势可以指示您的函数问题。有关更多信息，请参阅[使用 AWS Lambda 函数指标 \(p. 368\)](#)。

教程：将 AWS Lambda 与 Amazon Kinesis 结合使用

在本教程中，您将创建一个 Lambda 函数来处理来自 Kinesis 流的事件。下图说明应用程序的流程：



1. 自定义应用程序将记录写入流。
2. AWS Lambda 轮询流并在检测到流中的新记录时调用 Lambda 函数。
3. AWS Lambda 通过代入您在创建 Lambda 函数时指定的执行角色来执行 Lambda 函数。

先决条件

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以[安装 Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

创建执行角色

创建[执行角色 \(p. 30\)](#)，向您的函数授予访问 AWS 资源的权限。

创建执行角色

1. 打开 IAM 控制台中的[“角色”页面](#)。
2. 选择 Create role (创建角色)。
3. 创建具有以下属性的角色。
 - 可信任的实体 – AWS Lambda。
 - Permissions (权限) – AWSLambdaKinesisExecutionRole。
 - 角色名称 (角色名称) – **lambda-kinesis-role**。

AWSLambdaKinesisExecutionRole 策略具有该函数从 Kinesis 中读取项目并将日志写入 CloudWatch Logs 所需的权限。

创建函数

以下示例代码接收 Kinesis 事件输入并对其所包含的消息进行处理。为了展示这个过程，代码会将一些传入的事件数据写入 CloudWatch Logs。

Note

有关使用其他语言的示例代码，请参阅[示例函数代码 \(p. 221\)](#)。

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);
    });
};
```

创建函数

1. 将示例代码复制到名为 `index.js` 的文件中。
2. 创建部署程序包。

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

3. 使用 `create-function` 命令创建 Lambda 函数。

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

测试 Lambda 函数。

使用 `invoke` AWS Lambda CLI 命令和示例 Kinesis 事件手动调用 Lambda 函数。

测试 Lambda 函数

- 将以下 JSON 复制到文件中并将其保存为 `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"  
        }  
    ]  
}
```

- 使用 `invoke` 命令将事件发送到该函数。

```
$ aws lambda invoke --function-name ProcessKinesisRecords --payload file://input.txt  
out.txt
```

响应将保存到 `out.txt` 中。

创建 Kinesis 流

使用 `create-stream` 命令创建流。

```
$ aws kinesis create-stream --stream-name lambda-stream --shard-count 1
```

运行下面的 `describe-stream` 命令以获取流 ARN。

```
$ aws kinesis describe-stream --stream-name lambda-stream  
{  
    "StreamDescription": {  
        "Shards": [  
            {  
                "ShardId": "shardId-000000000000",  
                "HashKeyRange": {  
                    "StartingHashKey": "0",  
                    "EndingHashKey": "340282366920746074317682119384634633455"  
                },  
                "SequenceNumberRange": {  
                    "StartingSequenceNumber":  
                        "49591073947768692513481539594623130411957558361251844610"  
                }  
            }  
        ],  
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/lambda-stream",  
        "StreamStatus": "ACTIVE",  
        "LastUpdateAt": "2018-05-10T14:44:10Z",  
        "RetentionPeriod": 300, "StreamCreationTimestamp": "2018-05-10T14:44:10Z",  
        "StreamSize": 1000000000000000000, "ShardLevelMetrics": "Enabled",  
        "StreamType": "Standard", "EncryptionType": "KMS",  
        "KMSMasterKeyId": "arn:aws:kms:us-west-2:123456789012:key/12345678901234567890123456789012",  
        "LastModifiedAt": "2018-05-10T14:44:10Z", "ShardCount": 1, "StreamArn": "arn:aws:kinesis:us-west-2:123456789012:stream/lambda-stream",  
        "StreamStatusDetails": {  
            "LastUpdateAt": "2018-05-10T14:44:10Z",  
            "LastUpdateReason": "Stream was created."  
        }  
    }  
}
```

```
    "StreamName": "lambda-stream",
    "StreamStatus": "ACTIVE",
    "RetentionPeriodHours": 24,
    "EnhancedMonitoring": [
        {
            "ShardLevelMetrics": []
        }
    ],
    "EncryptionType": "NONE",
    "KeyId": null,
    "StreamCreationTimestamp": 1544828156.0
}
```

您将使用下一步中的流 ARN 来将该流关联到您的 Lambda 函数。

在 AWS Lambda 中添加事件源。

运行以下 AWS CLI `add-event-source` 命令。

```
$ 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
```

记下映射 ID 以供将来使用。您可以通过运行 `list-event-source-mappings` 命令获取事件源映射的列表。

```
$ aws lambda list-event-source-mappings --function-name ProcessKinesisRecords \
--event-source arn:aws:kinesis:us-west-2:123456789012:stream/lambda-stream
```

在该响应中，您可以验证状态值是否为 `enabled`。可以禁用事件源映射，以临时暂停轮询而不丢失任何记录。

测试设置

要测试事件源映射，请将事件记录添加到 Kinesis 流中。`--data` 值是一个字符串，CLI 先将其编码为 base64 字符串，然后才发送到 Kinesis。您可以多次运行同一命令来向流中添加多条记录。

```
$ aws kinesis put-record --stream-name lambda-stream --partition-key 1 \
--data "Hello, this is a test."
```

Lambda 使用执行角色来读取来自流的记录。然后它将调用 Lambda 函数，批量传递记录。该函数解码每条记录中的数据并记录它，将输出发送到 CloudWatch Logs 中。在 CloudWatch 控制台中查看这些日志。

示例函数代码

要处理来自 Amazon Kinesis 的事件，请遍历事件对象中包含的记录，并对每个对象中包含的 Base64 编码的数据进行解码。

Note

此页面上的代码不支持[聚合记录](#)。您可以在 Kinesis Producer Library 配置中禁用聚合，也可以使用[Kinesis Record Aggregation 库](#)来取消聚合记录。

示例代码具有以下语言。

主题

- [Node.js 8 \(p. 222\)](#)
- [Java 11 \(p. 222\)](#)
- [C# \(p. 223\)](#)
- [Python 3 \(p. 224\)](#)
- [转到 \(p. 224\)](#)

Node.js 8

以下示例代码接收 Kinesis 事件输入并对其所包含的消息进行处理。为了展示这个过程，代码会将一些传入的事件数据写入 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);
    });
};
```

压缩示例代码以创建部署程序包。有关说明，请参阅[Node.js 中的 AWS Lambda 部署程序包 \(p. 268\)](#)。

Java 11

以下是将 Kinesis 事件记录数据作为输入接收并对其进行处理的示例 Java 代码。为了展示这个过程，代码会将一些传入的事件数据写入 CloudWatch Logs。

在代码中，`recordHandler` 是处理程序。该处理程序使用了在 `aws-lambda-java-events` 库中定义的预定义 `KinesisEvent` 类。

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.KinesisEvent.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;
    }
}
```

如果该处理程序正常返回并且没有异常，则 Lambda 认为输入的记录批次得到成功处理并开始读取流中的新记录。如果该处理程序引发异常，则 Lambda 认为输入的记录批次未得到处理，并用相同的记录批次再次调用该函数。

附属物

- aws-lambda-java-core
- aws-lambda-java-events
- aws-java-sdk

使用 Lambda 库依赖项构建代码以创建部署程序包。有关说明，请参阅[Java 中的 AWS Lambda 部署程序包 \(p. 302\)](#)。

C#

以下是将 Kinesis 事件记录数据作为输入接收并对其进行处理的示例 C# 代码。为了展示这个过程，代码会将一些传入的事件数据写入 CloudWatch Logs。

在代码中，`HandleKinesisRecord` 是处理程序。该处理程序使用了在 `Amazon.Lambda.KinesisEvents` 库中定义的预定义 `KinesisEvent` 类。

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();
            }
        }
    }
}
```

使用以上示例替换.NET Core 中的 `Program.cs` 有关说明，请参阅[C# 中的 AWS Lambda 部署程序包 \(p. 342\)](#)。

Python 3

以下是将 Kinesis 事件记录数据作为输入接收并对其进行处理的示例 Python 代码。为了展示这个过程，代码会将一些传入的事件数据写入 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))
```

压缩示例代码以创建部署程序包。有关说明，请参阅[Python 中的 AWS Lambda 部署程序包 \(p. 279\)](#)。

转到

以下是将 Kinesis 事件记录数据作为输入接收并对其进行处理的示例 Go 代码。为了展示这个过程，代码会将一些传入的事件数据写入 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)
    }
}
```

使用 `go build` 构建可执行文件并创建部署程序包。有关说明，请参阅[Go 中的 AWS Lambda 部署程序包 \(p. 330\)](#)。

Kinesis 应用程序的 AWS SAM 模板

您可以使用 [AWS SAM](#) 构建此应用程序。要了解有关创建 AWS SAM 模板的更多信息，请参阅 AWS 无服务器应用程序模型 开发人员指南 中的 [AWS SAM 模板基础知识](#)。

下面是[教程 \(p. 218\)](#)中 Lambda 应用程序的示例 AWS SAM 模板。模板中的函数和处理程序适用于 Node.js 代码。如果您使用不同的代码示例，请相应地更新这些值。

Example template.yaml - Kinesis Stream

```
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

```

该模板创建一个 Lambda 函数、一个 Kinesis 流以及一个事件源映射。事件源映射从流中读取并调用该函数。

要使用 [HTTP/2 流使用者 \(p. 213\)](#)，请在模板中创建使用者，并将事件源映射配置为从使用者而不是从流中读取。

Example template.yaml - Kinesis Stream 使用者

```

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
```

有关如何使用程序包和部署命令打包和部署无服务器应用程序的信息，请参阅 AWS 无服务器应用程序模型开发人员指南 中的[部署无服务器应用程序](#)。

将 AWS Lambda 与 Amazon Kinesis Data Firehose 结合使用

Amazon Kinesis Data Firehose 捕获、转换流数据并将其加载到下游服务，如 Kinesis Data Analytics 或 Amazon S3。您可以编写 Lambda 函数来请求附加的自定义数据处理，然后它会在下游发送。

Example Amazon Kinesis Data Firehose 消息事件

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

有关更多信息，请参阅 Kinesis Data Firehose 开发人员指南中的[Amazon Kinesis Data Firehose 数据转换](#)。

将 AWS Lambda 与 Amazon Lex 结合使用

您可以使用 Amazon Lex 将会话自动程序集成到您的应用程序中。Amazon Lex 自动程序提供了与用户的对话界面。Amazon Lex 提供了与 Lambda 的预构建集成，这使您能够将 Lambda 函数与您的 Amazon Lex 自动程序一起使用。

配置 Amazon Lex 自动程序时，您可以指定一个 Lambda 函数来执行验证和/或履行。为了进行验证，Amazon Lex 在用户的每次响应后调用 Lambda 函数。Lambda 功能可以验证响应，并在必要时向用户提供纠正性反馈。为了进行履行，在自动程序成功收集所有必需信息并收到用户的确认后，Amazon Lex 调用 Lambda 函数来完成用户请求。

您可以对 Lambda 函数管理并发性 ([p. 54](#))，以控制您提供服务的同时发生的自动程序对话的最大数量。如果函数处于最大并发状态，Amazon Lex API 将返回 HTTP 429 状态代码（请求过多）。

如果 Lambda 函数引发异常，API 将返回 HTTP 424 状态代码（依赖项失败异常）。

Amazon Lex 自动程序同步 ([p. 81](#)) 调用 Lambda 函数。事件参数包含有关自动程序和对话框中每个槽的值的信息。`invocationSource` 参数指示 Lambda 函数是应验证输入 (DialogCodeHook) 还是履行意图 (FulfillmentCodeHook)。

Example Amazon Lex 消息事件

```
{  
    "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"  
    }  
}
```

Amazon Lex 预计来自 Lambda 函数的响应，格式如下。`dialogAction` 字段为必填项。`sessionAttributes` 和 `recentIntentSummaryView` 字段是可选的。

Example Amazon Lex 消息事件

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

请注意，dialogAction 所需的其他字段根据 type 字段的值而有所不同。有关事件和响应字段的更多信息，请参阅 Amazon Lex 开发人员指南 中的 [Lambda 事件和响应格式](#)。有关演示如何将 Lambda 与 Amazon Lex 一起使用的示例教程，请参阅 Amazon Lex 开发人员指南 中的 [练习 1：使用蓝图创建 Amazon Lex 自动程序](#)。

角色和权限

您需要将服务相关角色配置为函数的[执行角色 \(p. 30\)](#)。Amazon Lex 定义具有预定义权限的服务相关角色。当您使用控制台创建 Amazon Lex 自动程序时，会自动创建服务相关角色。要使用 AWS CLI 创建服务相关角色，请使用 `create-service-linked-role` 命令。

```
$ aws iam create-service-linked-role --aws-service-name lex.amazonaws.com
```

此命令创建以下角色。

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

```
    "RoleName": "AWSLambdaRoleForLexBots",
    "Path": "/aws-service-role/lex.amazonaws.com/",
    "Arn": "arn:aws:iam::account-id:role/aws-service-role/lex.amazonaws.com/
AWSLambdaRoleForLexBots"
}
```

如果您的 Lambda 函数使用其他 AWS 服务，则需要向服务相关角色添加相应的权限。

您可以使用基于资源的权限策略来允许 Amazon Lex 意图调用您的 Lambda 函数。如果您使用 Amazon Lex 控制台，将自动创建权限策略。从 AWS CLI 中，使用 Lambda add-permission 命令设置权限。以下示例设置 OrderFlowers 意图的权限。

```
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:*
```

将 AWS Lambda 与 Amazon RDS 配合使用

您可以使用 AWS Lambda 处理来自 Amazon Relational Database Service (Amazon RDS) 数据库的事件通知。Amazon RDS 将通知发送到 Amazon Simple Notification Service (Amazon SNS) 主题，您可以将其配置为调用 Lambda 函数。Amazon SNS 将来自 Amazon RDS 的消息包装在自己的事件文档中，并将其发送到您的函数。

Example Amazon SNS 事件中的 Amazon RDS 消息

```
{
  "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/ebXEkaAi6RibDsvpi+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"
      }
    }
  ]
}
```

}

有关配置 Amazon RDS 数据库来发送通知的说明，请参阅 Amazon Relational Database Service 用户指南中的[使用 Amazon RDS 事件通知](#)。

有关使用 Amazon SNS 作为触发器的详细信息，请参阅[将 AWS Lambda 与 Amazon SNS 结合使用 \(p. 250\)](#)。

主题

- 教程 : 配置 Lambda 函数以访问 Amazon VPC 中的 Amazon RDS (p. 230)

教程 : 配置 Lambda 函数以访问 Amazon VPC 中的 Amazon RDS

在本教程中，您将执行以下操作：

- 在您的默认 Amazon VPC 中启动 Amazon RDS MySQL 数据库引擎实例。在 MySQL 实例中，您将创建一个数据库 (ExampleDB)，其中包含一个示例表 (Employee)。有关 Amazon RDS 的更多信息，请参阅[Amazon RDS](#)。
- 创建一个 Lambda 函数来访问 ExampleDB 数据库，创建一个表 (Employee)，添加几个记录，然后检索表中的记录。
- 调用 Lambda 函数并验证查询结果。这样您可以验证您的 Lambda 函数是否可以访问 VPC 中的 RDS MySQL 实例。

有关将 Lambda 与 Amazon VPC 结合使用的详细信息，请参阅[配置 Lambda 函数以访问 VPC 中的资源 \(p. 72\)](#)。

先决条件

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以[安装 Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

创建执行角色

创建[执行角色 \(p. 30\)](#)，向您的函数授予访问 AWS 资源的权限。

创建执行角色

1. 打开 IAM 控制台中的[“角色”页面](#)。
2. 选择 Create role (创建角色)。
3. 创建具有以下属性的角色。
 - Trusted entity (可信任的实体) – Lambda.

- 权限 – AWSLambdaVPCAccessExecutionRole。
- 角色名称 (角色名称) – **lambda-vpc-role**。

AWSLambdaVPCAccessExecutionRole 具有函数管理与 VPC 的网络连接所需的权限。

创建 Amazon RDS 数据库实例

在本教程中，示例 Lambda 函数创建一个表 (Employee)，插入一些记录，然后检索这些记录。Lambda 函数创建的表具有以下架构：

```
Employee(EmpID, Name)
```

其中 EmpID 是主键。现在，您需要向该表添加一些记录。

首先，您在您的默认 VPC 中启动 ExampleDB 数据库的 RDS MySQL 实例。如果您的默认 VPC 中已经在运行 RDS MySQL 实例，请跳过此步骤。

您可以使用以下方法之一启动 RDS MySQL 实例：

- 按照 Amazon RDS 用户指南 的 [创建 MySQL 数据库实例并连接到 MySQL 数据库实例上的数据库](#) 中的说明操作。
- 使用以下 AWS CLI 命令：

```
$ 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
```

记下数据库名称、用户名和密码。您还需要数据库实例的主机地址（终端节点），您可以从 RDS 控制台中获取该地址。您可能需要等待，直到实例状态变为可用并且终端节点值显示在控制台中。

创建部署程序包

以下 Python 代码示例针对您在 VPC 中创建的 MySQL RDS 实例中的 Employee 表运行 SELECT 查询。该代码在 ExampleDB 数据库中创建表，添加示例记录，并检索这些记录。

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(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)
```

在处理程序之外执行 `pymysql.connect()` 允许您的函数重新使用数据库连接，以获得更好的性能。

第二个文件包含函数的连接信息。

Example rds_config.py

```
#config file containing credentials for RDS MySQL instance
db_username = "username"
db_password = "password"
db_name = "ExampleDB"
```

附属物

- `pymysql` – Lambda 函数代码使用此库来访问您的 MySQL 实例（请参阅 [PyMySQL](#)）。

安装 Pip 的依赖项并创建部署程序包。有关说明，请参阅[Python 中的 AWS Lambda 部署程序包 \(p. 279\)](#)。

创建 Lambda 函数

使用 `create-function` 命令创建 Lambda 函数。您可以在 [Amazon VPC 控制台](#) 中找到默认 VPC 的子网 ID 和安全组 ID。

```
$ 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
```

测试 Lambda 函数。

在此步骤中，您将使用 `invoke` 命令手动调用 Lambda 函数。当 Lambda 函数执行时，它会对 RDS MySQL 实例中的 `Employee` 表运行 `SELECT` 查询，然后输出结果，这些结果还会显示在 CloudWatch Logs 中。

1. 使用 invoke 命令调用 Lambda 函数。

```
$ aws lambda invoke --function-name CreateTableAddRecordsAndRead output.txt
```

2. 按以下过程验证 Lambda 函数是否已成功执行：

- 查看 output.txt 文件。
- 在 AWS Lambda 控制台中查看结果。
- 在 CloudWatch Logs 中验证结果。

您现已创建 Lambda 函数来访问您的 VPC 中的数据库，可以调用该函数来响应事件。有关配置事件源和示例的信息，请参阅[将 AWS Lambda 与其他服务结合使用 \(p. 138\)](#)。

将 AWS Lambda 与 Amazon S3 事件结合使用

您可以使用 Lambda 处理来自 Amazon Simple Storage Service 的[事件通知](#)。在创建或删除对象时，Amazon S3 会向 Lambda 函数发送事件。您在存储桶上配置通知设置，并向 Amazon S3 授予权限来根据函数的基于资源的权限策略调用函数。

Warning

如果您的 Lambda 函数使用触发它的同一存储桶，则会导致在一个循环中执行该函数。例如，如果每当上传一个对象，存储桶就触发某个函数，而该函数又上传一个对象给存储桶，则该函数间接触发了自身。为避免这种情况，请使用两个存储桶，或将触发器配置为仅适用于传入对象所用的前缀。

Amazon S3 使用包含有关对象的详细信息的事件[异步 \(p. 83\)](#)调用您的函数。以下示例显示了在将部署包上传到 Amazon S3 时 Amazon S3 发送的事件。

Example Amazon S3 通知事件

```
{
  "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":
        "v1R7PnpV2Ce81l0PRw6jlUpck7Jo5ZsQjryTjKlc5aLWGVHPZLj5NeC6qMa0emYBDXOo6QBU0Wo="
      },
      "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"
        }
    }
}
```

要调用您的函数，Amazon S3 需要来自该函数的[基于资源的策略 \(p. 33\)](#)的权限。当您在 Lambda 控制台中配置 Amazon S3 触发器时，该控制台将修改基于资源的策略以允许 Amazon S3 在存储桶名称和账户 ID 匹配时调用函数。如果您在 Amazon S3 中配置通知，请使用 Lambda API 更新策略。您还可以使用 Lambda API 向另一个账户授予权限，或将权限限制到指定的别名。

如果您的函数使用 AWS 开发工具包来管理 Amazon S3 资源，则其[执行角色 \(p. 30\)](#)也需要 Amazon S3 权限。

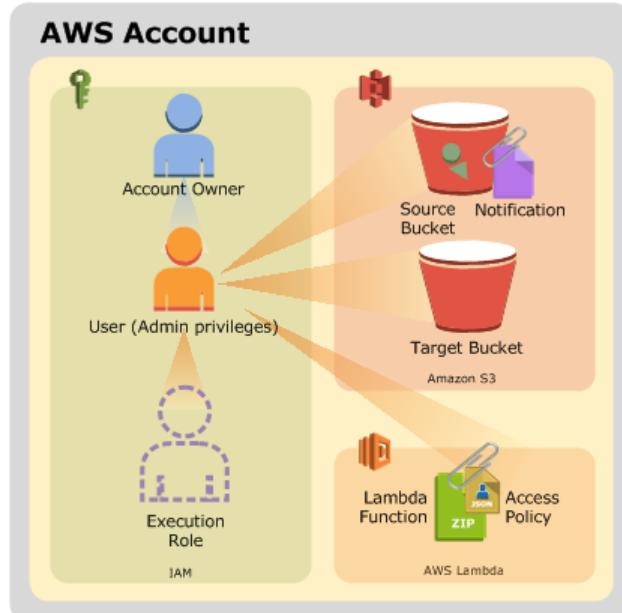
主题

- 教程：将 AWS Lambda 与 Amazon S3 结合使用 (p. 234)
- 示例 Amazon S3 函数代码 (p. 240)
- Amazon S3 应用程序的 AWS SAM 模板 (p. 246)

教程：将 AWS Lambda 与 Amazon S3 结合使用

假设您要为上传到存储桶的每个图像文件创建一个缩略图。您可以创建一个 Lambda 函数 (CreateThumbnail)，在创建对象后，Amazon S3 可调用该函数。之后，Lambda 函数可以从源存储桶读取图像对象并创建缩略图目标存储桶。

完成本教程后，您的账户中将具有以下 Amazon S3、Lambda 和 IAM 资源：



Lambda 资源

- Lambda 函数。

- 一个与 Lambda 函数关联的访问策略，此策略向 Amazon S3 授予调用 Lambda 函数的权限。

IAM 资源

- 一个执行角色，此角色通过与其关联的权限策略授予您的 Lambda 函数所需的权限。

Amazon S3 资源

- 具有调用 Lambda 函数的通知配置的源存储桶。
- 函数在其中保存已调整大小的图像的目标存储桶。

先决条件

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command  
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以[安装 Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

安装 npm 来管理函数的依赖关系。

本教程使用 AWS CLI 命令来创建和调用 Lambda 函数。安装 [AWS CLI](#)，并[配置 AWS CLI 以便与您的 AWS 凭证结合使用](#)

创建执行角色

创建[执行角色 \(p. 30\)](#)，向您的函数授予访问 AWS 资源的权限。

创建执行角色

1. 打开 IAM 控制台中的[“角色”页面](#)。
2. 选择 Create role (创建角色)。
3. 创建具有以下属性的角色。
 - 可信任的实体 – AWS Lambda。
 - 权限 – AWSLambdaExecute。
 - 角色名称 (角色名称) – **lambda-s3-role**。

AWSLambdaExecute 策略具有该函数在 Amazon S3 中管理对象并将日志写入 CloudWatch Logs 所需的权限。

创建存储桶并上传示例对象

按照以下步骤创建存储桶并上传对象。

1. 打开 [Amazon S3 控制台](#)。

2. 创建两个存储桶。目标存储桶名称必须为后跟 **-resized** 的 **source**，其中 **source** 是您希望用于源的存储桶的名称。例如，`mybucket` 和 `mybucket-resized`。
3. 在源存储桶中，上传一个 `.jpg` 对象 `HappyFace.jpg`。

在连接到 Amazon S3 之前手动调用 Lambda 函数时，您要将示例事件数据传递到指定源存储桶和 `HappyFace.jpg` 作为新建对象的函数，因此您需要先创建此示例对象。

创建函数

以下示例代码接收 Amazon S3 事件输入并对其所包含的消息进行处理。它调整源存储桶中图像的大小并将输出保存到目标存储桶。

Note

有关使用其他语言的示例代码，请参阅 [示例 Amazon S3 函数代码 \(p. 240\)](#)。

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);
};

};
```

查看上述代码并注意以下内容：

- 函数通过作为参数接收的事件数据获知源存储桶名称和对象键名称。如果对象为 .jpg 或 .png，则该代码会创建一个缩略图并将其保存到目标存储桶。
- 该代码假定目标存储桶已存在，且其名称为源存储桶名称后跟字符串 -resized。例如，如果在事件数据中识别的源存储桶为 examplebucket，则代码假定您具有目标存储桶 examplebucket-resized。
- 对于所创建的缩略图，该代码会将其键名称派生为后跟源对象键名称的字符串 resized-。例如，如果源对象键为 sample.jpg，则代码会创建具有键 resized-sample.jpg 的缩略图对象。

部署程序包是包含 Lambda 函数代码和依赖项的 .zip 文件。

创建部署程序包

1. 将函数代码保存为名为 lambda-s3 的文件夹中的 index.js。
2. 使用 npm 安装 Sharp 库。对于 Linux，请使用以下命令。

```
lambda-s3$ npm install sharp
```

对于 macOS，请使用以下命令。

```
lambda-s3$ npm install --arch=x64 --platform=linux --target=12.13.0 sharp
```

完成此步骤后，文件夹结构如下：

```
lambda-s3
|- index.js
|- /node_modules/sharp
# /node_modules/...
```

3. 创建包含函数代码和依赖项的部署包。

```
lambda-s3$ zip -r function.zip .
```

创建函数

- 使用 `create-function` 命令创建 Lambda 函数。

```
$ aws lambda create-function --function-name CreateThumbnail \
--zip-file file://function.zip --handler index.handler --runtime nodejs12.x \
--timeout 10 --memory-size 1024 \
--role arn:aws:iam::123456789012:role/lambda-s3-role
```

对于角色参数，请将编号序列替换为您的 AWS 账户 ID。之前的示例命令指定 10 秒超时值作为函数配置。根据上传的对象的大小，可能需要使用下面的 AWS CLI 命令增大超时值。

```
$ aws lambda update-function-configuration --function-name CreateThumbnail --timeout 30
```

测试 Lambda 函数。

在本步骤中，您将使用示例 Amazon S3 事件数据手动调用 Lambda 函数。

测试 Lambda 函数

1. 将下面的 Amazon S3 示例事件数据保存到某个文件中，并将该文件另存为 `inputFile.txt`。您需要提供 `sourcebucket` 名称和 `.jpg` 对象键来更新该 JSON。

```
{
  "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",
        "configurationId": "testConfigRule",
        "bucket": {
          "name": "sourcebucket",
          "ownerIdentity": {
            "principalId": "AIDAJDPLRKLG7UEXAMPLE"
          }
        }
      }
    }
  ]
}
```

```
        "principalId": "A3NL1KOZZKExample"
    },
    "arn": "arn:aws:s3:::sourcebucket"
},
"object": {
    "key": "HappyFace.jpg",
    "size": 1024,
    "eTag": "d41d8cd98f00b204e9800998ecf8427e",
    "versionId": "096fKKXTRTl3on89fVO.nfljtsv6qko"
}
}
]
```

- 运行下面的 Lambda CLI invoke 命令以调用函数。请注意，该命令会请求异步执行。（可选）可通过将 RequestResponse 指定为 invocation-type 参数值来同步调用它。

```
$ aws lambda invoke --function-name CreateThumbnail --invocation-type Event \
--payload file://inputFile.txt outputFile.txt
```

- 验证目标存储桶中是否已创建缩略图。

配置 Amazon S3 以发布事件

在本步骤中，您将添加剩余的配置，以便 Amazon S3 能够向 AWS Lambda 发布对象创建事件并调用 Lambda 函数。您将在本步骤中执行以下操作：

- 向 Lambda 函数访问策略添加权限以允许 Amazon S3 调用该函数。
- 向源存储桶添加通知配置。在通知配置中，您需要提供以下内容：
 - 需要 Amazon S3 发布的事件的事件类型。在本教程中，您将指定 s3:ObjectCreated:* 事件类型，以便 Amazon S3 在创建对象时发布事件。
 - 要调用的 Lambda 函数。

向函数策略添加权限

- 运行下面的 Lambda CLI add-permission 命令以向 Amazon S3 服务委托人 (s3.amazonaws.com) 授予执行 lambda:InvokeFunction 操作的权限。请注意，向 Amazon S3 授予权限，使其只能在满足以下条件时调用该函数：

- 在特定的存储桶上检测到对象创建事件。
- 该存储桶由您的账户拥有。如果删除一个存储桶，则另一个账户可能会创建具有相同 ARN 的存储桶。

```
$ 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
```

- 通过运行 AWS CLI get-policy 命令验证函数的访问策略。

```
$ aws lambda get-policy --function-name CreateThumbnail
```

在源存储桶上添加通知配置，以请求 Amazon S3 向 Lambda 发布对象创建事件。

Important

此过程将存储桶配置为每当在其中创建对象时调用您的函数。确保仅在源存储桶上配置此选项，并且不要通过触发的函数在源存储桶中创建对象。否则，您的函数可能会导致自身[在循环中被连续调用 \(p. 233\)](#)。

配置通知

1. 打开 [Amazon S3 控制台](#)。
2. 选择源存储桶。
3. 选择 **Properties**。
4. 在 **Events (事件)** 下，使用以下设置配置通知。
 - 名称 – **lambda-trigger**。
 - 事件 – **ObjectCreate (All)**。
 - 发送到 – **Lambda function**。
 - Lambda – **CreateThumbnail**。

有关事件配置的更多信息，请参阅 Amazon Simple Storage Service 控制台用户指南 中的[启用事件通知](#)。

测试设置

现在，可以按以下方式测试设置：

1. 使用 Amazon S3 控制台将 .jpg 或 .png 对象上传到源存储桶。
2. 使用 `CreateThumbnail` 函数验证是否在目标存储桶中创建了缩略图。
3. 在 CloudWatch 控制台中查看日志。

示例 Amazon S3 函数代码

示例代码具有以下语言。

主题

- [Node.js 12.x \(p. 240\)](#)
- [Java 11 \(p. 242\)](#)
- [Python 3 \(p. 245\)](#)

Node.js 12.x

以下示例代码接收 Amazon S3 事件输入并对其所包含的消息进行处理。它调整源存储桶中图像的大小并将输出保存到目标存储桶。

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);  
    };
```

部署程序包是包含 Lambda 函数代码和依赖项的 .zip 文件。

创建部署程序包

1. 创建文件夹 (examplefolder) , 然后创建子文件夹 (node_modules)。
2. 安装依赖项。本代码示例使用以下库：
 - 适用于 Node.js 中 JavaScript 的 AWS 开发工具包
 - 适用于 node.js 的 Sharp

AWS Lambda 运行时已具有用 Node.js 编写的适用于 JavaScript 的 AWS 开发工具包，因此您只需安装 Sharp 库。打开命令提示符，导航到 examplefolder，使用 npm 命令（Node.js 的一部分）安装这些库。对于 Linux，请使用以下命令。

```
$ npm install sharp
```

对于 macOS，请使用以下命令。

```
$ npm install --arch=x64 --platform=linux --target=12.13.0 sharp
```

3. 将示例代码保存到名为 index.js 的文件中。
4. 查看上述代码并注意以下内容：
 - 函数通过作为参数接收的事件数据获知源存储桶名称和对象键名称。如果对象为 .jpg，则该代码会创建一个缩略图并将其保存到目标存储桶。
 - 该代码假定目标存储桶已存在，且其名称为源存储桶名称后跟字符串 -resized。例如，如果在事件数据中识别的源存储桶为 examplebucket，则代码假定您具有目标存储桶 examplebucket-resized。
 - 对于所创建的缩略图，该代码会将其键名称派生为后跟源对象键名称的字符串 resized-。例如，如果源对象键为 sample.jpg，则代码会创建具有键 resized-sample.jpg 的缩略图对象。
5. 将该文件保存到 index.js 中，文件名为 examplefolder。完成此步骤后，文件夹结构如下：

```
index.js  
/node_modules/sharp
```

6. 将 index.js 文件和 node_modules 文件夹压缩为 CreateThumbnail.zip。

Java 11

下面是读取传入的 Amazon S3 事件并创建缩略图的示例 Java 代码。请注意，它实现了 aws-lambda-java-core 库中提供的 RequestHandler 接口。因此，在创建 Lambda 函数时，您可以将该类指定为处理程序（即 example.handler）。有关使用接口提供处理程序的更多信息，请参阅[处理程序接口 \(p. 309\)](#)。

被该处理程序用作输入类型的 S3Event 类型是 aws-lambda-java-events 库中的一个预定义类，它为您提供了一些方便地从传入的 Amazon S3 事件读取信息的方法。该处理程序返回字符串作为输出。

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(".*\\\\.([^\n\\.]*).").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);
}
}
}
```

Amazon S3 使用 Event 调用类型调用您的 Lambda 函数，其中 AWS Lambda 以异步方式执行代码。返回什么不重要。但在本示例中，我们实现的接口要求指定返回类型，因此，本示例中的处理程序使用了 String 作为返回类型。

附属物

- aws-lambda-java-core
- aws-lambda-java-events
- aws-java-sdk

使用 Lambda 库依赖项构建代码以创建部署程序包。有关说明，请参阅[Java 中的 AWS Lambda 部署程序包 \(p. 302\)](#)。

Python 3

以下示例代码接收 Amazon S3 事件输入并对其所包含的消息进行处理。它调整源存储桶中图像的大小并将输出保存到目标存储桶。

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)
```

Note

此代码使用的映像库必须安装在 Linux 环境中，才能创建有效的部署包。

创建部署程序包

1. 将示例代码复制到名为 `lambda_function.py` 的文件中。
2. 创建虚拟环境。

```
s3-python$ virtualenv v-env
s3-python$ source v-env/bin/activate
```

3. 在虚拟环境中安装库

```
(v-env) s3-python$ pip install Pillow boto3
```

4. 使用已安装库的内容创建部署包。

```
(v-env) s3-python$ cd $VIRTUAL_ENV/lib/python3.8/site-packages
(v-env) python-s3/v-env/lib/python3.8/site-packages$ zip -r9 ${OLDPWD}/function.zip .
```

5. 将处理程序代码添加到部署包并停用虚拟环境。

```
(v-env) python-s3/v-env/lib/python3.8/site-packages$ cd ${OLDPWD}
(v-env) python-s3$ zip -g function.zip lambda_function.py
  adding: lambda_function.py (deflated 55%)
(v-env) python-s3$ deactivate
```

Amazon S3 应用程序的 AWS SAM 模板

您可以使用 [AWS SAM](#) 构建此应用程序。要了解有关创建 AWS SAM 模板的更多信息，请参阅 AWS 无服务器应用程序模型开发人员指南中的 [AWS SAM 模板基础知识](#)。

下面是[教程 \(p. 234\)](#)中 Lambda 应用程序的示例 AWS SAM 模板。将以下文本复制到.yaml 文件中，并将其保存到您之前创建的 ZIP 程序包旁。请注意，Handler 和 Runtime 参数值应与上一节中创建函数时所用的参数值匹配。

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
```

有关如何使用程序包和部署命令打包和部署无服务器应用程序的信息，请参阅 AWS 无服务器应用程序模型开发人员指南中的[部署无服务器应用程序](#)。

将 AWS Lambda 与 Amazon S3 批处理操作结合使用

可以使用 Amazon S3 批处理操作对一大组 Amazon S3 对象调用 Lambda 函数。Amazon S3 将跟踪批处理操作的进度，发送通知，并存储显示每个操作的状态的完成报告。

要运行批处理操作，请创建 Amazon S3 批处理操作作业。在创建作业时，您将提供清单（对象列表）并配置要对这些对象执行的操作。

在批处理作业启动时，Amazon S3 会为清单中的每个对象[同步 \(p. 81\)](#)调用 Lambda 函数。事件参数包含存储桶和对象的名称。

以下示例显示 Amazon S3 为 awsexamplebucket 存储桶中名为 customerImage1.jpg 的对象发送到 Lambda 函数的事件。

Example Amazon S3 批处理请求事件示例

```
{  
    "invocationSchemaVersion": "1.0",  
    "invocationId": "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",  
    "job": {  
        "id": "f3cc4f60-61f6-4a2b-8a21-d07600c373ce"  
    },  
    "tasks": [  
        {  
            "taskId": "dGFza2lkZ29lc2hlcUK",  
            "s3Key": "customerImage1.jpg",  
            "s3VersionId": "1",  
            "s3BucketArn": "arn:aws:s3:us-east-1:0123456788:awsexamplebucket"  
        }  
    ]  
}
```

您的 Lambda 函数必须返回带字段的 JSON 对象，如以下示例所示。您可以从事件参数复制 invocationId 和 taskId。您可以在 resultString 中返回字符串。Amazon S3 将 resultString 值保存在完成报告中。

Example Amazon S3 批处理请求响应

```
{  
    "invocationSchemaVersion": "1.0",  
    "treatMissingKeysAs" : "PermanentFailure",  
    "invocationId" : "YXNkbGZqYWRmaiBhc2RmdW9hZHNmZGpmaGFzbGtkaGZza2RmaAo",  
    "results": [  
        {  
            "taskId": "dGFza2lkZ29lc2hlcUK",  
            "resultCode": "Succeeded",  
            "resultString": "[\"Alice\", \"Bob\"]"  
        }  
    ]  
}
```

从 Amazon S3 批处理操作调用 Lambda 函数

您可以使用非限定的或限定的函数 ARN 调用 Lambda 函数。如果要对整个批处理作业使用同一函数版本，请在创建作业时在 FunctionARN 参数中配置特定的函数版本。在配置别名或 \$LATEST 限定符的情况下，如果在作业执行期间更新别名或 \$LATEST，则批处理作业会立即开始调用函数的新版本。

请注意，您不能对批处理操作重用现有的 Amazon S3 的基于事件的函数。这是因为 Amazon S3 批处理操作将不同的事件参数传递给 Lambda 函数，并且需要一条带特定 JSON 结构的返回消息。

在为 Amazon S3 批处理作业创建的[基于资源的策略 \(p. 33\)](#)中，请确保为作业设置调用 Lambda 函数的权限。

在函数的[执行角色 \(p. 30\)](#)中，为 Amazon S3 设置一个信任策略以便它在执行函数时代入该角色。

如果您的函数使用 AWS 开发工具包来管理 Amazon S3 资源，则您需要在执行角色中添加 Amazon S3 权限。

在作业执行时，Amazon S3 会启动多个函数实例来并行处理 Amazon S3 对象，直至函数的[并发限制 \(p. 94\)](#)。Amazon S3 会限制实例的初始增加以避免较小作业的额外成本。

如果 Lambda 函数返回 `TemporaryFailure` 响应代码，则 Amazon S3 会重试操作。

有关 Amazon S3 批处理操作的更多信息，请参阅 Amazon S3 开发人员指南中的[执行批处理操作](#)。

有关如何在 Amazon S3 批处理操作中使用 Lambda 函数的示例，请参阅 Amazon S3 开发人员指南中的[从 Amazon S3 批处理操作调用 Lambda 函数](#)。

将 AWS Lambda 与 Amazon SES 结合使用

在使用 Amazon SES 接收消息时，可以将 Amazon SES 配置为在消息到达时调用您的 Lambda 函数。然后，该服务通过将实际上是 Amazon SNS 事件中的 Amazon SES 消息的传入电子邮件事件作为一个参数传入，来调用您的 Lambda 函数。

Example Amazon SES 消息事件

```
{  
    "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>"
        },
        {
          "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"
}
]
}

```

有关更多信息，请参阅 Amazon SES 开发人员指南 中的 [Lambda 操作](#)。

将 AWS Lambda 与 Amazon SNS 结合使用

您可使用 Lambda 函数处理 Amazon Simple Notification Service 通知。Amazon SNS 支持 Lambda 函数作为发送到主题的消息的目标。您可以将函数订阅到同一账户或其他 AWS 账户中的主题。

Amazon SNS 通过包含消息和元数据的事件异步 (p. 83) 调用您的函数。

Example Amazon SNS 消息事件

```
{  
    "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": "tcc6fafal2yUC6dgZdmrwh1Y4cGa/ebXEkaAi6RibDsvpi+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"  
            }  
        }  
    ]  
}
```

对于异步调用，Lambda 对消息排队并处理重试。如果 Amazon SNS 无法到达 Lambda 或消息被拒绝，Amazon SNS 将在几个小时内以递增的间隔重试。有关详细信息，请参阅 Amazon SNS 常见问题中的[可靠性](#)。

要执行跨账户 Amazon SNS 传输到 Lambda，您需要授权从 Amazon SNS 调用 Lambda 函数。反过来，Amazon SNS 需要允许 Lambda 账户订阅 Amazon SNS 主题。例如，如果 Amazon SNS 主题在账户 A 中且 Lambda 函数在账户 B 中，则这两个账户都必须相互授予对其各自的资源的访问权限。由于并非所有设置跨账户权限的选项均能从 AWS 控制台使用，您可使用 AWS CLI 设置整个过程。

有关更多信息，请参阅 Amazon Simple Notification Service 开发人员指南 中的[使用 Amazon SNS 通知调用 Lambda 函数](#)。

主题

- 教程：将 AWS Lambda 与 Amazon Simple Notification Service 结合使用 (p. 251)
- 示例函数代码 (p. 253)

教程：将 AWS Lambda 与 Amazon Simple Notification Service 结合使用

您可以在一个 AWS 账户中使用 Lambda 函数来订阅单独的 AWS 账户中的 Amazon SNS 主题。在本教程中，您使用 AWS Command Line Interface 执行 AWS Lambda 操作，例如，创建 Lambda 函数，创建 Amazon SNS 主题以及授予权限以允许这两类资源相互访问。

先决条件

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command  
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以[安装 Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

在本教程中，您将使用两个账户。AWS CLI 命令通过以下方式对此进行说明：使用两个[命名配置文件](#)，每个配置为用于不同的账户。如果您使用具有不同名称的配置文件，或者使用默认配置文件和一个命名配置文件，请根据需要修改命令。

创建一个 Amazon SNS 主题

从账户 A，创建源 Amazon SNS 主题。

```
$ aws sns create-topic --name lambda-x-account --profile accountA
```

记下该命令返回的主题 ARN。在将权限添加到 Lambda 函数以订阅主题时，需要使用此 ARN。

创建执行角色

从账户 B 中，创建[执行角色 \(p. 30\)](#)，此角色向您的函数授予访问 AWS 资源的权限。

创建执行角色

1. 打开 IAM 控制台中的“[角色](#)”页面。
2. 选择 Create role (创建角色)。
3. 创建具有以下属性的角色。
 - 可信任的实体 – AWS Lambda。
 - 权限 – AWSLambdaBasicExecutionRole。
 - 角色名称 (角色名称) – **lambda-sns-role**。

AWSLambdaBasicExecutionRole 策略具有函数将日志写入 CloudWatch Logs 所需的权限。

创建 Lambda 函数

从账户 B 中，创建处理来自 Amazon SNS 的事件的函数。以下示例代码接收 Amazon SNS 事件输入并对其所包含的消息进行处理。为了展示这个过程，代码会将一些传入的事件数据写入 CloudWatch Logs。

Note

有关使用其他语言的示例代码，请参阅 [示例函数代码 \(p. 253\)](#)。

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");
};
```

创建函数

1. 将示例代码复制到名为 `index.js` 的文件中。
2. 创建部署程序包。

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

3. 使用 `create-function` 命令创建 Lambda 函数。

```
$ aws lambda create-function --function-name SNS-X-Account \
--zip-file fileb://function.zip --handler index.handler --runtime nodejs12.x \
--role arn:aws:iam::012345678901B:role/service-role/lambda-sns-execution-role \
--timeout 60 --profile accountB
```

记下该命令返回的函数 ARN。在添加权限以允许 Amazon SNS 调用函数时，需要使用此 ARN。

设置跨账户权限

从账户 A 中，授予账户 B 订阅主题的权限：

```
$ aws sns add-permission --label lambda-access --aws-account-id 12345678901B \
--topic-arn arn:aws:sns:us-east-2:12345678901A:lambda-x-account \
--action-name Subscribe ListSubscriptionsByTopic Receive --profile accountA
```

从账户 B 中，添加 Lambda 权限以允许从 Amazon SNS 进行调用。

```
$ aws lambda add-permission --function-name SNS-X-Account \
--source-arn arn:aws:sns:us-east-2:12345678901A:lambda-x-account \
--statement-id sns-x-account --action "lambda:InvokeFunction" \
--principal sns.amazonaws.com --profile accountB
{
    "Statement": "{\"Condition\": {\"ArnLike\": {\"AWS:SourceArn\": \"arn:aws:lambda:us-east-2:12345678901B:function:SNS-X-Account\"}}, \"Action\": [\"lambda:InvokeFunction\"], \"Resource\": \"arn:aws:lambda:us-east-2:01234567891A:function:SNS-X-Account\", \"Effect\": \"Allow\", \"Principal\": {\"Service\": \"sns.amazonaws.com\"}, \"Sid\": \"sns-x-account1\"}"
}
```

在添加策略时，请不要使用 `--source-account` 参数将源账户添加到 Lambda 策略。Amazon SNS 事件源不支持源账户，并且将导致访问被拒绝。

创建订阅

从账户 B 中，将 Lambda 函数订阅到主题。在将消息发送到账户 A 中的 lambda-x-account 主题时，Amazon SNS 将调用账户 B 中的 SNS-X-Account 函数。

```
$ aws sns subscribe --protocol lambda \
--topic-arn arn:aws:sns:us-east-2:12345678901A:lambda-x-account \
--notification-endpoint arn:aws:lambda:us-east-2:12345678901B:function:SNS-X-Account \
--profile accountB
{
    "SubscriptionArn": "arn:aws:sns:us-east-2:12345678901A:lambda-x-
account:5d906xxxx-7c8x-45dx-a9dx-0484e31c98xx"
}
```

输出包含主题订阅的 ARN。

测试订阅

从账户 A 中测试订阅。在一个文本文件中键入 Hello World，并将该文本文件另存为 message.txt。然后运行以下命令：

```
$ aws sns publish --message file://message.txt --subject Test \
--topic-arn arn:aws:sns:us-east-2:12345678901A:lambda-x-account \
--profile accountA
```

这将返回一个具有唯一标识符的消息 ID，指示 Amazon SNS 服务已接受消息。然后，Amazon SNS 会尝试将它传输给主题订阅者。或者，您可以直接将 JSON 字符串提供给 message 参数，但使用文本文件将允许消息中出现换行符。

要了解有关 Amazon SNS 的更多信息，请参阅[什么是 Amazon Simple Notification Service](#)。

示例函数代码

示例代码具有以下语言。

主题

- [Node.js 8 \(p. 253\)](#)
- [Java 11 \(p. 254\)](#)
- [转到 \(p. 254\)](#)
- [Python 3 \(p. 255\)](#)

Node.js 8

以下示例处理来自 Amazon SNS 的消息并记录其内容。

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");  
};
```

压缩示例代码以创建部署程序包。有关说明，请参阅[Node.js 中的 AWS Lambda 部署程序包 \(p. 268\)](#)。

Java 11

以下示例处理来自 Amazon SNS 的消息并记录其内容。

Example LambdaWithSNS.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;  
    }  
}
```

附属物

- [aws-lambda-java-core](#)
- [aws-lambda-java-events](#)

使用 Lambda 库依赖项构建代码以创建部署程序包。有关说明，请参阅[Java 中的 AWS Lambda 部署程序包 \(p. 302\)](#)。

转到

以下示例处理来自 Amazon SNS 的消息并记录其内容。

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)
}
```

使用 `go build` 构建可执行文件并创建部署程序包。有关说明，请参阅[Go 中的 AWS Lambda 部署程序包 \(p. 330\)](#)。

Python 3

以下示例处理来自 Amazon SNS 的消息并记录其内容。

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
```

压缩示例代码以创建部署程序包。有关说明，请参阅[Python 中的 AWS Lambda 部署程序包 \(p. 279\)](#)。

将 AWS Lambda 与 Amazon SQS 结合使用

您可以使用 AWS Lambda 函数处理 Amazon Simple Queue Service (Amazon SQS) 队列中的消息。[Lambda 事件源映射 \(p. 90\)](#) 支持[标准队列](#)和[先进先出 \(FIFO\) 队列](#)。在 Amazon SQS 中，您可以通过将来自一个应用程序组件的任务发送到一个队列中并异步处理它们来进行分载。

Lambda 轮询该队列，并通过一个包含队列消息的事件来[同步 \(p. 81\)](#)调用您的函数。Lambda 读取批次中的消息，并为每个批次调用一次函数。当您的函数成功处理一个批次后，Lambda 就会将其消息从队列中删除。以下示例显示了包含两条消息的批次事件。

Example Amazon SQS 消息事件（标准队列）

```
{
    "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": "AIDAIEENQZJLO23YVJ4VO",
        "ApproximateFirstReceiveTimestamp": "1545082650649"
    },
    "messageAttributes": {},
    "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
    "eventSource": "aws:sqs",
    "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
    "awsRegion": "us-east-2"
}
]
}

```

对于 FIFO 队列，记录包含与重复数据消除和顺序相关的其他属性。

Example Amazon SQS 消息事件 (FIFO 队列)

```

{
    "Records": [
        {
            "messageId": "11d6ee51-4cc7-4302-9e22-7cd8afdaadf5",
            "receiptHandle": "AQEBBX8nesZExmkhsmZeyIE8iQAMig7qw...",
            "body": "Test message.",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1573251510774",
                "SequenceNumber": "18849496460467696128",
                "MessageGroupId": "1",
                "SenderId": "AIDAI023YVJENQZJOL4VO",
                "MessageDeduplicationId": "1",
                "ApproximateFirstReceiveTimestamp": "1573251510774"
            },
            "messageAttributes": {},
            "md5OfBody": "e4e68fb7bd0e697a0ae8f1bb342846b3",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:fifo fifo",
            "awsRegion": "us-east-2"
        }
    ]
}

```

当 Lambda 读取批次时，消息将保留在队列中，但会在队列**可见性超时**的长度内隐藏。如果您的函数成功处理一个批次，Lambda 会将其消息从队列中删除。如果您的函数**受到限制** (p. 94)、返回错误或没有响应，则消息将再次可见。所有批处理失败的消息都会返回队列中，因此您的函数代码必须能够多次处理同一条消息，而不会产生副作用。

扩展和处理

对于标准队列，Lambda 使用**长轮询**来轮询一个队列，直到它变为活动状态。当消息可用时，Lambda 最多可读取 5 个批次并将其发送到您的函数。如果仍有消息可用，则 Lambda 增加批量读取的进程数，最多每分钟增加 60 个实例。事件源映射可以同时处理的最大批次数为 1000。

对于 FIFO 队列，Lambda 按照接收消息的顺序向函数发送消息。向 FIFO 队列发送消息时，您可以指定[消息组 ID](#)。Amazon SQS 确保同一组中的消息按顺序传递到 Lambda。Lambda 将消息排列到组中，一次仅为一个组发送一批消息。如果函数返回错误，则在对受影响的消息尝试了所有重试之后，Lambda 会从同一个组收到其他消息。

您的函数可以在并发范围内扩展到活动消息组的数量。有关更多信息，请参阅 AWS 计算博客上的[作为事件源的 SQS FIFO](#)。

配置队列以便与 Lambda 一起使用

创建一个[SQS 队列](#)，用作您的 Lambda 函数的事件源。然后将队列配置为可使您的 Lambda 函数有时间处理每批事件——并使 Lambda 在扩展时出现限制错误时能够重试。

为使您的函数有时间处理每批记录，请将源队列的可见性超时至少设置为是您在函数上配置的[超时 \(p. 47\)](#)的 6 倍。额外时间可以使 Lambda 在您的函数处理前一批次时，如果您的函数执行受到限制，它可以重试。

如果消息多次处理失败，Amazon SQS 可以将其发送到[死信队列](#)。当您的函数返回错误时，Lambda 将其留在队列中。在发生可见性超时之后，Lambda 重新接收消息。要在多次接收之后将消息发送到第二个队列，请在源队列上配置死信队列。

Note

确保在源队列上配置死信队列，而不是在 Lambda 函数上配置。您在函数上配置的死信队列用于函数的[异步调用队列 \(p. 83\)](#)，而不是用于事件源队列。

如果您的函数返回错误，或者由于处于最大并发而无法调用，则处理可能会成功，但需要额外尝试。要在将消息发送到死信队列之前给予更好的处理机会，请将源队列重新驱动策略的 `maxReceiveCount` 设置为至少 5。

执行角色权限

Lambda 需要以下权限来管理您的 Amazon SQS 队列中的消息。将这些权限添加到您的函数的执行角色中。

- `sqs:ReceiveMessage`
- `sqs:DeleteMessage`
- `sqs:GetQueueAttributes`

有关更多信息，请参阅 [AWS Lambda 执行角色 \(p. 30\)](#)。

将队列配置为事件源

创建事件源映射以指示 Lambda 将队列中的项目发送到 Lambda 函数。您可以创建多个事件源映射，以使用单个函数处理来自多个队列的项目。当 Lambda 调用目标函数时，事件可以包含多个项目（多达可配置的最大批处理大小）。

要在 Lambda 控制台中将您的函数配置为从 Amazon SQS 读取，请创建 SQS 触发器。

创建触发器

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Designer 下方，选择 Add trigger (添加触发器)。
4. 选择触发器类型。

5. 配置所需选项，然后选择 Add (添加)。

Lambda 支持 Amazon SQS 事件源的以下选项。

事件源选项

- SQS 队列 – 要从其读取记录的 Amazon SQS 队列。
- 批处理大小 – 从队列中读取的每个批次的项目数，最多 10 个。如果 Lambda 从队列读取的批次具有较少的项目，则事件可能包含较少的项目。
- 已启用 – 禁用事件源以停止处理项目。

之后，要管理事件源配置，请在设计器中选择触发器。

配置您的函数超时，以允许有足够的时间来处理整个批次的项目。如果项目处理需要很长时间，请选择一个较小的批处理大小。大批量处理可以提高非常快速或拥有大量开销的工作负载的效率。但是，如果您的函数返回错误，则批处理中的所有项目都将返回到队列中。如果您在函数上配置预留并发 (p. 54)，请将最小并发执行数设置为 5，以降低在 Lambda 调用函数时出现限制错误的几率。

事件源映射 API

要使用 AWS CLI 或 AWS 开发工具包管理事件源映射，请使用以下 API 操作：

- [CreateEventSourceMapping \(p. 415\)](#)
- [ListEventSourceMappings \(p. 492\)](#)
- [GetEventSourceMapping \(p. 451\)](#)
- [UpdateEventSourceMapping \(p. 547\)](#)
- [DeleteEventSourceMapping \(p. 432\)](#)

以下示例使用 AWS CLI 将名为 my-function 的函数映射到由 Amazon 资源名称 (ARN) 指定的 Amazon SQS 队列，批处理大小为 5。

```
$ aws lambda create-event-source-mapping --function-name my-function --batch-size 5 \
--event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue
{
    "UUID": "2b733gdc-8ac3-cdf5-af3a-1827b3b11284",
    "BatchSize": 5,
    "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"
}
```

教程：将 AWS Lambda 与 Amazon Simple Queue Service 结合使用

在本教程中，您将创建一个 Lambda 函数来处理来自 [Amazon SQS](#) 队列的消息。

先决条件

本教程假设您对基本 Lambda 操作和 Lambda 控制台有一定了解。如果尚不了解，请按照[开始使用 AWS Lambda \(p. 3\)](#)中的说明创建您的第一个 Lambda 函数。

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command  
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以 [安装 Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

创建执行角色

[创建执行角色 \(p. 30\)](#)，向您的函数授予访问 AWS 资源的权限。

创建执行角色

1. 打开 IAM 控制台中的“[角色](#)”页面。
2. 选择 Create role (创建角色)。
3. 创建具有以下属性的角色。
 - 可信任的实体 – AWS Lambda。
 - 权限 – AWSLambdaSQSQueueExecutionRole。
 - 角色名称 (角色名称) – **lambda-sqs-role**。

AWSLambdaSQSQueueExecutionRole 策略具有该函数从 Amazon SQS 中读取项目并将日志写入 CloudWatch Logs 所需的权限。

创建函数

以下示例代码接收 Amazon SQS 事件输入并对其所包含的消息进行处理。为了展示这个过程，代码会将一些传入的事件数据写入 CloudWatch Logs。

Note

有关使用其他语言的示例代码，请参阅 [示例 Amazon SQS 函数代码 \(p. 261\)](#)。

Example index.js

```
exports.handler = async function(event, context) {  
    event.Records.forEach(record => {  
        const { body } = record;  
        console.log(body);  
    });  
    return {};  
}
```

创建函数

1. 将示例代码复制到名为 `index.js` 的文件中。
2. 创建部署程序包。

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

3. 使用 `create-function` 命令创建 Lambda 函数。

```
$ 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
```

测试函数。

使用 `invoke` AWS Lambda CLI 命令和示例 Amazon Simple Queue Service 事件手动调用 Lambda 函数。

如果该处理程序正常返回并且没有异常，则 Lambda 认为该消息得到成功处理并开始读取队列中的新消息。在成功处理消息后，它会立即从队列中删除。如果该处理程序引发异常，则 Lambda 认为消息的输入未得到处理，并用相同的批量消息调用该函数。

1. 将以下 JSON 复制到文件中并将其保存为 `input.txt`。

```
{
    "Records": [
        {
            "messageId": "059f36b4-87a3-44ab-83d2-661975830a7d",
            "receiptHandle": "AQEBwJnKyrHigUMZj6rYigCgxlaS3SLy0a...",
            "body": "test",
            "attributes": {
                "ApproximateReceiveCount": "1",
                "SentTimestamp": "1545082649183",
                "SenderId": "AIDAIEQZJOL023YVJ4VO",
                "ApproximateFirstReceiveTimestamp": "1545082649185"
            },
            "messageAttributes": {},
            "md5OfBody": "098f6bcd4621d373cade4e832627b4f6",
            "eventSource": "aws:sqs",
            "eventSourceARN": "arn:aws:sqs:us-east-2:123456789012:my-queue",
            "awsRegion": "us-east-2"
        }
    ]
}
```

2. 执行下面的 `invoke` 命令。

```
$ aws lambda invoke --function-name ProcessSQSRecord \
--payload file://input.txt outputfile.txt
```

3. 在 `outputfile.txt` 文件中验证输出。

创建 Amazon SQS 队列

创建一个可由 Lambda 函数用作事件源的 Amazon SQS 队列。

创建队列

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon SQS 控制台：<https://console.aws.amazon.com/sqs/>。
2. 在 Amazon SQS 控制台中，创建一个队列。
3. 记下或以其他方式记录标识的队列 ARN (Amazon 资源名称)。在下一步中将该队列与您的 Lambda 函数关联时，您将需要此类信息。

在 AWS Lambda 中创建事件源映射。此事件源映射将 Amazon SQS 队列与您的 Lambda 函数关联。创建此事件源映射后，AWS Lambda 即开始轮询该队列。

测试端到端体验。在您执行队列更新时，Amazon Simple Queue Service 会将消息写入队列。AWS Lambda 将轮询该队列，检测新记录并通过向该函数传递事件（在本示例中为 Amazon SQS 消息）来代表您执行 Lambda 函数。

配置事件源

要创建指定的 Amazon SQS 队列与 Lambda 函数之间的映射，请运行以下 AWS CLI `create-event-source-mapping` 命令。在此命令执行后，记下或以其他方式记录 UUID。在任何其他命令中，如选择删除事件源映射时，您都需要该 UUID 来引用事件源映射。

```
$ aws lambda create-event-source-mapping --function-name ProcessSQSRecord --batch-size 10 \
--event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue
```

您可以通过运行以下命令获取事件源映射的列表。

```
$ aws lambda list-event-source-mappings --function-name ProcessSQSRecord \
--event-source-arn arn:aws:sqs:us-east-2:123456789012:my-queue
```

该列表返回您创建的所有事件源映射，而对于每个映射，它都显示 `LastProcessingResult` 等信息。该字段用于在出现任何问题时提供信息性消息。`No records processed`（指示 AWS Lambda 未开始轮询或队列中没有任何记录）和 `OK`（指示 AWS Lambda 已成功读取队列中的记录并调用了您的 Lambda 函数）等值表示未出现任何问题。如果出现问题，您将收到一条错误消息。

测试设置

现在，可以按以下方式测试设置：

1. 在 Amazon SQS 控制台中，将消息发送到队列。Amazon SQS 会将这些操作记录写入队列。
2. AWS Lambda 轮询该队列，当检测到有更新时，它会通过传递在队列中发现的事件数据来调用您的 Lambda 函数。
3. 您的函数将执行并在 Amazon CloudWatch 中创建日志。您可以验证 Amazon CloudWatch 控制台中报告的日志。

示例 Amazon SQS 函数代码

示例代码具有以下语言。

主题

- [Node.js \(p. 261\)](#)
- [Java \(p. 262\)](#)
- [C# \(p. 262\)](#)
- [转到 \(p. 263\)](#)
- [Python \(p. 264\)](#)

Node.js

以下是将 Amazon SQS 事件消息作为输入接收并对其进行处理的示例代码。为了展示这个过程，代码会将一些传入的事件数据写入 CloudWatch Logs。

Example index.js (Node.js 8)

```
exports.handler = async function(event, context) {
```

```
event.Records.forEach(record => {
  const { body } = record;
  console.log(body);
});
return {};
}
```

Example index.js (Node.js 6)

```
event.Records.forEach(function(record) {
  var body = record.body;
  console.log(body);
});
callback(null, "message");
};
```

压缩示例代码以创建部署程序包。有关说明，请参阅[Node.js 中的 AWS Lambda 部署程序包 \(p. 268\)](#)。

Java

以下是将 Amazon SQS 事件消息作为输入接收并对其进行处理的示例 Java 代码。为了展示这个过程，代码会将一些传入的事件数据写入 CloudWatch Logs。

在代码中，`handleRequest` 是处理程序。该处理程序使用了在 `aws-lambda-java-events` 库中定义的预定义 `SQSEvent` 类。

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;
    }
}
```

附属物

- `aws-lambda-java-core`
- `aws-lambda-java-events`

使用 Lambda 库依赖项构建代码以创建部署程序包。有关说明，请参阅[Java 中的 AWS Lambda 部署程序包 \(p. 302\)](#)。

C#

以下是将 Amazon SQS 事件消息作为输入接收并对其进行处理的示例 C# 代码。为了展示这个过程，代码会将一些传入的事件数据写入控制台。

在代码中，`handleRequest` 是处理程序。该处理程序使用了在 `AWS.Lambda.SQSEvents` 库中定义的预定类 `SQSEvent`。

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.");

            return $"Processed {sqsEvent.Records.Count} records.";
        }
    }
}
```

使用以上示例替换.NET Core 中的 `Program.cs` 有关说明，请参阅[C# 中的 AWS Lambda 部署程序包 \(p. 342\)](#)。

转到

以下是将 Amazon SQS 事件消息作为输入接收并对其进行处理的示例 Go 代码。为了展示这个过程，代码会将一些传入的事件数据写入 CloudWatch Logs。

在代码中，`handler` 是处理程序。该处理程序使用了在 `aws-lambda-go-events` 库中定义的预定类 `SQSEvent`。

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)
}
```

使用 `go build` 构建可执行文件并创建部署程序包。有关说明，请参阅[Go 中的 AWS Lambda 部署程序包 \(p. 330\)](#)。

Python

以下是将 Amazon SQS 记录作为输入接受并对其进行处理的示例 Python 代码。为了展示这个过程，代码会将一些传入的事件数据写入 CloudWatch Logs。

按照说明创建 AWS Lambda 函数部署程序包。

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))
```

压缩示例代码以创建部署程序包。有关说明，请参阅[Python 中的 AWS Lambda 部署程序包 \(p. 279\)](#)。

Amazon SQS 应用程序的 AWS SAM 模板

您可以使用 [AWS SAM](#) 构建此应用程序。要了解有关创建 AWS SAM 模板的更多信息，请参阅 AWS 无服务器应用程序模型 开发人员指南 中的 [AWS SAM 模板基础知识](#)。

下面是[教程 \(p. 258\)](#)中 Lambda 应用程序的示例 AWS SAM 模板。将以下文本复制到 .yaml 文件中，并将其保存到您之前创建的 ZIP 程序包旁。请注意，Handler 和 Runtime 参数值应与上一节中创建函数时所用的参数值匹配。

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
```

有关如何使用程序包和部署命令打包和部署无服务器应用程序的信息，请参阅 AWS 无服务器应用程序模型 开发人员指南 中的[部署无服务器应用程序](#)。

使用 Node.js 构建 Lambda 函数

您可以在 AWS Lambda 中运行 采用 Node.js 的 JavaScript 代码。Lambda 为执行代码来处理事件的 Node.js 提供[运行时 \(p. 108\)](#)。您的代码在包含 AWS SDK for JavaScript 的环境中运行，其中包含来自您管理的 AWS Identity and Access Management (IAM) 角色的凭证。

Lambda 支持以下 Node.js 运行时。

Node.js 运行时

名称	标识符	适用于 JavaScript 的 AWS 开发工具包	操作系统	
Node.js 12	nodejs12.x	2.631.0	Amazon Linux 2	
Node.js 10	nodejs10.x	2.631.0	Amazon Linux 2	

Lambda 函数使用[执行角色 \(p. 30\)](#)来获取将日志写入 Amazon CloudWatch Logs 以及访问其他服务和资源的权限。如果您还没有函数开发的执行角色，请创建一个。

创建执行角色

1. 打开 IAM 控制台中的[“角色”页面](#)。
2. 选择 Create role (创建角色)。
3. 创建具有以下属性的角色。
 - 可信任的实体 – Lambda。
 - 权限 – AWSLambdaBasicExecutionRole。
 - 角色名称 (角色名称) – **lambda-role**。

AWSLambdaBasicExecutionRole 策略具有函数将日志写入 CloudWatch Logs 所需的权限。

您可以稍后向此角色添加权限，或将其与特定于单一函数的其他角色交换。

创建 Node.js 函数

1. 打开 [Lambda 控制台](#)。
2. 选择创建函数。
3. 配置以下设置：
 - 名称 – **my-function**。
 - 运行时 – Node.js 12.x。
 - 角色 – 选择现有角色。
 - 现有角色 – **lambda-role**。
4. 选择创建函数。
5. 要配置测试事件，请选择测试。

6. 对于事件名称，输入 **test**。
7. 选择创建。
8. 要执行函数，请选择测试。

控制台将创建一个带有单个名为 `index.js` 的源文件的 Lambda 函数。您可以在[内置代码编辑器 \(p. 5\)](#)中编辑此文件并添加更多文件。要保存您的更改，请选择 Save。然后，要运行代码，请选择 Test (测试)。

Note

Lambda 控制台使用 AWS Cloud9 在浏览器中提供集成开发环境。您还可以使用 AWS Cloud9 在您自己的环境中开发 Lambda 函数。有关更多信息，请参阅《AWS Cloud9 用户指南》中的[使用 AWS Lambda 函数](#)。

`index.js` 文件导出一个名为 `handler` 的函数，此函数接受事件对象和上下文对象。这是在调用函数时 Lambda 调用的[处理程序函数 \(p. 266\)](#)。Node.js 函数运行时从 Lambda 获取调用事件并将它们传递到处理程序。在函数配置中，处理程序值为 `index.handler`。

每次保存函数代码时，Lambda 控制台都会创建一个部署程序包，它是一个包含函数代码的 ZIP 存档。随着函数开发的进行，您需要将函数代码存储在源代码控制中、添加库和实现部署自动化。首先，通过命令行[创建部署程序包 \(p. 268\)](#)并更新代码。

除了调用事件之外，函数运行时还将上下文对象传递给处理程序。[上下文对象 \(p. 270\)](#)包含有关调用、函数和执行环境的其他信息。环境变量中提供了更多信息。

您的 Lambda 函数附带了 CloudWatch Logs 日志组。函数运行时将发送有关对 CloudWatch Logs 的每个调用的详细信息。它会中继调用期间[函数输出的任何日志 \(p. 271\)](#)。如果您的函数[返回错误 \(p. 274\)](#)，则 Lambda 将为错误设置格式，并将其返回给调用方。

主题

- [Node.js 中的 AWS Lambda 函数处理程序 \(p. 266\)](#)
- [Node.js 中的 AWS Lambda 部署程序包 \(p. 268\)](#)
- [Node.js 中的 AWS Lambda 上下文对象 \(p. 270\)](#)
- [Node.js 中的 AWS Lambda 函数日志记录 \(p. 271\)](#)
- [Node.js 中的 AWS Lambda 函数错误 \(p. 274\)](#)
- [在 AWS Lambda 中检测 Node.js 代码 \(p. 275\)](#)

Node.js 中的 AWS Lambda 函数处理程序

该处理程序是 Lambda 函数中处理事件的方法。当您调用某个函数时，[运行时 \(p. 108\)](#)将运行该处理程序方法。当处理程序退出或返回响应时，它可用于处理另一个事件。

以下示例函数记录事件对象的内容并返回日志的位置。

Example index.js

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

当您[配置函数 \(p. 47\)](#)时，处理程序设置的值是文件的名称，也是导出的处理程序模块的名称（由点分隔）。控制台中的默认设置；在本指南的示例中，此为 `index.handler`。这表示由 `index.js` 导出的 `handler` 模块。

运行时会将三个参数传递到处理程序方法。第一个参数是 `event` 对象，它包含来自调用程序的信息。调用程序在调用 [Invoke \(p. 482\)](#) 时将该信息作为 JSON 格式字符串传递，运行时将它转换为对象。当 AWS 服务调用您的函数时，事件结构[因服务而异 \(p. 138\)](#)。

第二个参数是[上下文对象 \(p. 270\)](#)，该对象包含有关调用、函数和执行环境的信息。在前面的示例中，函数将从上下文对象获取[日志流 \(p. 271\)](#)的名称，然后将其返回到调用方。

第三个参数 `callback` 是一个函数，您可以在[非异步处理程序 \(p. 267\)](#)中调用它来发送响应。回调函数采用两个参数：一个 `Error` 和一个响应。当您调用它时，Lambda 等待事件循环为空，然后将响应或错误返回给调用者。响应对象必须与 `JSON.stringify` 兼容。

对于异步处理程序，会将响应、错误或承诺返回到运行时，而不是使用 `callback`。

异步处理程序

对于异步处理程序，您可以分别使用 `return` 和 `throw` 来发送响应和错误。函数必须使用 `async` 关键字来使用这些方法返回响应或错误。

如果您的代码执行了一个异步任务，请返回一个承诺以确保该代码完成运行。当您完成或拒绝该承诺时，Lambda 会向调用方发送响应或错误。

Example index.js 文件 – 包含异步处理程序和承诺的 HTTP 请求

```
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
}
```

对于返回了一个承诺的库，您可以直接将该承诺返回到运行时。

Example index.js 文件 – 包含异步处理程序和承诺的 AWS 开发工具包

```
const AWS = require('aws-sdk')
const s3 = new AWS.S3()

exports.handler = async function(event) {
  return s3.listBuckets().promise()
}
```

非异步处理程序

以下示例函数检查 URL 并向调用方返回状态代码。

Example index.js 文件 – 包含回调的 HTTP 请求

```
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))
  })
}
```

对于非异步处理程序，函数将继续执行，直到[事件循环](#)为空或函数超时。在完成所有事件循环任务之前，不会将响应发送给调用程序。如果函数超时，则会返回错误。您可以通过将[context.callbackWaitsForEmptyEventLoop \(p. 270\)](#)设置为 false，来将运行时配置为立即发送响应。

在以下示例中，来自 Amazon S3 的响应将在可用时立即返回到调用方。针对事件循环运行的超时将被冻结，并在下次调用该函数时继续运行。

Example index.js 文件 – [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)
}
```

Node.js 中的 AWS Lambda 部署程序包

部署程序包是包含函数代码和依赖项的 ZIP 存档。如果您使用 Lambda API 管理函数，或者需要包含 AWS 开发工具包以外的库和依赖项，则需要创建部署程序包。您可以将程序包直接上传到 Lambda，也可以使用 Amazon S3 存储桶、然后再将其上传到 Lambda。如果部署包大于 50 MB，则必须使用 Amazon S3。

如果您使用 Lambda [控制台编辑器 \(p. 5\)](#) 编写您的函数，则控制台会管理部署程序包。如果您不需要添加任何库，则可以使用此方法。您也可以使用此方法更新在部署程序包中已经存在库的函数，前提是总大小不超过 3 MB。

Note

为了减小部署程序包的大小，请将函数的依赖项打包到层中。层可让您独立管理依赖项，可以供多个函数使用，并且可以与其他账户共享。有关详细信息，请参阅[AWS Lambda 层 \(p. 68\)](#)。

小节目录

- [更新没有依赖项的函数 \(p. 268\)](#)
- [更新具有额外依赖项的函数 \(p. 269\)](#)

更新没有依赖项的函数

要使用 Lambda API 更新函数，请使用 [UpdateFunctionCode \(p. 553\)](#) 操作。创建包含函数代码的存档，然后使用 AWS CLI 上传该存档。

更新没有依赖项的 Node.js 函数

1. 创建 ZIP 存档。

```
~/my-function$ zip function.zip index.js
```

2. 使用 update-function-code 命令上传程序包。

```
~/my-function$ aws lambda update-function-code --function-name my-function --zip-file fileb://function.zip
{
    "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": "QfOhMc1I2di6YFMi9aXm3JtGTmcDbjniEuiYonYptAk=",
    "Version": "$LATEST",
    "TracingConfig": {
        "Mode": "Active"
    },
    "RevisionId": "983ed1e3-ca8e-434b-8dc1-7d72ebadd83d",
    ...
}
```

更新具有额外依赖项的函数

如果您的函数依赖于 SDK for JavaScript 之外的库，请使用 [npm](#) 将它们安装到本地目录中，并将它们包含在部署程序包中。如果您需要比[运行时上包含的版本 \(p. 265\)](#)更新的版本，或者需要确保该版本在将来不会变化，则还可以包含 SDK for JavaScript。

使用依赖项更新 Node.js 函数

1. 使用 `npm install` 命令在 `node_modules` 目录中安装库。

```
~/my-function$ npm install aws-xray-sdk
```

这将创建一个类似于下面的文件夹结构。

```
~/my-function
### index.js
### node_modules
###  async
###  async-listener
###  atomic-batcher
###  aws-sdk
###  aws-xray-sdk
###  aws-xray-sdk-core
```

2. 创建一个包含您的项目文件夹内容的 ZIP 文件。

```
~/my-function$ zip -r function.zip .
```

3. 使用 `update-function-code` 命令上传程序包。

```
~/my-function$ aws lambda update-function-code --function-name my-function --zip-file fileb://function.zip
{
    "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",
...
}
```

除了代码和库之外，您的部署程序包还可以包含可执行文件和其他资源。有关更多信息，请参阅下列内容：

- 在 AWS Lambda 中运行可执行文件
- 在 AWS Lambda 中使用程序包和本机 nodejs 模块

Node.js 中的 AWS Lambda 上下文对象

当 Lambda 运行您的函数时，它会将上下文对象传递到[处理程序 \(p. 266\)](#)。此对象提供的方法和属性包含有关调用、函数和执行环境的信息。

上下文方法

- `getRemainingTimeInMillis()` – 返回在执行超时以前剩余的毫秒数。

上下文属性

- `functionName` – Lambda 函数的名称。
- `functionVersion` – 函数的[版本 \(p. 63\)](#)。
- `invokedFunctionArn` – 用于调用函数的 Amazon 资源名称 (ARN)。指示调用方是否已指定版本或别名。
- `memoryLimitInMB` – 为函数分配的内存量。
- `awsRequestId` – 调用请求的标识符。
- `logGroupName` – 函数的日志组。
- `logStreamName` – 函数实例的日志流。
- `identity` – (移动应用程序) 有关授权请求的 Amazon Cognito 身份的信息。
 - `cognitoIdentityId` – 已经过身份验证的 Amazon Cognito 身份。
 - `cognitoIdentityPoolId` – 授权调用的 Amazon Cognito 身份池。
- `clientContext` – (移动应用程序) 由客户端应用程序向 Lambda 提供的客户端上下文。
 - `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 – 由移动应用程序设置的自定义值。
- callbackWaitsForEmptyEventLoop – 设置为 false 可在[回调 \(p. 267\)](#)执行时立即发送响应，而不是等待 Node.js 事件循环成空。如果为 false，则任何未完成的事件将在下次调用期间继续运行。

以下示例函数记录了上下文信息并返回了日志的位置。

Example index.js 文件

```
exports.handler = async function(event, context) {
    console.log('Remaining time: ', context.getRemainingTimeInMillis())
    console.log('Function name: ', context.functionName)
    return context.logStreamName
}
```

Node.js 中的 AWS Lambda 函数日志记录

您的 Lambda 函数带有一个 CloudWatch Logs 日志组，其中包含您的函数的每个实例的日志流。运行时会将每个调用的详细信息发送到该日志流，然后中继日志和来自您的函数代码的其他输出。

要从函数代码输出日志，您可以使用[控制台对象](#)的方法或使用写入到 `stdout` 或 `stderr` 的任何日志记录库。以下示例记录环境变量和事件对象的值。

Example index.js 文件 – 日志记录

```
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 日志格式

```
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-0a8c7fd02xmpl480aed55ef0 SegmentId: 3d752xmpl1bbe37e Sampled: true
```

Node.js 运行时记录每次调用的 START、END 和 REPORT 行。它向函数记录的每个条目添加时间戳、请求 ID 和日志级别。报告行提供了以下详细信息。

报告日志

- RequestId – 调用的唯一请求 ID。
- Duration (持续时间) – 函数的处理程序方法处理事件所花费的时间。
- Billed Duration (计费持续时间) – 针对调用计费的时间量。
- Memory Size (内存大小) – 分配给函数的内存量。
- Max Memory Used (最大内存使用量) – 函数使用的内存量。
- Init Duration (初始持续时间) – 对于提供的第一个请求，为运行时在处理程序方法外部加载函数和运行代码所花费的时间。
- XRAY TraceId – 对于跟踪的请求，为 [AWS X-Ray 跟踪编码 \(p. 371\)](#)。
- SegmentId – 对于跟踪的请求，为 X-Ray 分段 ID。
- Sampled (采样) – 对于跟踪的请求，为采样结果。

您可以在 Lambda 控制台中、在 CloudWatch Logs 控制台中或从命令行查看日志。

小节目录

- [在 AWS 管理控制台中查看日志 \(p. 272\)](#)
- [使用 AWS CLI \(p. 272\)](#)
- [删除日志 \(p. 274\)](#)

在 AWS 管理控制台中查看日志

当您在函数配置页上测试函数时，Lambda 控制台会显示日志输出。要查看所有调用的日志，请使用 CloudWatch Logs 控制台。

查看 Lambda 函数的日志

1. 打开 [CloudWatch 控制台的日志页面](#)。
2. 选择您的函数 (/aws/lambda/**function-name**) 的日志组。
3. 选择列表中的第一个流。

每个日志流对应一个[函数实例 \(p. 109\)](#)。当您更新函数以及创建更多实例以处理多个并发调用时，会显示新的流。要找到特定调用的日志，您可以使用 X-Ray 分析您的函数并在跟踪中记录有关请求和日志流的详细信息。如需将日志和跟踪与 X-Ray 相关联的示例应用程序，请参阅[AWS Lambda 错误处理器示例应用程序 \(p. 128\)](#)。

使用 AWS CLI

要从命令行获取调用的日志，请使用 --log-type 选项。响应包含一个 LogResult 字段，该字段包含来自调用的多达 4 KB 的 base64 编码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult": "U1RBULQgUmVxdWVzdElkOia4N2QwNDRIoC1mMTU0LTexZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
```

}

您可以使用 base64 实用程序来解码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
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: 100 ms      Memory Size: 128 MB      Max Memory Used: 73 MB
```

base64 实用工具在 Linux、macOS 和 [Ubuntu on Windows](#) 上可用。对于 macOS，命令为 base64 -D。

要从命令行获取完整的日志事件，您可以在函数输出中包含日志流名称，如上例中所示。以下示例脚本调用名为 my-function 的函数并下载最后 5 个日志事件。

Example get-logs.sh 脚本

此示例要求 my-function 返回日志流 ID。

```
#!/bin/bash
aws lambda invoke --function-name my-function --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
```

此脚本使用 sed 从输出文件中删除引号，并休眠 15 秒以等待日志可用。输出包括来自 Lambda 的响应，以及来自 get-log-events 命令的输出。

```
$ ./get-logs.sh
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
{
  "events": [
    {
      "timestamp": 1559763003171,
      "message": "START RequestId: 4ce9340a-b765-490f-ad8a-02ab3415e2bf Version:
$LATEST\n",
      "ingestionTime": 1559763003309
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tnfo\tENVIRONMENT VARIABLES\r{\r  \"AWS_LAMBDA_FUNCTION_VERSION\": \"$LATEST\", \r ...",
      "ingestionTime": 1559763018353
    },
    {
      "timestamp": 1559763003173,
      "message": "2019-06-05T19:30:03.173Z\t4ce9340a-b765-490f-ad8a-02ab3415e2bf
\tnfo\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: 100 ms \tMemory Size: 128 MB\tMax Memory Used: 75 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

删除日志

在删除函数时，不会自动删除日志组。为避免存储无限的日志，请删除日志组或者[配置保留期](#)，在该期限后会自动删除日志。

Node.js 中的 AWS Lambda 函数错误

当您的代码引发错误时，Lambda 将生成错误的 JSON 表示形式。此错误文档会出现在调用日志中，对于同步调用，它出现在输出中。

Example index.js 文件 – 引用错误

```
exports.handler = async function() {
    return x + 10
}
```

此代码将导致引用错误。Lambda 将捕获此错误并生成一个包含错误消息、类型和堆栈跟踪字段的 JSON 文档。

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

在您从命令行调用函数时，AWS CLI 将响应拆分为两个文档。为指示出现函数错误，在终端中显示的响应包含 `FunctionError` 字段。函数返回的响应或错误写入到输出文件。

```
$ aws lambda invoke --function-name my-function out.json
{
    "StatusCode": 200,
    "FunctionError": "Unhandled",
    "ExecutedVersion": "$LATEST"
}
```

查看输出文件以查看错误文档。

```
$ cat out.json
```

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

Note

来自 Lambda 的响应中的 200 (成功) 状态代码指示您发送到 Lambda 的请求没有出错。有关导致错误状态代码的问题，请参阅[Errors \(p. 484\)](#)。

Lambda 还会在函数日志中记录错误对象，最多 256 KB。要在从命令行调用函数时查看日志，请使用 --log-type 选项并解码响应中的 base64 字符串。

```
$ aws lambda invoke --function-name my-function out.json --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 8bbfb91-a3ff-4502-b1b7-cb8f6658de64 Version: $LATEST
2019-06-05T22:11:27.082Z      8bbfb91-a3ff-4502-b1b7-cb8f6658de64    ERROR    Invoke
Error   {"errorType": "ReferenceError", "errorMessage": "x is not defined", "stack": [
[ "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)"]
]}
END RequestId: 8bbfb91-a3ff-4502-b1b7-cb8f6658de64
REPORT RequestId: 8bbfb91-a3ff-4502-b1b7-cb8f6658de64 Duration: 76.85 ms      Billed
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 74 MB
```

有关日志的更多信息，请参阅[Node.js 中的 AWS Lambda 函数日志记录 \(p. 271\)](#)。

AWS Lambda 可能会重试失败的 Lambda 函数，具体视事件源而定。例如，如果 Kinesis 为事件源，则 AWS Lambda 会重试失败的调用，直到 Lambda 函数成功或流中的记录过期。有关重试的更多信息，请参阅[AWS Lambda 中的错误处理和自动重试 \(p. 98\)](#)。

在 AWS Lambda 中检测 Node.js 代码

在 Node.js 中，您可以让 Lambda 向 X-Ray 发送子分段，显示您的函数对其他 AWS 服务进行的下游调用的相关信息。要执行此操作，首先需要在部署程序包中包含[适用于 Node.js 的 AWS X-Ray 开发工具包](#)。此外，请按如下方式封装您的 AWS 开发工具包 require 语句。

```
const AWSXRay = require('aws-xray-sdk-core')
const AWS = AWSXRay.captureAWS(require('aws-sdk'))
```

然后，使用前例中定义的 AWS 变量初始化 X-Ray 需要跟踪的所有服务客户端，例如：

```
const s3 = new AWS.S3()
```

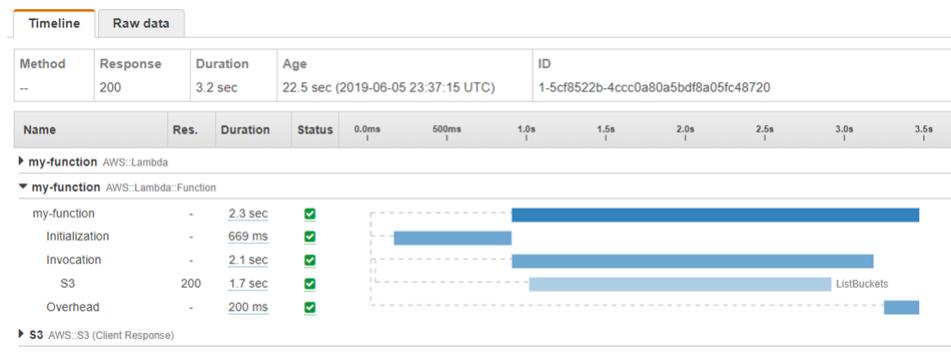
完成这些步骤后，您的函数使用 s3Client 进行的任何调用都会生成代表该调用的 X-Ray 子分段。您可以运行如下 Node.js 函数示例，了解跟踪在 X-Ray 中的样子。

Example index.js

```
const AWSXRay = require('aws-xray-sdk-core')
const AWS = AWSXRay.captureAWS(require('aws-sdk'))
const s3 = new AWS.S3()

exports.handler = async function(event) {
    return s3.listBuckets().promise()
}
```

以下示例显示一个包含 2 个分段的跟踪，这两个分段都命名为 my-function。



第一个分段表示由 Lambda 服务处理的调用请求。第二个分段记录由函数完成的工作。函数分段有 3 个子分段。

- 初始化 – 表示加载函数和运行[初始化代码 \(p. 17\)](#)所花费的时间。此子分段仅对于由函数的每个实例处理的第一个事件才显示。
- 调用 – 表示处理程序代码完成的工作。通过检测您的代码，您可以使用额外的子分段来扩展此子分段。
- 开销 – 表示 Lambda 运行时为准备处理下一个事件而完成的工作。

使用 Python 构建 Lambda 函数

您可以在 AWS Lambda 中运行 Python code。Lambda 为执行代码来处理事件的 Python 提供[运行时 \(p. 108\)](#)。您的代码在包含适用于 Python 的 开发工具包 (Boto3)的环境中运行，其中包含来自您管理的 AWS Identity and Access Management (IAM) 角色的凭证。

Lambda 支持以下 Python 运行时。

Python 运行时

名称	标识符	适用于 Python 的 AWS 开发工具包	操作系统
Python 3.8	<code>python3.8</code>	<code>boto3-1.12.22</code> <code>botocore-1.15.22</code>	Amazon Linux 2
Python 3.7	<code>python3.7</code>	<code>boto3-1.12.22</code> <code>botocore-1.15.22</code>	Amazon Linux
Python 3.6	<code>python3.6</code>	<code>boto3-1.12.22</code> <code>botocore-1.15.22</code>	Amazon Linux
Python 2.7	<code>python2.7</code>	<code>boto3-1.12.22</code> <code>botocore-1.15.22</code>	Amazon Linux

Lambda 函数使用[执行角色 \(p. 30\)](#)来获取将日志写入 Amazon CloudWatch Logs 以及访问其他服务和资源的权限。如果您还没有函数开发的执行角色，请创建一个。

创建执行角色

1. 打开 IAM 控制台中的[“角色”页面](#)。
2. 选择 Create role (创建角色)。
3. 创建具有以下属性的角色。
 - 可信任的实体 – Lambda。
 - 权限 – `AWSLambdaBasicExecutionRole`。
 - 角色名称 (角色名称) – `lambda-role`。

`AWSLambdaBasicExecutionRole` 策略具有函数将日志写入 CloudWatch Logs 所需的权限。

您可以稍后向此角色添加权限，或将其与特定于单一函数的其他角色交换。

创建 Python 函数

1. 打开 [Lambda 控制台](#)。
2. 选择创建函数。
3. 配置以下设置：
 - 名称 – `my-function`。

- 运行时 – Python 3.8。
 - 角色 – 选择现有角色。
 - 现有角色 – **lambda-role**。
4. 选择创建函数。
 5. 要配置测试事件，请选择测试。
 6. 对于事件名称，输入 **test**。
 7. 选择创建。
 8. 要执行函数，请选择测试。

控制台将创建一个带有单个名为 `lambda_function` 的源文件的 Lambda 函数。您可以在内置[代码编辑器 \(p. 5\)](#)中编辑此文件并添加更多文件。要保存您的更改，请选择 Save。然后，要运行代码，请选择 Test (测试)。

Note

Lambda 控制台使用 AWS Cloud9 在浏览器中提供集成开发环境。您还可以使用 AWS Cloud9 在您自己的环境中开发 Lambda 函数。有关更多信息，请参阅《AWS Cloud9 用户指南》中的[使用 AWS Lambda 函数](#)。

`lambda_function` 文件导出一个名为 `lambda_handler` 的函数，此函数接受事件对象和上下文对象。这是在调用函数时 Lambda 调用的[处理程序函数 \(p. 278\)](#)。Python 函数运行时从 Lambda 获取调用事件并将它们传递到处理程序。在函数配置中，处理程序值为 `lambda_function.lambda_handler`。

每次保存函数代码时，Lambda 控制台都会创建一个部署程序包，它是一个包含函数代码的 ZIP 存档。随着函数开发的进行，您需要将函数代码存储在源代码控制中、添加库和实现部署自动化。首先，通过命令行[创建部署程序包 \(p. 279\)](#)并更新代码。

除了调用事件之外，函数运行时还将上下文对象传递给处理程序。[上下文对象 \(p. 283\)](#)包含有关调用、函数和执行环境的其他信息。环境变量中提供了更多信息。

您的 Lambda 函数附带了 CloudWatch Logs 日志组。函数运行时将发送有关对 CloudWatch Logs 的每个调用的详细信息。它会中继调用期间[函数输出的任何日志 \(p. 284\)](#)。如果您的函数[返回错误 \(p. 287\)](#)，则 Lambda 将为错误设置格式，并将其返回给调用方。

主题

- [Python 中的 AWS Lambda 函数处理程序 \(p. 278\)](#)
- [Python 中的 AWS Lambda 部署程序包 \(p. 279\)](#)
- [Python 中的 AWS Lambda 上下文对象 \(p. 283\)](#)
- [Python 中的 AWS Lambda 函数日志记录 \(p. 284\)](#)
- [Python 中的 AWS Lambda 函数错误 \(p. 287\)](#)
- [在 AWS Lambda 中检测 Python 代码 \(p. 288\)](#)

Python 中的 AWS Lambda 函数处理程序

在创建 Lambda 函数时，需要指定一个处理程序（此处理程序是代码中的函数），AWS Lambda 可在服务执行代码时调用它。在 Python 中创建处理程序函数时，使用以下一般语法结构。

```
def handler_name(event, context):  
    ...
```

```
return some_value
```

在该语法中，需要注意以下方面：

- `event` – AWS Lambda 使用此参数将事件数据传递到处理程序。此参数通常是 Python `dict` 类型。它也可以是 `list`、`str`、`int`、`float` 或 `NoneType` 类型。

在调用您的函数时，您可以确定事件的内容和结构。当 AWS 服务调用您的函数时，事件结构因服务而异。有关详细信息，请参阅[将 AWS Lambda 与其他服务结合使用 \(p. 138\)](#)。

- `context` – AWS Lambda 使用此参数向您的处理程序提供运行时信息。有关详细信息，请参阅[Python 中的 AWS Lambda 上下文对象 \(p. 283\)](#)。
- (可选) 处理程序可返回值。返回的值所发生的状况取决于调用 Lambda 函数时使用的调用类型：
 - 如果您使用 `RequestResponse` 调用类型（同步执行），AWS Lambda 会将 Python 函数调用的结果返回到调用 Lambda 函数的客户端（在对调用请求的 HTTP 响应中，序列化为 JSON）。例如，AWS Lambda 控制台使用 `RequestResponse` 调用类型，因此当您使用控制台调用函数时，控制台将显示返回的值。
 - 如果处理程序返回 `json.dumps` 无法序列化的对象，则运行时返回错误。
 - 如果处理程序返回 `None`（就像不具有 `return` 语句的 Python 函数隐式执行的那样），则运行时返回 `null`。
 - 如果您使用 `Event` 调用类型（异步执行），则丢弃该值。

例如，考虑以下 Python 示例代码。

```
def my_handler(event, context):
    message = 'Hello {} {}!'.format(event['first_name'],
                                    event['last_name'])
    return {
        'message' : message
    }
```

此示例具有一个名为 `my_handler` 的函数。此函数从它接收为输入的事件返回包含数据的消息。

Python 中的 AWS Lambda 部署程序包

部署程序包是包含函数代码和依赖项的 ZIP 存档。如果您使用 Lambda API 管理函数，或者需要包含 AWS 开发工具包以外的库和依赖项，则需要创建部署程序包。您可以将程序包直接上传到 Lambda，也可以使用 Amazon S3 存储桶、然后再将其上传到 Lambda。如果部署包大于 50 MB，则必须使用 Amazon S3。

如果您使用 Lambda [控制台编辑器 \(p. 5\)](#) 编写您的函数，则控制台会管理部署程序包。如果您不需要添加任何库，则可以使用此方法。您也可以使用此方法更新在部署程序包中已经存在库的函数，前提是总大小不超过 3 MB。

Note

您可以使用 AWS SAM CLI `build` 命令为 Python 函数代码和依赖项创建部署包。AWS SAM CLI 还提供了在与 Lambda 执行环境兼容的 Docker 镜像内构建部署包的选项。有关说明，请参阅[《AWS SAM 开发人员指南》中的构建具有依赖项的应用程序](#)。

小节目录

- [先决条件 \(p. 280\)](#)
- [更新没有依赖项的函数 \(p. 280\)](#)
- [更新具有额外依赖项的函数 \(p. 280\)](#)
- [使用虚拟环境 \(p. 281\)](#)

先决条件

以下说明假定您已有一个函数。如果您尚未创建函数，请参阅[使用 Python 构建 Lambda 函数 \(p. 277\)](#)。

为了遵循本指南中的步骤，您需要命令行终端或外壳，以便运行命令。命令显示在列表中，以提示符 (\$) 和当前目录名称（如果有）开头：

```
~/lambda-project$ this is a command  
this is output
```

对于长命令，使用转义字符 (\) 将命令拆分到多行中。

在 Linux 和 macOS 中，可使用您首选的外壳程序和程序包管理器。在 Windows 10 中，您可以[安装 Windows Subsystem for Linux](#)，获取 Ubuntu 和 Bash 与 Windows 集成的版本。

更新没有依赖项的函数

要使用 Lambda API 创建或更新函数，请创建包含函数代码的存档，并使用 AWS CLI 上传。

更新没有依赖项的 Python 函数

1. 创建 ZIP 存档。

```
~/my-function$ zip function.zip lambda_function.py  
adding: lambda_function.py (deflated 17%)
```

2. 使用 update-function-code 命令上传程序包。

```
~/my-function$ aws lambda update-function-code --function-name my-function --zip-file  
fileb://function.zip  
{  
    "FunctionName": "my-function",  
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",  
    "Runtime": "python3.8",  
    "Role": "arn:aws:iam::123456789012:role/lambda-role",  
    "Handler": "lambda_function.lambda_handler",  
    "CodeSize": 815,  
    "CodeSha256": "GcZ05oeHoJi61VpQj7vCLPs8DwCXmX5sE/fE2Ihsizc=",  
    "Version": "$LATEST",  
    "RevisionId": "d1e983e3-ca8e-434b-8dc1-7add83d72ebd",  
    ...  
}
```

更新具有额外依赖项的函数

如果您的函数依赖于适用于 Python 的开发工具包 (Boto3) 之外的库，请使用 [pip](#) 将它们安装到本地目录中，并将它们包含在部署包程序中。

Note

对于具有使用 C 或 C ++ 编写的扩展模块的库，请在 Amazon Linux 环境中构建您的部署程序包。您可以使用[SAM CLI 构建命令](#)（使用 Docker），也可以在 Amazon EC2 或 AWS CodeBuild 上构建您的部署程序包。

以下示例演示如何创建包含名为“Pillow”的常用图形库的部署包。

更新有依赖项的 Python 函数

1. 使用 pip 的 --target 选项在新的项目本地 package 目录中安装库。

```
~/my-function$ pip install --target ./package Pillow
Collecting Pillow
  Using cached https://files.pythonhosted.org/
  packages/62/8c/230204b8e968f6db00c765624f51cf1ecb6aea57b25ba00b240ee3fb0bd/
  Pillow-5.3.0-cp37-cp37m-manylinux1_x86_64.whl
  Installing collected packages: Pillow
  Successfully installed Pillow-5.3.0
```

Note

为了使 --target 能够在[基于 Debian 的系统](#)（例如 Ubuntu）上正常工作，可能还需要传递 --system 标志来避免 distutils 错误。

2. 创建包含依赖项的 ZIP 存档。

```
~/my-function$ cd package
~/my-function/package$ zip -r9 ${OLDPWD}/function.zip .
  adding: PIL/ (stored 0%)
  adding: PIL/.libs/ (stored 0%)
  adding: PIL/.libs/libfreetype-7ce95de6.so.6.16.1 (deflated 65%)
  adding: PIL/.libs/libjpeg-3fe7dfc0.so.9.3.0 (deflated 72%)
  adding: PIL/.libs/liblcms2-a6801db4.so.2.0.8 (deflated 67%)
  ...
```

3. 将您的函数代码添加到存档中。

```
~/my-function/package$ cd ${OLDPWD}
~/my-function$ zip -g function.zip lambda_function.py
  adding: lambda_function.py (deflated 56%)
```

4. 更新函数代码。

```
~/my-function$ aws lambda update-function-code --function-name my-function --zip-file
  file:///function.zip
{
    "FunctionName": "my-function",
    "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
    "Runtime": "python3.8",
    "Role": "arn:aws:iam::123456789012:role/lambda-role",
    "Handler": "lambda_function.lambda_handler",
    "CodeSize": 2269409,
    "CodeSha256": "GcZ05oeHoJi61VpQj7vCLPs8DwCXmX5sE/fE2IHsizc=",
    "Version": "$LATEST",
    "RevisionId": "a9c05ffd-8ad6-4d22-b6cd-d34a00c1702c",
    ...
}
```

使用虚拟环境

有些情况下，您可能需要使用[虚拟环境](#)来安装函数的依赖项。如果您的函数或其依赖项依赖于本机库，或者您使用 Homebrew 安装 Python，则会发生这种情况。

使用虚拟环境更新 Python 函数

1. 创建虚拟环境。

```
~/my-function$ virtualenv v-env
Using base prefix '/.local/python-3.7.0'
New python executable in v-env/bin/python3.8
Also creating executable in v-env/bin/python
Installing setuptools, pip, wheel...
done.
```

Note

对于 Python 3.3 及更高版本，可以使用内置的 [venv 模块](#) 创建虚拟环境，而不是安装 `virtualenv`。

```
~/my-function$ python3 -m venv v-env
```

2. 激活环境。

```
~/my-function$ source v-env/bin/activate
(v-env) ~/my-function$
```

3. 使用 pip 安装库。

```
(v-env) ~/my-function$ pip install Pillow
Collecting Pillow
  Using cached https://files.pythonhosted.org/
packages/62/8c/230204b8e968f6db00c765624f51cf1ecb6aea57b25ba00b240ee3fb0bd/
Pillow-5.3.0-cp37-cp37m-manylinux1_x86_64.whl
Installing collected packages: Pillow
Successfully installed Pillow-5.3.0
```

4. 停用虚拟环境。

```
(v-env) ~/my-function$ deactivate
```

5. 使用库内容创建一个 ZIP 存档。

```
~/my-function$ cd v-env/lib/python3.8/site-packages
~/my-function/v-env/lib/python3.8/site-packages$ zip -r9 ${OLDPWD}/function.zip .
  adding: easy_install.py (deflated 17%)
  adding: PIL/ (stored 0%)
  adding: PIL/.libs/ (stored 0%)
  adding: PIL/.libs/libfreetype-7ce95de6.so.6.16.1 (deflated 65%)
  adding: PIL/.libs/libjpeg-3fe7dfc0.so.9.3.0 (deflated 72%)
...
```

根据库的情况，依赖项可能出现在 `site-packages` 或 `dist-packages` 中，而虚拟环境中的第一个文件夹可能是 `lib` 或 `lib64`。可以使用 `pip show` 命令来定位特定包。

6. 将您的函数代码添加到存档中。

```
~/my-function/v-env/lib/python3.8/site-packages$ cd ${OLDPWD}
~/my-function$ zip -g function.zip lambda_function.py
  adding: lambda_function.py (deflated 56%)
```

7. 更新函数代码。

```
~/my-function$ aws lambda update-function-code --function-name my-function --zip-file
fileb://function.zip
{
```

```
"FunctionName": "my-function",
"FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
"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",
...
}
```

Python 中的 AWS Lambda 上下文对象

当 Lambda 运行您的函数时，它会将上下文对象传递到[处理程序 \(p. 278\)](#)。此对象提供的方法和属性包含有关调用、函数和执行环境的信息。

上下文方法

- `get_remaining_time_in_millis` – 返回在执行超时以前剩余的毫秒数。

上下文属性

- `function_name` – Lambda 函数的名称。
- `function_version` – 函数的版本 ([p. 63](#))。
- `invoked_function_arn` – 用于调用函数的 Amazon 资源名称 (ARN)。指示调用方是否已指定版本或别名。
- `memory_limit_in_mb` – 为函数分配的内存量。
- `aws_request_id` – 调用请求的标识符。
- `log_group_name` – 函数的日志组。
- `log_stream_name` – 函数实例的日志流。
- `identity` – (移动应用程序) 有关授权请求的 Amazon Cognito 身份的信息。
 - `cognito_identity_id` – 已经过身份验证的 Amazon Cognito 身份。
 - `cognito_identity_pool_id` – 授权调用的 Amazon Cognito 身份池。
- `client_context` – (移动应用程序) 由客户端应用程序向 Lambda 提供的客户端上下文。
 - `client.installation_id`
 - `client.app_title`
 - `client.app_version_name`
 - `client.app_version_code`
 - `client.app_package_name`
 - `custom` – 由移动客户端应用程序设置的自定义值的 dict。
 - `env` – 由 AWS 开发工具包提供的环境信息的 dict。

以下示例显示记录上下文信息的处理程序函数。

Example handler.py

```
import time
def get_my_log_stream(event, context):
```

```
print("Log stream name:", context.log_stream_name)
print("Log group name:", context.log_group_name)
print("Request ID:", context.aws_request_id)
print("Mem. limits(MB):", context.memory_limit_in_mb)
# Code will execute quickly, so we add a 1 second intentional delay so you can see that
in time remaining value.
time.sleep(1)
print("Time remaining (MS):", context.get_remaining_time_in_millis())
```

除了上面列出的选项，您还可以使用适用于 [在 AWS Lambda 中检测 Python 代码 \(p. 288\)](#) 的 AWS X-Ray 开发工具包来识别关键代码路径、跟踪其性能并收集数据以用于分析。

Python 中的 AWS Lambda 函数日志记录

您的 Lambda 函数带有一个 CloudWatch Logs 日志组，其中包含您的函数的每个实例的日志流。运行时会将每个调用的详细信息发送到该日志流，然后中继日志和来自您的函数代码的其他输出。

要从函数代码输出日志，您可以使用 `print` 方法或使用写入到 `stdout` 或 `stderr` 的任何日志记录库。以下示例记录环境变量和事件对象的值。

Example `lambda_function.py`

```
import json
import os

def lambda_handler(event, context):
    print('## ENVIRONMENT VARIABLES')
    print(os.environ)
    print('## EVENT')
    print(event)
```

Example 日志格式

```
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: 100 ms Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 130.49 ms
XRAY TraceId: 1-5e34a614-10bdxmplf1fb44f07bc535a1 SegmentId: 07f5xmpl2d1f6f85 Sampled:
true
```

Python 运行时记录每次调用的 START、END 和 REPORT 行。报告行提供了以下详细信息。

报告日志

- RequestId – 调用的唯一请求 ID。
- Duration (持续时间) – 函数的处理程序方法处理事件所花费的时间。
- Billed Duration (计费持续时间) – 针对调用计费的时间量。
- Memory Size (内存大小) – 分配给函数的内存量。
- Max Memory Used (最大内存使用量) – 函数使用的内存量。

- Init Duration (初始持续时间) – 对于提供的第一个请求，为运行时在处理程序方法外部加载函数和运行代码所花费的时间。
- XRAY Traceld – 对于跟踪的请求，为 [AWS X-Ray 跟踪编码 \(p. 371\)](#)。
- SegmentId – 对于跟踪的请求，为 X-Ray 分段 ID。
- Sampled (采样) – 对于跟踪的请求，为采样结果。

您可以在 Lambda 控制台中、在 CloudWatch Logs 控制台中或从命令行查看日志。

小节目录

- [在 AWS 管理控制台中查看日志 \(p. 285\)](#)
- [使用 AWS CLI \(p. 285\)](#)
- [删除日志 \(p. 287\)](#)
- [日志记录库 \(p. 287\)](#)

在 AWS 管理控制台中查看日志

当您在函数配置页上测试函数时，Lambda 控制台会显示日志输出。要查看所有调用的日志，请使用 CloudWatch Logs 控制台。

查看 Lambda 函数的日志

1. 打开 [CloudWatch 控制台的日志页面](#)。
2. 选择您的函数 (`/aws/lambda/function-name`) 的日志组。
3. 选择列表中的第一个流。

每个日志流对应一个 [函数实例 \(p. 109\)](#)。当您更新函数以及创建更多实例以处理多个并发调用时，会显示新的流。要找到特定调用的日志，您可以使用 X-Ray 分析您的函数并在跟踪中记录有关请求和日志流的详细信息。如需将日志和跟踪与 X-Ray 相关联的示例应用程序，请参阅[AWS Lambda 错误处理器示例应用程序 \(p. 128\)](#)。

使用 AWS CLI

要从命令行获取调用的日志，请使用 `--log-type` 选项。响应包含一个 `LogResult` 字段，该字段包含来自调用的多达 4 KB 的 base64 编码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc21vb...",
    "ExecutedVersion": "$LATEST"
}
```

您可以使用 `base64` 实用程序来解码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
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: 100 ms           Memory Size: 128 MB      Max Memory Used: 73 MB
```

base64 实用工具在 Linux、macOS 和 [Ubuntu on Windows](#) 上可用。对于 macOS，命令为 `base64 -D`。

要从命令行获取完整的日志事件，您可以在函数输出中包含日志流名称，如上例中所示。以下示例脚本调用名为 `my-function` 的函数并下载最后 5 个日志事件。

Example get-logs.sh 脚本

此示例要求 `my-function` 返回日志流 ID。

```
#!/bin/bash
aws lambda invoke --function-name my-function --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
```

此脚本使用 `sed` 从输出文件中删除引号，并休眠 15 秒以等待日志可用。输出包括来自 Lambda 的响应，以及来自 `get-log-events` 命令的输出。

```
$ ./get-logs.sh
{
  "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 ...",
      "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: 100 ms \tMemory Size: 128 MB\tMax Memory Used: 75 MB\t\n",
      "ingestionTime": 1559763018353
    }
  ],
  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
  "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

删除日志

在删除函数时，不会自动删除日志组。为避免存储无限的日志，请删除日志组或者[配置保留期](#)，在该期限后会自动删除日志。

日志记录库

如需更详细的日志，请使用[日志记录库](#)。

```
import json
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)
```

logger 的输出包括日志级别、时间戳和请求 ID。

```
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: 100 ms Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 113.51 ms
XRAY TraceId: 1-5e34a66a-474xmpl7c2534a87870b4370 SegmentId: 073cxmpl3e442861 Sampled:
true
```

Python 中的 AWS Lambda 函数错误

当您的代码引发错误时，Lambda 将生成错误的 JSON 表示形式。此错误文档会出现在调用日志中，对于同步调用，它出现在输出中。

Example `lambda_function.py` 文件 – 异常

```
def lambda_handler(event, context):
    return x + 10
```

此代码将导致名称错误。Lambda 将捕获此错误并生成一个包含错误消息、类型和堆栈跟踪字段的 JSON 文档。

```
{  
    "errorMessage": "name 'x' is not defined",  
    "errorType": "NameError",  
    "stackTrace": [  
        "  File \"/var/task/error_function.py\", line 2, in lambda_handler\n            return x +  
10\\n"  
    ]  
}
```

在您从命令行调用函数时，AWS CLI 将响应拆分为两个文档。为指示出现函数错误，在终端中显示的响应包含 `FunctionError` 字段。函数返回的响应或错误写入到输出文件。

```
$ aws lambda invoke --function-name my-function out.json  
{  
    "StatusCode": 200,  
    "FunctionError": "Unhandled",  
    "ExecutedVersion": "$LATEST"  
}
```

查看输出文件以查看错误文档。

```
$ cat out.json  
{"errorMessage": "name 'x' is not defined", "errorType": "NameError", "stackTrace": ["  
File \"/var/task/error_function.py\", line 2, in lambda_handler\n            return x + 10\\n"]}
```

Note

来自 Lambda 的响应中的 200 (成功) 状态代码指示您发送到 Lambda 的请求没有出错。有关导致错误状态代码的问题，请参阅 [Errors \(p. 484\)](#)。

Lambda 还会在函数日志中记录错误对象，最多 256 KB。要在从命令行调用函数时查看日志，请使用 `--log-type` 选项并解码响应中的 base64 字符串。

```
$ aws lambda invoke --function-name my-function out.json --log-type Tail \  
--query 'LogResult' --output text | base64 -d  
START RequestId: fc4f8810-88ff-4800-974c-12cec018a4b9 Version: $LATEST  
    return x + 10/lambda_function.py", line 2, in lambda_handler  
END RequestId: fc4f8810-88ff-4800-974c-12cec018a4b9  
REPORT RequestId: fc4f8810-88ff-4800-974c-12cec018a4b9 Duration: 12.33 ms Billed Duration:  
100 ms Memory Size: 128 MB Max Memory Used: 56 MB
```

有关日志的更多信息，请参阅 [Python 中的 AWS Lambda 函数日志记录 \(p. 284\)](#)。

在 AWS Lambda 中检测 Python 代码

在 Python 中，您可以让 Lambda 向 X-Ray 发送子段，显示您的函数对其他 AWS 服务进行的下游调用的相关信息。要执行此操作，首先需要在部署程序包中包含[适用于 Python 的 AWS X-Ray 开发工具包](#)。此外，您还可以修补 `boto3` (或 `botocore`，如果您使用的是会话)，以便您为了访问其他 AWS 服务而创建的任何客户端都将自动被 X-Ray 跟踪。

```
import boto3  
from aws_xray_sdk.core import xray_recorder  
from aws_xray_sdk.core import patch
```

```
patch(['boto3'])
```

您修补用于创建客户端的模块后，便可以使用它来创建您的被跟踪客户端，在下面的情况下 Amazon S3：

```
s3_client = boto3.client('s3')
```

适用于 Python 的 X-Ray 开发工具包为调用创建子分段，并记录请求和响应中的信息。您可以使用 `aws_xray_sdk_sdk.core.xray_recorder` 通过装饰 Lambda 函数自动创建子分段，或通过在函数内调用 `xray_recorder.begin_subsegment()` 和 `xray_recorder.end_subsegment()` 手动创建子分段，如以下 Lambda 函数中所示。

```
import boto3
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch

patch(['boto3'])

s3_client = boto3.client('s3')

def lambda_handler(event, context):
    bucket_name = event['bucket_name']
    bucket_key = event['bucket_key']
    body = event['body']

    put_object_into_s3(bucket_name, bucket_key, body)
    get_object_from_s3(bucket_name, bucket_key)

# Define subsegments manually
def put_object_into_s3(bucket_name, bucket_key, body):
    try:
        xray_recorder.begin_subsegment('put_object')
        response = s3_client.put_object(Bucket=bucket_name, Key=bucket_key, Body=body)
        status_code = response['ResponseMetadata']['HTTPStatusCode']
        xray_recorder.current_subsegment().put_annotation('put_response', status_code)
    finally:
        xray_recorder.end_subsegment()

# Use decorators to automatically set the subsegments
@xray_recorder.capture('get_object')
def get_object_from_s3(bucket_name, bucket_key):
    response = s3_client.get_object(Bucket=bucket_name, Key=bucket_key)
    status_code = response['ResponseMetadata']['HTTPStatusCode']
    xray_recorder.current_subsegment().put_annotation('get_response', status_code)
```

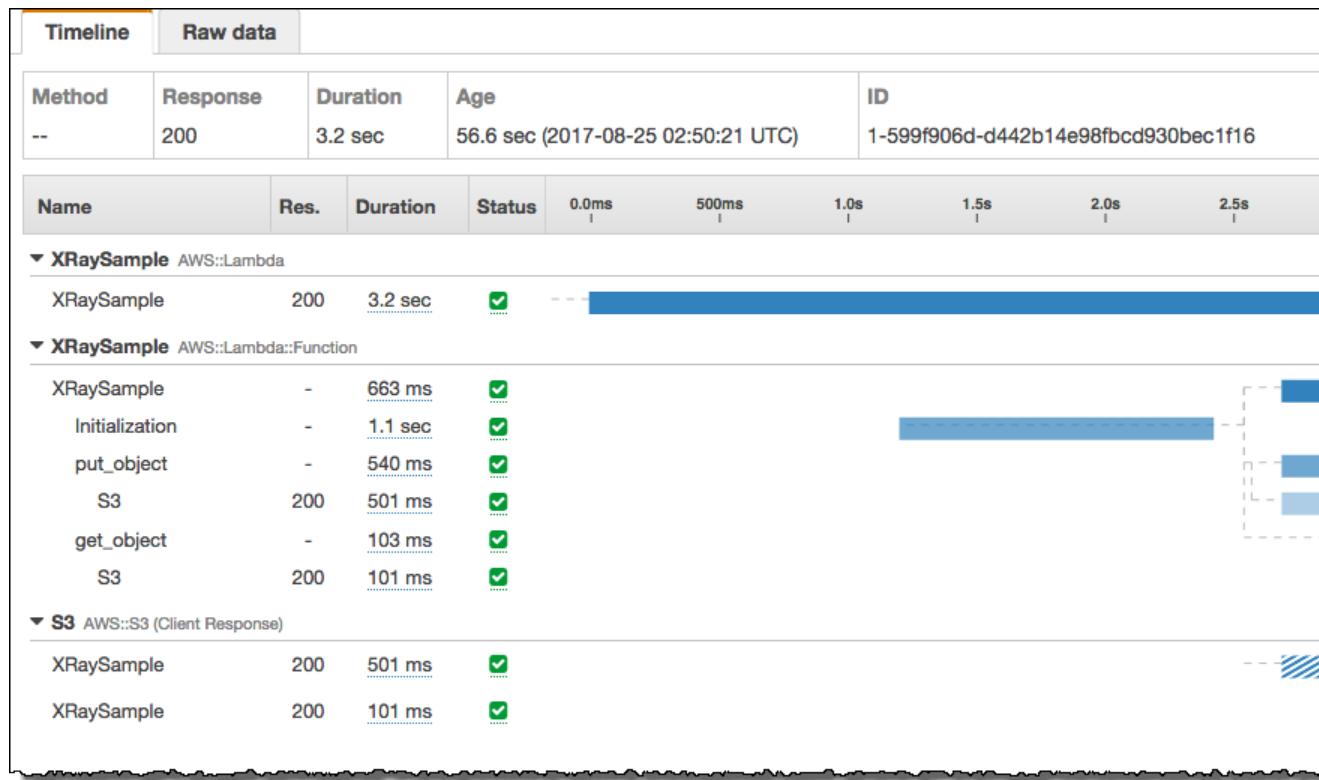
Note

适用于 Python 的 X-Ray 开发工具包能让您修补以下模块：

- botocore
- boto3
- 请求
- sqlite3
- mysql

您可以使用 `patch_all()` 来同时将它们全部修补。

以下是上述代码发送的跟踪的样子（同步调用）：



使用 Ruby 构建 Lambda 函数

您可以在 AWS Lambda 中运行 Ruby code。Lambda 为执行代码来处理事件的 Ruby 提供[运行时 \(p. 108\)](#)。您的代码在包含适用于 Ruby 的 AWS 开发工具包的环境中运行，其中包含来自您管理的 AWS Identity and Access Management (IAM) 角色的凭证。

Lambda 支持以下 Ruby 运行时。

Ruby 运行时

名称	标识符	适用于 Ruby 的 AWS 开发工具包	操作系统
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

Lambda 函数使用[执行角色 \(p. 30\)](#)来获取将日志写入 Amazon CloudWatch Logs 以及访问其他服务和资源的权限。如果您还没有函数开发的执行角色，请创建一个。

创建执行角色

1. 打开 IAM 控制台中的[“角色”页面](#)。
2. 选择 Create role (创建角色)。
3. 创建具有以下属性的角色。
 - 可信任的实体 – Lambda。
 - 权限 – `AWSLambdaBasicExecutionRole`。
 - 角色名称 (角色名称) – `lambda-role`。

`AWSLambdaBasicExecutionRole` 策略具有函数将日志写入 CloudWatch Logs 所需的权限。

您可以稍后向此角色添加权限，或将其与特定于单一函数的其他角色交换。

创建 Ruby 函数

1. 打开 [Lambda 控制台](#)。
2. 选择创建函数。
3. 配置以下设置：
 - 名称 – `my-function`。
 - 运行时 – Ruby 2.7。
 - 角色 – 选择现有角色。
 - 现有角色 – `lambda-role`。
4. 选择创建函数。
5. 要配置测试事件，请选择测试。
6. 对于事件名称，输入 `test`。

7. 选择创建。
8. 要执行函数，请选择测试。

控制台将创建一个带有单个名为 `lambda_function.rb` 的源文件的 Lambda 函数。您可以在内置[代码编辑器 \(p. 5\)](#)中编辑此文件并添加更多文件。要保存您的更改，请选择 Save。然后，要运行代码，请选择 Test (测试)。

Note

Lambda 控制台使用 AWS Cloud9 在浏览器中提供集成开发环境。您还可以使用 AWS Cloud9 在您自己的环境中开发 Lambda 函数。有关更多信息，请参阅《AWS Cloud9 用户指南》中的[使用 AWS Lambda 函数](#)。

`lambda_function.rb` 文件导出一个名为 `lambda_handler` 的函数，此函数接受事件对象和上下文对象。这是在调用函数时 Lambda 调用的[处理程序函数 \(p. 292\)](#)。Ruby 函数运行时从 Lambda 获取调用事件并将它们传递到处理程序。在函数配置中，处理程序值为 `lambda_function.lambda_handler`。

每次保存函数代码时，Lambda 控制台都会创建一个部署程序包，它是一个包含函数代码的 ZIP 存档。随着函数开发的进行，您需要将函数代码存储在源代码控制中、添加库和实现部署自动化。首先，通过命令行[创建部署程序包 \(p. 293\)](#)并更新代码。

除了调用事件之外，函数运行时还将上下文对象传递给处理程序。[上下文对象 \(p. 295\)](#)包含有关调用、函数和执行环境的其他信息。环境变量中提供了更多信息。

您的 Lambda 函数附带了 CloudWatch Logs 日志组。函数运行时将发送有关对 CloudWatch Logs 的每个调用的详细信息。它会中继调用期间[函数输出的任何日志 \(p. 295\)](#)。如果您的函数[返回错误 \(p. 299\)](#)，则 Lambda 将为错误设置格式，并将其返回给调用方。

主题

- [Ruby 中的 AWS Lambda 函数处理程序 \(p. 292\)](#)
- [Ruby 中的 AWS Lambda 部署程序包 \(p. 293\)](#)
- [Ruby 中的 AWS Lambda 上下文对象 \(p. 295\)](#)
- [Ruby 中的 AWS Lambda 函数日志记录 \(p. 295\)](#)
- [Ruby 中的 AWS Lambda 函数错误 \(p. 299\)](#)

Ruby 中的 AWS Lambda 函数处理程序

您的 Lambda 函数的处理程序是 Lambda 在该函数被调用时调用的方法。在以下示例中，文件 `function.rb` 将定义一个名为 `handler` 的处理程序方法。该处理程序函数将选取两个对象作为输入并返回一个 JSON 文档。

Example function.rb

```
require 'json'

def handler(event:, context:)
  { event: JSON.generate(event), context: JSON.generate(context.inspect) }
end
```

在您的函数配置中，`handler` 设置指示 Lambda 在何处查找处理程序。对于上述示例，此设置的正确值为 `function.handler`。它包含两个由点分隔的名称：文件的名和处理程序方法的名称。

您还可以在类中定义处理程序方法。以下示例在名为 `LambdaFunctions` 的模块中的名为 `Handler` 的类上定义了一个名为 `process` 的处理程序。

Example source.rb

```
module LambdaFunctions
  class Handler
    def self.process(event:, context:)
      "Hello!"
    end
  end
end
```

在本例中，处理程序设置为 `source.LambdaFunctions::Handler.process`。

该处理程序接受的两个对象是调用事件和上下文。事件是一个 Ruby 对象，包含由调用方提供的负载。如果该负载为 JSON 文档，则事件对象为 Ruby 哈希。否则，它是一个字符串。[上下文对象 \(p. 295\)](#)具有一些方法和属性，它们提供了有关调用、函数和执行环境的信息。

函数处理程序在您的 Lambda 函数每次被调用时执行。处理程序外部的静态代码对该函数的每个实例执行一次。如果您的处理程序使用开发工具包客户端和数据库连接之类的资源，您可以在处理程序方法外部创建这些资源以对多个调用重复使用它们。

您的函数的每个实例都可以处理多个调用事件，但一次只能处理一个事件。在给定的任何时间，处理事件的实例数都是您的函数的并发。有关 Lambda 执行上下文的更多信息，请参阅 [AWS Lambda 执行上下文 \(p. 109\)](#)。

Ruby 中的 AWS Lambda 部署程序包

部署程序包是包含函数代码和依赖项的 ZIP 存档。如果您使用 Lambda API 管理函数，或者需要包含 AWS 开发工具包以外的库和依赖项，则需要创建部署程序包。您可以将程序包直接上传到 Lambda，也可以使用 Amazon S3 存储桶、然后再将其上传到 Lambda。如果部署包大于 50 MB，则必须使用 Amazon S3。

如果您使用 Lambda [控制台编辑器 \(p. 5\)](#)编写您的函数，则控制台会管理部署程序包。如果您不需要添加任何库，则可以使用此方法。您也可以使用此方法更新在部署程序包中已经存在库的函数，前提是总大小不超过 3 MB。

Note

为了减小部署程序包的大小，请将函数的依赖项打包到层中。层可让您独立管理依赖项，可以供多个函数使用，并且可以与其他账户共享。有关详细信息，请参阅[AWS Lambda 层 \(p. 68\)](#)。

小目录

- [更新没有依赖项的函数 \(p. 293\)](#)
- [更新具有额外依赖项的函数 \(p. 294\)](#)

更新没有依赖项的函数

要使用 Lambda API 更新函数，请使用 [UpdateFunctionCode \(p. 553\)](#) 操作。创建包含函数代码的存档，然后使用 AWS CLI 上传该存档。

更新没有依赖项的 Ruby 函数

1. 创建 ZIP 存档。

```
~/my-function$ zip function.zip function.rb
```

2. 使用 `update-function-code` 命令上传程序包。

```
~/my-function$ aws lambda update-function-code --function-name my-function --zip-file fileb://function.zip
{
    "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",
    ...
}
```

更新具有额外依赖项的函数

如果您的函数依赖于适用于 Ruby 的 AWS 开发工具包之外的库，请使用 [Bundler](#) 将它们安装到本地目录，并将它们包含在部署程序包中。

更新具有依赖项的 Ruby 函数

1. 使用 `bundle` 命令安装供应商目录中的库。

```
~/my-function$ bundle install --path vendor/bundle
Fetching gem metadata from https://rubygems.org/.....
Resolving dependencies...
Fetching aws-eventstream 1.0.1
Installing aws-eventstream 1.0.1
...
```

`--path` 会将 Gem 安装在项目目录而不是系统位置，并将该目录设置为将来安装的默认路径。要稍后全局安装 Gem，请使用 `--system` 选项。

2. 创建 ZIP 存档。

```
package$ zip -r function.zip function.rb vendor
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. 更新函数代码。

```
~/my-function$ aws lambda update-function-code --function-name my-function --zip-file fileb://function.zip
{
    "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",  
...  
}
```

Ruby 中的 AWS Lambda 上下文对象

当 Lambda 运行您的函数时，它会将上下文对象传递到[处理程序 \(p. 292\)](#)。此对象提供的方法和属性包含有关调用、函数和执行环境的信息。

上下文方法

- `get_remaining_time_in_millis` – 返回在执行超时以前剩余的毫秒数。

上下文属性

- `function_name` – Lambda 函数的名称。
- `function_version` – 函数的版本 (p. 63)。
- `invoked_function_arn` – 用于调用函数的 Amazon 资源名称 (ARN)。指示调用方是否已指定版本或别名。
- `memory_limit_in_mb` – 为函数分配的内存量。
- `aws_request_id` – 调用请求的标识符。
- `log_group_name` – 函数的日志组。
- `log_stream_name` – 函数实例的日志流。
- `deadline_ms` – 执行超时的日期 (Unix 时间形式，以毫秒为单位)。
- `identity` – (移动应用程序) 有关授权请求的 Amazon Cognito 身份的信息。
- `client_context` – (移动应用程序) 由客户端应用程序向 Lambda 提供的客户端上下文。

Ruby 中的 AWS Lambda 函数日志记录

您的 Lambda 函数带有一个 CloudWatch Logs 日志组，其中包含您的函数的每个实例的日志流。运行时会将每个调用的详细信息发送到该日志流，然后中继日志和来自您的函数代码的其他输出。

要从函数代码输出日志，您可以使用 `puts` 语句或使用写入到 `stdout` 或 `stderr` 的任何日志记录库。以下示例记录环境变量和事件对象的值。

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
```

有关更详细的日志，请使用[记录器库](#)。

```
# 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
```

来自 logger 的输出包括时间戳、进程 ID、日志级别和请求 ID。

```
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"}
```

当您在函数配置页上测试函数时，Lambda 控制台会显示日志输出。要查看所有调用的日志，请使用 CloudWatch Logs 控制台。

查看 Lambda 函数的日志

1. 打开 [CloudWatch 控制台的日志页面](#)。
2. 选择您的函数 (`/aws/lambda/function-name`) 的日志组。
3. 选择列表中的第一个流。

每个日志流对应一个[函数实例 \(p. 109\)](#)。当您更新函数以及创建更多实例以处理多个并发调用时，会显示新的流。要找到特定调用的日志，您可以使用 X-Ray 分析您的函数并在跟踪中记录有关请求和日志流的详细信息。如需将日志和跟踪与 X-Ray 相关联的示例应用程序，请参阅[AWS Lambda 错误处理器示例应用程序 \(p. 128\)](#)。

要从命令行获取调用的日志，请使用 `--log-type` 选项。响应包含一个 `LogResult` 字段，该字段包含来自调用的多达 4 KB 的 base64 编码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
  "StatusCode": 200,
  "LogResult":
  "U1RBULQgUmVxdWVzdElkOiA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
  "ExecutedVersion": "$LATEST"
}
```

您可以使用 base64 实用程序来解码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
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: 100 ms          Memory Size: 128 MB      Max Memory Used: 73 MB
```

base64 实用工具在 Linux、macOS 和 [Ubuntu on Windows](#) 上可用。对于 macOS，命令为 `base64 -D`。

在删除函数时，不会自动删除日志组。为避免存储无限的日志，请删除日志组或者[配置保留期](#)，在该期限后会自动删除日志。

Example 日志格式

```
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:
100 ms Memory Size: 128 MB Max Memory Used: 48 MB Init Duration: 117.86 ms
XRAY TraceId: 1-5e34a246-2a04xmpl0fa44eb60ea08c5f SegmentId: 454xmpl46ca1c7d3 Sampled: true
```

Ruby 运行时记录每次调用的 START、END 和 REPORT 行。报告行提供了以下详细信息。

报告日志

- RequestId – 调用的唯一请求 ID。
- Duration (持续时间) – 函数的处理程序方法处理事件所花费的时间。
- Billed Duration (计费持续时间) – 针对调用计费的时间量。
- Memory Size (内存大小) – 分配给函数的内存量。
- Max Memory Used (最大内存使用量) – 函数使用的内存量。
- Init Duration (初始持续时间) – 对于提供的第一个请求，为运行时在处理程序方法外部加载函数和运行代码所花费的时间。
- XRAY Traceld – 对于跟踪的请求，为 [AWS X-Ray 跟踪编码 \(p. 371\)](#)。
- SegmentId – 对于跟踪的请求，为 X-Ray 分段 ID。
- Sampled (采样) – 对于跟踪的请求，为采样结果。

您可以在 Lambda 控制台中、在 CloudWatch Logs 控制台中或从命令行查看日志。

小目录

- [在 AWS 管理控制台中查看日志 \(p. 285\)](#)
- [使用 AWS CLI \(p. 285\)](#)
- [删除日志 \(p. 287\)](#)

在 AWS 管理控制台中查看日志

当您在函数配置页上测试函数时，Lambda 控制台会显示日志输出。要查看所有调用的日志，请使用 CloudWatch Logs 控制台。

查看 Lambda 函数的日志

1. 打开 [CloudWatch 控制台的日志页面](#)。
2. 选择您的函数 (`/aws/lambda/function-name`) 的日志组。
3. 选择列表中的第一个流。

每个日志流对应一个[函数实例 \(p. 109\)](#)。当您更新函数以及创建更多实例以处理多个并发调用时，会显示新的流。要找到特定调用的日志，您可以使用 X-Ray 分析您的函数并在跟踪中记录有关请求和日志流的详细信息。如需将日志和跟踪与 X-Ray 相关联的示例应用程序，请参阅[AWS Lambda 错误处理器示例应用程序 \(p. 128\)](#)。

使用 AWS CLI

要从命令行获取调用的日志，请使用 `--log-type` 选项。响应包含一个 `LogResult` 字段，该字段包含来自调用的多达 4 KB 的 base64 编码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

您可以使用 `base64` 实用程序来解码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
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: 100 ms          Memory Size: 128 MB      Max Memory Used: 73 MB
```

`base64` 实用工具在 Linux、macOS 和 [Ubuntu on Windows](#) 上可用。对于 macOS，命令为 `base64 -D`。

要从命令行获取完整的日志事件，您可以在函数输出中包含日志流名称，如上例中所示。以下示例脚本调用名为 `my-function` 的函数并下载最后 5 个日志事件。

Example get-logs.sh 脚本

此示例要求 `my-function` 返回日志流 ID。

```
#!/bin/bash
aws lambda invoke --function-name my-function --payload '{"key": "value"}' out
sed -i'' -e 's/"/\n/g' out
sleep 15
aws logs get-log-events --log-group-name /aws/lambda/my-function --log-stream-name $(cat
out) --limit 5
```

此脚本使用 `sed` 从输出文件中删除引号，并休眠 15 秒以等待日志可用。输出包括来自 Lambda 的响应，以及来自 `get-log-events` 命令的输出。

```
$ ./get-logs.sh
{
    "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: 100 ms \tMemory Size: 128 MB\tMax Memory Used: 75 MB\t\n",
        "ingestionTime": 1559763018353
    }
],
"nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
"nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}

```

删除日志

在删除函数时，不会自动删除日志组。为避免存储无限的日志，请删除日志组或者[配置保留期](#)，在该期限后会自动删除日志。

Ruby 中的 AWS Lambda 函数错误

当您的代码引发错误时，Lambda 将生成错误的 JSON 表示形式。此错误文档会出现在调用日志中，对于同步调用，它出现在输出中。

Example function.rb

```

def handler(event:, context:)
  puts "Processing event..."
  [1, 2, 3].first("two")
  "Success"
end

```

此代码将导致类型错误。Lambda 将捕获此错误并生成一个包含错误消息、类型和堆栈跟踪字段的 JSON 文档。

```

{
  "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'"
  ]
}

```

}

在您从命令行调用函数时，AWS CLI 将响应拆分为两个文档。为指示出现函数错误，在终端中显示的响应包含 `FunctionError` 字段。函数返回的响应或错误写入到输出文件。

```
$ aws lambda invoke --function-name my-function out.json
{
    "StatusCode": 200,
    "FunctionError": "Unhandled",
    "ExecutedVersion": "$LATEST"
}
```

查看输出文件以查看错误文档。

```
$ cat out.json
{"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'"]}
```

Note

来自 Lambda 的响应中的 200 (成功) 状态代码指示您发送到 Lambda 的请求没有出错。有关导致错误状态代码的问题，请参阅 [Errors \(p. 484\)](#)。

Lambda 还会在函数日志中记录错误对象，最多 256 KB。要在从命令行调用函数时查看日志，请使用 `--log-type` 选项并解码响应中的 base64 字符串。

```
$ aws lambda invoke --function-name my-function out.json --log-type Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 5ce6a15a-f156-11e8-b8aa-25371a5ca2a3 Version: $LATEST
Processing event...
Error raised from handler method
{
    "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'"
    ]
}
END RequestId: 5ce6a15a-f156-11e8-b8aa-25371a5ca2a3
REPORT RequestId: 5ce6a15a-f156-11e8-b8aa-25371a5ca2a3 Duration: 22.74 ms      Billed
Duration: 100 ms          Memory Size: 128 MB      Max Memory Used: 18 MB
```

有关日志的更多信息，请参阅 [Ruby 中的 AWS Lambda 函数日志记录 \(p. 295\)](#)。

AWS Lambda 可能会重试失败的 Lambda 函数，具体视事件源而定。例如，如果 Kinesis 为事件源，则 AWS Lambda 会重试失败的调用，直到 Lambda 函数成功或流中的记录过期。有关重试的更多信息，请参阅 [AWS Lambda 中的错误处理和自动重试 \(p. 98\)](#)。

使用 Java 构建 Lambda 函数

您可以在 AWS Lambda 中运行 Java 代码。Lambda 为执行代码来处理事件的 Java 提供[运行时 \(p. 108\)](#)。您的代码在一个 Amazon Linux 环境中运行，该环境包含来自您所管理的 AWS Identity and Access Management (IAM) 的角色的 AWS 凭证。

Lambda 支持以下 Java 运行时。

Java 运行时

名称	标识符	JDK	操作系统
Java 11	java11	amazon-corretto-11	Amazon Linux 2
Java 8	java8	java-1.8.0-openjdk	Amazon Linux

Lambda 函数使用[执行角色 \(p. 30\)](#)来获取将日志写入 Amazon CloudWatch Logs 以及访问其他服务和资源的权限。如果您还没有函数开发的执行角色，请创建一个。

创建执行角色

1. 打开 IAM 控制台中的[“角色”页面](#)。
2. 选择 Create role (创建角色)。
3. 创建具有以下属性的角色。
 - 可信任的实体 – Lambda。
 - 权限 – AWSLambdaBasicExecutionRole。
 - 角色名称 (角色名称) – **lambda-role**。

AWSLambdaBasicExecutionRole 策略具有函数将日志写入 CloudWatch Logs 所需的权限。

您可以稍后向此角色添加权限，或将其与特定于单一函数的其他角色交换。

创建 Java 函数

1. 打开 [Lambda 控制台](#)。
2. 选择创建函数。
3. 配置以下设置：
 - 名称 – **my-function**。
 - 运行时 – Java 11。
 - 角色 – 选择现有角色。
 - 现有角色 – **lambda-role**。
4. 选择创建函数。
5. 要配置测试事件，请选择测试。
6. 对于事件名称，输入 **test**。
7. 选择创建。
8. 要执行函数，请选择测试。

控制台会创建具有名为 Hello 的处理程序类的 Lambda 函数。由于 Java 是编译语言，因此您无法在 Lambda 控制台中查看或编辑源代码，但可以修改源代码的配置、调用源代码以及配置触发器。

Hello 类具有一个名为 handleRequest 的函数，此函数接受事件对象和上下文对象。这是在调用函数时 Lambda 调用的[处理程序函数 \(p. 307\)](#)。Java 函数运行时从 Lambda 获取调用事件并将它们传递到处理程序。在函数配置中，处理程序值为 example.Hello::handleRequest。

要更新函数的代码，您需要创建一个部署程序包，这是一个包含函数代码的 ZIP 存档。随着函数开发的进行，您需要将函数代码存储在源代码控制中、添加库和实现部署自动化。首先，通过命令行[创建部署程序包 \(p. 302\)](#)并更新代码。

除了调用事件之外，函数运行时还将上下文对象传递给处理程序。[上下文对象 \(p. 311\)](#)包含有关调用、函数和执行环境的其他信息。环境变量中提供了更多信息。

您的 Lambda 函数附带了 CloudWatch Logs 日志组。函数运行时将发送有关对 CloudWatch Logs 的每个调用的详细信息。它会中继调用期间[函数输出的任何日志 \(p. 313\)](#)。如果您的函数[返回错误 \(p. 319\)](#)，则 Lambda 将为错误设置格式，并将其返回给调用方。

主题

- [Java 中的 AWS Lambda 部署程序包 \(p. 302\)](#)
- [Java 中的 AWS Lambda 函数处理程序 \(p. 307\)](#)
- [Java 中的 AWS Lambda 上下文对象 \(p. 311\)](#)
- [Java 中的 AWS Lambda 函数日志记录 \(p. 313\)](#)
- [Java 中的 AWS Lambda 函数错误 \(p. 319\)](#)
- [在 AWS Lambda 中检测 Java 代码 \(p. 323\)](#)
- [使用 Eclipse 创建部署程序包 \(p. 327\)](#)

Java 中的 AWS Lambda 部署程序包

部署程序包是包含已编译的函数代码和依赖项的 ZIP 存档。您可以将程序包直接上传到 Lambda，也可以使用 Amazon S3 存储桶、然后再将其上传到 Lambda。如果部署包大于 50 MB，则必须使用 Amazon S3。

AWS Lambda 提供以下适用于 Java 函数的库：

- [com.amazonaws:aws-lambda-java-core](#) (必需) – 定义处理程序方法接口和运行时传递给处理程序的上下文对象。如果您定义自己的输入类型，则这是您唯一需要的库。
- [com.amazonaws:aws-lambda-java-events](#) – 来自调用 Lambda 函数的服务的事件的输入类型。
- [com.amazonaws:aws-lambda-java-log4j2](#) – Log4j 2 的 Appender 库，可用于将当前调用的请求 ID 添加到[功能日志 \(p. 313\)](#)中。

这些库可通过[Maven 中央存储库](#)获得。将它们添加到您的构建定义中，如下所示。

Gradle

```
dependencies {
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.0'
    implementation 'com.amazonaws:aws-lambda-java-events:2.2.7'
    runtimeOnly 'com.amazonaws:aws-lambda-java-log4j2:1.1.0'
}
```

Maven

```
<dependencies>
```

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-core</artifactId>
    <version>1.2.0</version>
</dependency>
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-events</artifactId>
    <version>2.2.7</version>
</dependency>
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-log4j2</artifactId>
    <version>1.1.0</version>
</dependency>
</dependencies>
```

要创建部署程序包，请将函数代码和依赖项编译成单个 ZIP 或 Java 存档 (JAR) 文件。对于 Gradle，请使用 [Zip 构建类型 \(p. 303\)](#)。对于 Maven，请使用 [Maven Shade 插件 \(p. 304\)](#)。

Note

为了减小部署程序包的大小，请将函数的依赖项打包到层中。层可让您独立管理依赖项，可以供多个函数使用，并且可以与其他账户共享。有关详细信息，请参阅[AWS Lambda 层 \(p. 68\)](#)。

您可以使用 Lambda 控制台、Lambda API 或 AWS SAM 上传部署程序包。

使用 Lambda 控制台上传部署程序包

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Function code (函数包) 下，选择 Upload (上传)。
4. 构建部署程序包。
5. 选择保存。

小节目录

- [使用 Gradle 构建部署程序包 \(p. 303\)](#)
- [使用 Maven 构建部署程序包 \(p. 304\)](#)
- [使用 Lambda API 上传部署程序包 \(p. 305\)](#)
- [使用 AWS SAM 上传部署程序包 \(p. 306\)](#)

使用 Gradle 构建部署程序包

使用 zip 构建类型以创建包含函数代码和依赖项的部署程序包。

Example build.gradle – 构建任务

```
task buildZip(type: Zip) {
    from compileJava
    from processResources
    into('lib') {
        from configurations.runtimeClasspath
    }
}
```

此构建配置在 build/distributions 文件夹中生成部署程序包。compileJava 任务编译函数的类。这些 processResources 任务将构建的类路径中的库复制到名为 lib 的文件夹中。

Example build.gradle – 依赖项

```
dependencies {
    implementation platform('software.amazon.awssdk:bom:2.10.73')
    implementation 'software.amazon.awssdk:lambda'
    implementation 'com.amazonaws:aws-lambda-java-core:1.2.0'
    implementation 'com.amazonaws:aws-lambda-java-events:2.2.7'
    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-log4j2:1.1.0'
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.6.0'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.6.0'
}
```

Lambda 按 Unicode 字母顺序加载 JAR 文件。如果 lib 文件夹中的多个 JAR 文件包含相同的类，则使用第一个。可以使用以下 shell 脚本来识别重复类。

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
```

使用 Maven 构建部署程序包

要使用 Maven 构建部署程序包，请使用 [Maven Shade 插件](#)。该插件创建一个包含编译的函数代码及其所有依赖项的 JAR 文件。

Example pom.xml – 插件配置

```
<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>
```

要构建部署程序包，请使用 mvn package 命令。

```
[INFO] Scanning for projects...
[INFO] -----
[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:1.2.0 in the shaded jar.
[INFO] Including com.amazonaws:aws-lambda-java-events:jar:2.2.7 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] -----
```

此命令在 target 文件夹中生成 JAR 文件。

如果您使用 Appender 库 (aws-lambda-java-log4j2)，还必须为 Maven Shade 插件配置一个转换器。转换器库合并同时出现在 Appender 库和 Log4j 中的缓存文件的版本。

Example pom.xml – 具有 Log4j 2 Appender 的插件配置

```
<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>
```

使用 Lambda API 上传部署程序包

要使用 AWS CLI 或 AWS 开发工具包更新函数的代码，请使用 [UpdateFunctionCode \(p. 553\)](#) API 操作。对于 AWS CLI，请使用 update-function-code 命令。以下命令上传当前目录中名为 my-function.zip 的部署程序包。

```
~/my-function$ aws lambda update-function-code --function-name my-function --zip-file fileb://my-function.zip
{
    "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",
    ...
}
```

如果您的部署程序包大于 50 MB，则无法直接上传。将其上传到 Amazon S3 存储桶并将 Lambda 指向此对象。以下示例命令将部署程序包上传到名为 my-bucket 的存储桶，并使用它更新函数的代码。

```
~/my-function$ aws s3 cp my-function.zip s3://my-bucket
upload: my-function.zip to s3://my-bucket/my-function
~/my-function$ aws lambda update-function-code --function-name my-function \
--s3-bucket my-bucket --s3-key my-function.zip
{
    "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",
    ...
}
```

您可以使用此方法上传最大为 250 MB 的函数包（已解压缩）。

使用 AWS SAM 上传部署程序包

您可以使用 AWS 无服务器应用程序模型 自动部署函数代码、配置和依赖项。AWS SAM 是 AWS CloudFormation 的一个扩展，它提供用于定义无服务器应用程序的简化语法。以下示例模板在 Gradle 使用的 build/distributions 目录中定义了一个包含部署程序包的函数。

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
- AWSLambdaReadOnlyAccess
- AWSXrayWriteOnlyAccess
- AWSLambdaVPCAccessExecutionRole
Tracing: Active
```

要创建函数，请使用 `package` 和 `deploy` 命令。这些命令是对 AWS CLI 的自定义。它们包装其他命令以将部署程序包上传到 Amazon S3，使用对象 URI 重写模板，然后更新函数的代码。

以下示例脚本运行 Gradle 构建并上传其创建的部署程序包。它在您第一次运行它时创建一个 AWS CloudFormation 堆栈。如果堆栈已经存在，脚本会更新它。

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
```

有关完整的工作示例，请参阅以下示例应用程序。

Java 示例应用程序

- [java-basic](#) – 具有单元测试和可变日志记录配置的最小 Java 函数。包括 Gradle 和 Maven 版本。
- [java-events](#) – 一个最小的 Java 函数，它将 [aws-lambda-java-events \(p. 302\)](#) 库与不需要 AWS 开发工具包作为依赖项的事件类型（例如 Amazon API Gateway 和 Amazon Simple Notification Service）结合使用。
- [java-events-v1sdk](#) – 一个 Java 函数，它将 [aws-lambda-java-events \(p. 302\)](#) 库与需要 AWS 开发工具包作为依赖项的事件类型（Amazon Simple Storage Service、Amazon DynamoDB 和 Amazon Kinesis）结合使用。
- [blank-java](#) – 一个 Java 函数，具有事件库、高级日志记录配置以及调用 Lambda API 以检索账户设置的适用于 Java 的 AWS 开发工具包 2.x。
- [s3-java](#) – 一个 Java 函数，它处理来自 Amazon S3 的通知事件，并使用 Java 类库 (JCL) 从上传的图像文件创建缩略图。

Java 中的 AWS Lambda 函数处理程序

您的 Lambda 函数的处理程序是函数代码中处理事件的方法。当调用函数时，Lambda 运行处理程序方法。当处理程序退出或返回响应时，它可用于处理另一个事件。

在以下示例中，名为 `Handler` 的类定义名为 `handleRequest` 的处理程序方法。处理程序方法接受一个事件和上下文对象作为输入并返回一个字符串。

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;
    }
}
```

[Lambda 运行时 \(p. 108\)](#) 将以 JSON 格式的字符串接收事件，并将其转换为对象。它将事件对象以及提供有关调用和函数详细信息的上下文对象传递给函数处理程序。您通过在函数的配置上设置处理程序参数来告诉运行时要调用哪个方法。

处理程序格式

- [package.Class::method](#) – 完整格式。例如：`example.Handler::handleRequest`。
- [package.Class](#) – 实现[处理程序接口 \(p. 309\)](#)的函数的缩写格式。例如：`example.Handler`。

您可以在处理程序方法外部添加[初始化代码 \(p. 17\)](#)，以便跨多个调用重用资源。当运行时加载处理程序时，它会运行静态代码和类构造函数。在初始化期间创建的资源在调用之间保留在内存中，处理程序可以重复使用这些资源数千次。

在以下示例中，当函数提供其第一个事件时，会创建记录器、序列化程序和 AWS 开发工具包客户端。由同一函数实例提供的后续事件要快得多，因为这些资源已经存在。

Example Handler.java – 初始化代码

```
// 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()));
        ...
    }
}
```

本指南的 GitHub 存储库提供了易于部署的示例应用程序，用于演示各种处理程序类型。有关详细信息，请参阅[本主题的末尾 \(p. 310\)](#)。

小目录

- [选择输入和输出类型 \(p. 309\)](#)
- [处理程序接口 \(p. 309\)](#)

- [示例处理程序代码 \(p. 310\)](#)

选择输入和输出类型

您可以在处理程序方法的签名中指定事件映射到的对象类型。在上述示例中，Java 运行时将事件反序列化为实现 `Map<String, String>` 接口的类型。字符串到字符串映射适用于平面事件，如下所示：

Example `Event.json` – 天气数据

```
{  
    "temperatureK": 281,  
    "windKmh": -3,  
    "humidityPct": 0.55,  
    "pressureHPa": 1020  
}
```

但是，每个字段的值必须是字符串或数字。如果事件包含具有对象作为值的字段，则运行时无法对该字段进行反序列化并返回错误。

选择与函数处理的事件数据一起使用的输入类型。您可以使用基本类型、泛型类型或明确定义的类型。

输入类型

- `Integer`、`Long`、`Double` 等 – 事件是一个没有其他格式的数字 — 例如，`3.5`。运行时将值转换为指定类型的对象。
- `String` – 事件是 JSON 字符串，包括引号 — 例如，`"My string."`。运行时将值（不带引号）转换为 `String` 对象。
- `Type`、`Map<String, Type>` 等 – 事件是一个 JSON 对象。运行时将其反序列化为指定类型或接口的对象。
- `List<Integer>`、`List<String>`、`List<Object>` 等 – 事件是一个 JSON 数组。运行时将其反序列化为指定类型或接口的对象。
- `InputStream` – 事件是任何 JSON 类型。运行时将文档的字节流传递给处理程序而不进行修改。您可以对输入进行反序列化并将输出写到输出流。
- 库类型 – 对于 AWS 服务发送的事件，请使用 [aws-lambda-java-events \(p. 302\)](#) 库中的类型。

如果您定义了自己的输入类型，它应该是可反序列化的、可变的普通旧 Java 对象 (POJO)，对事件中的每个字段具有默认的构造函数和属性。事件中未映射到属性的键以及未包含在事件中的属性将被删除而不显示错误。

输出类型可以是对象或 `void`。运行时将返回值序列化为文本。如果输出是具有字段的对象，运行时将其序列化为 JSON 文档。如果它是包装原始值的类型，则运行时返回该值的文本表示形式。

处理程序接口

`aws-lambda-java-core` 库为处理程序方法定义了两个接口。使用提供的接口简化处理程序配置，并在编译时验证处理程序方法签名。

- `com.amazonaws.services.lambda.runtime.RequestHandler`
- `com.amazonaws.services.lambda.runtime.RequestStreamHandler`

`RequestHandler` 接口是一个泛型类型，它采用两个参数：输入类型和输出类型。这两种类型都必须是对象。使用此接口时，Java 运行时会将事件反序列化为具有输入类型的对象，并将输出序列化为文本。当内置序列化与输入和输出类型配合使用时，请使用此接口。

Example Handler.java – 处理程序接口

```
// Handler value: example.Handler
public class Handler implements RequestHandler<Map<String, String>, String>{
    @Override
    public String handleRequest(Map<String, String> event, Context context)
```

要使用您自己的序列化，请实现 `RequestStreamHandler` 接口。使用此接口，Lambda 将向您的处理程序传递输入流和输出流。处理程序从输入流读取字节，写到输出流，并返回 `void`。

以下示例使用缓冲读取器和写入器类型来处理输入和输出流。它使用 [Gson](#) 库进行序列化和反序列化。

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 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();
        }
    }
}
```

示例处理程序代码

本指南的 GitHub 存储库包括演示如何使用各种处理程序类型和接口的示例应用程序。每个示例应用程序都包含用于轻松部署和清理的脚本、一个 AWS SAM 模板和支持资源。

Java 示例应用程序

- [java-basic](#) – 具有单元测试和可变日志记录配置的最小 Java 函数。包括 Gradle 和 Maven 版本。

- **java-events** – 一个最小的 Java 函数，它将 [aws-lambda-java-events \(p. 302\)](#) 库与不需要 AWS 开发工具包作为依赖项的事件类型（例如 Amazon API Gateway 和 Amazon Simple Notification Service）结合使用。
- **java-events-v1sdk** – 一个 Java 函数，它将 [aws-lambda-java-events \(p. 302\)](#) 库与需要 AWS 开发工具包作为依赖项的事件类型（Amazon Simple Storage Service、Amazon DynamoDB 和 Amazon Kinesis）结合使用。
- **blank-java** – 一个 Java 函数，具有事件库、高级日志记录配置以及调用 Lambda API 以检索账户设置的适用于 Java 的 AWS 开发工具包 2.x。
- **s3-java** – 一个 Java 函数，它处理来自 Amazon S3 的通知事件，并使用 Java 类库 (JCL) 从上传的图像文件创建缩略图。

`blank-java` 和 `s3-java` 应用程序将 AWS 服务事件作为输入并返回字符串。`java-basic` 应用程序包括几种类型的处理器：

- `Handler.java` – 以 `Map<String, String>` 作为输入。
- `HandlerInteger.java` – 以 `Integer` 作为输入。
- `HandlerList.java` – 以 `List<Integer>` 作为输入。
- `HandlerStream.java` – 以 `InputStream` 和 `OutputStream` 作为输入。
- `HandlerString.java` – 以 `String` 作为输入。
- `HandlerWeatherData.java` – 以自定义类型作为输入。

要测试不同的处理器类型，只需更改 AWS SAM 模板中的处理器值即可。有关详细说明，请参阅示例应用程序的自述文件。

Java 中的 AWS Lambda 上下文对象

当 Lambda 运行您的函数时，它会将上下文对象传递到[处理器 \(p. 307\)](#)。此对象提供的方法和属性包含有关调用、函数和执行环境的信息。

上下文方法

- `getRemainingTimeInMillis()` – 返回在执行超时以前剩余的毫秒数。
- `getFunctionName()` – 返回 Lambda 函数的名称。
- `getFunctionVersion()` – 返回函数的版本 (p. 63)。
- `getInvokedFunctionArn()` – 返回用于调用函数的 Amazon 资源名称 (ARN)。指示调用方是否已指定版本或别名。
- `getMemoryLimitInMB()` – 返回为函数分配的内存量。
- `getAwsRequestId()` – 返回调用请求的标识符。
- `getLogGroupName()` – 返回函数的日志组。
- `getLogStreamName()` – 返回函数实例的日志流。
- `getIdentity()` – (移动应用程序) 返回有关授权请求的 Amazon Cognito 身份的信息。
- `getClientContext()` – (移动应用程序) 返回客户端应用程序向 Lambda 提供的客户端上下文。
- `getLogger()` – 返回函数的记录器对象 (p. 313)。

以下示例显示一个使用上下文对象访问 Lambda 记录器的函数。

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;
    }
}

```

该函数将上下文对象序列化为 JSON 并将其记录在其日志流中。

Example 日志输出

```

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

```

上下文对象的接口在 [aws-lambda-java-core](#) 库中可用。您可以实现此接口来创建用于测试的上下文类。以下示例显示一个上下文类，该类返回大多数属性的虚拟值和一个有效的测试记录器。

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();
}

}
```

有关日志记录的更多信息，请参阅[Java 中的 AWS Lambda 函数日志记录 \(p. 313\)](#)。

示例应用程序中的上下文

本指南的 GitHub 存储库包括演示如何使用上下文对象的示例应用程序。每个示例应用程序都包含用于轻松部署和清理的脚本、一个 AWS SAM 模板和支持资源。

Java 示例应用程序

- [java-basic](#) – 具有单元测试和可变日志记录配置的最小 Java 函数。包括 Gradle 和 Maven 版本。
- [java-events](#) – 一个最小的 Java 函数，它将 [aws-lambda-java-events \(p. 302\)](#) 库与不需要 AWS 开发工具包作为依赖项的事件类型（例如 Amazon API Gateway 和 Amazon Simple Notification Service）结合使用。
- [java-events-v1sdk](#) – 一个 Java 函数，它将 [aws-lambda-java-events \(p. 302\)](#) 库与需要 AWS 开发工具包作为依赖项的事件类型（Amazon Simple Storage Service、Amazon DynamoDB 和 Amazon Kinesis）结合使用。
- [blank-java](#) – 一个 Java 函数，具有事件库、高级日志记录配置以及调用 Lambda API 以检索账户设置的适用于 Java 的 AWS 开发工具包 2.x。
- [s3-java](#) – 一个 Java 函数，它处理来自 Amazon S3 的通知事件，并使用 Java 类库 (JCL) 从上传的图像文件创建缩略图。

所有示例应用程序都有一个用于单元测试的测试上下文类。[java-basic](#) 应用程序显示使用上下文对象获取记录器。它使用 SLF4J 和 Log4J 2 提供适用于本地单元测试的记录器。

Java 中的 AWS Lambda 函数日志记录

您的 Lambda 函数带有一个 CloudWatch Logs 日志组，其中包含您的函数的每个实例的日志流。运行时会将每个调用的详细信息发送到该日志流，然后中继日志和来自您的函数代码的其他输出。

要从函数代码输出日志，您可以使用 `java.lang.System` 的方法，或使用写入到 `stdout` 或 `stderr` 的任何日志记录模块。[aws-lambda-java-core \(p. 302\)](#) 库提供一个名为 `LambdaLogger` 的记录器类，您可以从上下文对象访问该类。记录器类支持多行日志。

以下示例使用上下文对象提供的 `LambdaLogger` 记录器。

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 日志格式

```
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
```

Java 运行时记录每次调用的 START、END 和 REPORT 行。报告行提供了以下详细信息。

报告日志

- RequestId – 调用的唯一请求 ID。
- Duration (持续时间) – 函数的处理程序方法处理事件所花费的时间。
- Billed Duration (计费持续时间) – 针对调用计费的时间量。
- Memory Size (内存大小) – 分配给函数的内存量。
- Max Memory Used (最大内存使用量) – 函数使用的内存量。
- Init Duration (初始持续时间) – 对于提供的第一个请求，为运行时在处理程序方法外部加载函数和运行代码所花费的时间。
- XRAY Traceld – 对于跟踪的请求，为 [AWS X-Ray 跟踪编码 \(p. 371\)](#)。
- SegmentId – 对于跟踪的请求，为 X-Ray 分段 ID。
- Sampled (采样) – 对于跟踪的请求，为采样结果。

您可以在 Lambda 控制台中、在 CloudWatch Logs 控制台中或从命令行查看日志。

小节目录

- [在 AWS 管理控制台中查看日志 \(p. 315\)](#)
- [使用 AWS CLI \(p. 315\)](#)

- [删除日志 \(p. 316\)](#)
- [使用 Log4j 2 和 SLF4J 进行高级日志记录 \(p. 316\)](#)
- [日志记录代码示例 \(p. 318\)](#)

在 AWS 管理控制台中查看日志

当您在函数配置页上测试函数时，Lambda 控制台会显示日志输出。要查看所有调用的日志，请使用 CloudWatch Logs 控制台。

查看 Lambda 函数的日志

1. 打开 [CloudWatch 控制台的日志页面](#)。
2. 选择您的函数 (`/aws/lambda/function-name`) 的日志组。
3. 选择列表中的第一个流。

每个日志流对应一个[函数实例 \(p. 109\)](#)。当您更新函数以及创建更多实例以处理多个并发调用时，会显示新的流。要找到特定调用的日志，您可以使用 X-Ray 分析您的函数并在跟踪中记录有关请求和日志流的详细信息。如需将日志和跟踪与 X-Ray 相关联的示例应用程序，请参阅[AWS Lambda 错误处理器示例应用程序 \(p. 128\)](#)。

使用 AWS CLI

要从命令行获取调用的日志，请使用 `--log-type` 选项。响应包含一个 `LogResult` 字段，该字段包含来自调用的多达 4 KB 的 base64 编码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult": "U1RBULQgUmVxdWVzdElkOia4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

您可以使用 `base64` 实用程序来解码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
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: 100 ms Memory Size: 128 MB Max Memory Used: 73 MB
```

`base64` 实用工具在 Linux、macOS 和 [Ubuntu on Windows](#) 上可用。对于 macOS，命令为 `base64 -D`。

要从命令行获取完整的日志事件，您可以在函数输出中包含日志流名称，如上例中所示。以下示例脚本调用名为 `my-function` 的函数并下载最后 5 个日志事件。

Example get-logs.sh 脚本

此示例要求 `my-function` 返回日志流 ID。

```
#!/bin/bash
```

```
aws lambda invoke --function-name my-function --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
```

此脚本使用 sed 从输出文件中删除引号，并休眠 15 秒以等待日志可用。输出包括来自 Lambda 的响应，以及来自 get-log-events 命令的输出。

```
$ ./get-logs.sh
{
  "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: 100 ms\tMemory Size: 128 MB\tMax Memory Used: 75 MB\t",
      "ingestionTime": 1559763018353
    }
  ],
  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
  "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

删除日志

在删除函数时，不会自动删除日志组。为避免存储无限的日志，请删除日志组或者[配置保留期](#)，在该期限后会自动删除日志。

使用 Log4j 2 和 SLF4J 进行高级日志记录

要自定义日志输出、在单元测试期间支持日志记录以及记录 AWS 开发工具包调用，请将 Apache Log4j 2 与 SLF4J 结合使用。Log4j 是 Java 程序的日志库，这些程序使您能够配置日志级别和使用 Appender 库。SLF4J 是一个 Facade 库，可让您更改您使用的库，而不更改函数代码。

要将请求 ID 添加到函数的日志中，请使用 [aws-lambda-java-log4j2 \(p. 302\)](#) 库中的 Appender。以下示例显示向所有日志添加时间戳和请求 ID 的 Log4j 2 配置文件。

Example [src/main/resources/log4j2.xml](#) – Appender 配置

```
<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>
```

使用此配置，每行都会在前面加上日期、时间、请求 ID、日志级别和类名。

Example Appender 的日志格式

```
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",
  ...
}
```

SLF4J 是一个用于在 Java 代码中进行日志记录的 Facade 库。在函数代码中，您可以使用 SLF4J 记录器工厂，通过适用于日志级别（`info()` 和 `warn()`）的方法来检索记录器。在构建配置中，您可以在类路径中包含日志记录库和 SLF4J 适配器。通过更改构建配置中的库，您可以在不更改函数代码的情况下更改记录器类型。SLF4J 需要从适用于 Java 的开发工具包中捕获日志。

在以下示例中，处理程序类使用 SLF4J 检索记录器。

Example [src/main/java/example/Handler.java](#) – 使用 SLF4J 进行日志记录

```
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()));
    ...
}
```

构建配置接受 Lambda Appender 和 SLF4J 适配器上的运行时依赖项以及 Log4J 2 上的实现依赖项。

Example `build.gradle` – 日志记录依赖项

```
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:1.2.0'
    implementation 'com.amazonaws:aws-lambda-java-events:2.2.7'
    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-log4j2:1.1.0'
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.6.0'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.6.0'
}
```

当您在本地运行您的代码进行测试时，带有 Lambda 记录器的上下文对象不可用，并且没有请求 ID 供 Lambda Appender 使用。有关测试配置示例，请参阅下一节中的示例应用程序。

日志记录代码示例

本指南的 GitHub 存储库包括演示如何使用各种日志记录配置的示例应用程序。每个示例应用程序都包含用于轻松部署和清理的脚本、一个 AWS SAM 模板和支持资源。

Java 示例应用程序

- [java-basic](#) – 具有单元测试和可变日志记录配置的最小 Java 函数。包括 Gradle 和 Maven 版本。
- [java-events](#) – 一个最小的 Java 函数，它将 [aws-lambda-java-events \(p. 302\)](#) 库与不需要 AWS 开发工具包作为依赖项的事件类型（例如 Amazon API Gateway 和 Amazon Simple Notification Service）结合使用。
- [java-events-v1sdk](#) – 一个 Java 函数，它将 [aws-lambda-java-events \(p. 302\)](#) 库与需要 AWS 开发工具包作为依赖项的事件类型（Amazon Simple Storage Service、Amazon DynamoDB 和 Amazon Kinesis）结合使用。
- [blank-java](#) – 一个 Java 函数，具有事件库、高级日志记录配置以及调用 Lambda API 以检索账户设置的适用于 Java 的 AWS 开发工具包 2.x。
- [s3-java](#) – 一个 Java 函数，它处理来自 Amazon S3 的通知事件，并使用 Java 类库 (JCL) 从上传的图像文件创建缩略图。

`java-basic` 示例应用程序显示支持日志记录测试的最小日志记录配置。处理程序代码使用上下文对象提供的 `LambdaLogger` 记录器。对于测试，应用程序使用一个自定义 `TestLogger` 类，此类实现带有 Log4j 2

记录器的 `LambdaLogger` 接口。它使用 SLF4J 作为 Facade 以与 AWS 开发工具包兼容。从构建输出中排除日志记录库，以使部署程序包保持较小。

`blank-java` 示例应用程序基于具有 AWS 开发工具包日志记录和 Lambda Log4j 2 Appender 的基本配置而构建。它使用 Lambda 中的 Log4j 2 和自定义 Appender，该 Appender 将调用请求 ID 添加到每行。

Java 中的 AWS Lambda 函数错误

当您的函数引发错误时，Lambda 将有关此错误的详细信息返回给调用者。Lambda 返回的响应的正文包含一个 JSON 文档，其中包含错误名称、错误类型和堆栈帧数组。调用该函数的客户端或服务可以处理错误或将其一直传递到最终用户。您可以使用自定义例外向用户返回客户端错误的有用信息。

Example [src/main/java/example/HandlerDivide.java](#) – 运行时异常

```
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;
    }
}
```

当函数引发 `InputLengthException` 时，Java 运行时将其序列化到以下文档中。

Example 错误文档（已添加空格）

```
{
    "errorMessage": "Input must contain 2 numbers.",
    "errorType": "java.lang.InputLengthException",
    "stackTrace": [
        "example.HandlerDividehandleRequest(HandlerDivide.java:23)",
        "example.HandlerDividehandleRequest(HandlerDivide.java:14)"
    ]
}
```

在此示例中，`InputLengthException` 为 `RuntimeException`。`RequestHandler` 接口 (p. 309) 不允许检查到的异常。`RequestStreamHandler` 接口支持引发 `IOException` 错误。

上面示例中的返回语句也可以引发运行时异常：

```
return numerator/denominator;
```

此代码可以返回算术错误。

```
{"errorMessage":"/ by zero","errorType":"java.lang.ArithmetricException","stackTrace":[]["example.HandlerDivide.handleRequest(HandlerDivide.java:28)","example.HandlerDivide.handleRequest(HandlerDivide.java:23)"]}
```

小节目录

- [查看错误输出 \(p. 320\)](#)
- [了解错误类型和来源 \(p. 321\)](#)
- [客户端中的错误处理 \(p. 322\)](#)
- [其他 AWS 服务中的错误处理 \(p. 323\)](#)
- [示例应用程序中的错误处理 \(p. 323\)](#)

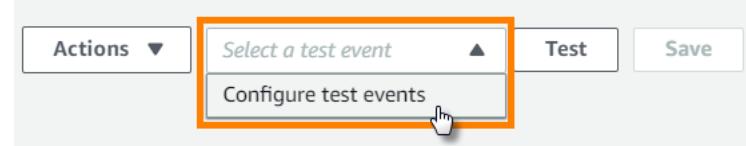
查看错误输出

您可以在 Lambda 控制台、命令行或 AWS 开发工具包中使用测试负载调用您的函数和查看错误输出。错误输出也会捕获到函数的执行日志中，当启用 [跟踪 \(p. 323\)](#) 时，会捕获到 AWS X-Ray 中。

要在 Lambda 控制台中查看错误输出，请使用测试事件调用它。

在 Lambda 控制台中调用函数

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 Test (测试) 按钮旁的下拉菜单中，选择 Configure test events (配置测试事件)。



4. 选择与函数处理的事件匹配的 Event template (事件模板)。
5. 输入测试事件的名称并根据需要修改事件结构。
6. 选择 Create。
7. 选择 Test。

Lambda 控制台 [同步 \(p. 81\)](#) 调用您的函数并显示结果。要查看响应、日志和其他信息，请展开 Details (详细信息) 部分。

在您从命令行调用函数时，AWS CLI 将响应拆分为两个文档。为指示出现函数错误，在终端中显示的响应包含 `FunctionError` 字段。函数返回的响应或错误写入到输出文件。

```
$ aws lambda invoke --function-name my-function out.json
{
    "StatusCode": 200,
    "FunctionError": "Unhandled",
    "ExecutedVersion": "$LATEST"
}
```

查看输出文件以查看错误文档。

```
$ cat out.json
{"errorMessage":"Input must contain 2 numbers.", "errorType":"java.lang.InputLengthException", "stackTrace":[]["example.HandlerDivide.handleRequest(HandlerDivide.java:23)", "example.HandlerDivide.handleRequest(HandlerDivide.java:23)"]}
```

Lambda 还会在函数日志中记录错误对象，最多 256 KB。要在从命令行调用函数时查看日志，请使用 `--log-type` 选项并解码响应中的 base64 字符串。

```
$ aws lambda invoke --function-name my-function --payload "[100,0]" out.json --log-type
Tail \
--query 'LogResult' --output text | base64 -d
START RequestId: 081f7522-xmpl-48e2-8f67-96686904bb4f Version: $LATEST
EVENT: [
  100,
  0
]EVENT TYPE: class java.util.ArrayList/ by zero: java.lang.ArithmetricException
java.lang.ArithmetricException: / by zero
    at example.HandlerDivide.handleRequest(HandlerDivide.java:28)
    at example.HandlerDivide.handleRequest(HandlerDivide.java:13)

END RequestId: 081f7522-xmpl-48e2-8f67-96686904bb4f
REPORT RequestId: 081f7522-xmpl-48e2-8f67-96686904bb4f Duration: 4.20 ms      Billed
Duration: 100 ms Memory Size: 512 MB      Max Memory Used: 95 MB
XRAY TraceId: 1-5e73162b-1919xmpl2592f4549e1c39be      SegmentId: 3dadxmpl48126cb8
Sampled: true
```

有关日志的更多信息，请参阅[Java 中的 AWS Lambda 函数日志记录 \(p. 313\)](#)。

了解错误类型和来源

当您调用函数时，多个子系统会处理请求、事件、输出和响应。错误可能来自您的 Lambda 服务（调用错误）或函数的实例。函数错误包括处理程序代码返回的异常和 Lambda 运行时返回的异常。

Lambda 服务接收调用请求并对其进行验证。它检查权限，验证事件文档是否为有效的 JSON 文档，并检查参数值。如果 Lambda 服务遇到错误，它会返回指示错误原因的异常类型、消息和 HTTP 状态代码。

Note

有关 Invoke API 操作可能返回的错误的完整列表，请参阅 Lambda API 参考中的[调用错误 \(p. 484\)](#)。

来自 Lambda 服务的 4xx 系列错误表示调用者可以通过修改请求、请求权限或重试来修复的错误。5xx 系列错误表示 Lambda 服务存在问题，或者函数的配置或资源存在问题。调用方无法解决这些问题，但函数的所有者可以解决它们。

如果请求通过验证，Lambda 将其发送到函数的实例。运行时将事件文档转换为一个对象，并将该对象传递给处理程序代码。在此过程中可能会发生错误，例如，如果处理程序方法的名称与函数的配置不匹配，或者如果调用在处理程序代码返回响应之前发生超时。Lambda 运行时错误的格式类似于您的代码返回的错误，但由运行时返回。

在以下示例中，运行时无法将事件反序列化为对象。输入是一个有效的 JSON 类型，但它与处理程序方法预期的类型不匹配。

Example Lambda 运行时错误

```
{
  "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\nfrom String value '1000,10': not a valid Integer value\\n at [Source:\nlambdainternal.util.NativeMemoryAsInputStream@35fc6dc4; line: 1, column: 1] (through\nreference 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.ja
        ...
    ]
}
}
```

对于 Lambda 运行时错误和其他函数错误，Lambda 服务不返回错误代码。2xx 系列状态代码表示 Lambda 服务接受了请求。Lambda 通过在响应中包含 `X-Amz-Function-Error` 标头（而不是错误代码）来指示错误。

对于异步调用，事件在 Lambda 将它们发送到函数之前排队。对于有效的请求，Lambda 立即返回成功响应并将事件添加到队列中。然后，Lambda 从队列中读取事件并调用该函数。如果发生错误，则 Lambda 会重试行为，而行为根据错误类型而异。有关更多信息，请参阅[异步调用 \(p. 83\)](#)。

客户端中的错误处理

调用 Lambda 函数的客户端可以选择处理错误或将错误传递给最终用户。正确的错误处理行为取决于应用程序的类型、受众以及错误来源。例如，如果调用失败且出现错误代码 429（请求太多），AWS CLI 会重试最多 4 次，然后向用户显示错误。

```
$ aws lambda invoke --function-name my-function out.json
An error occurred (TooManyRequestsException) when calling the Invoke operation (reached max
retries: 4): Rate Exceeded.
```

对于其他调用错误，正确的行为取决于响应代码。5xx 系列错误可能表示用户无需采取任何操作即可解决的临时状况。重试可能会成功，也可能会不成功。4xx 系列错误（而不是 429）通常表示请求发生错误。重试不可能成功。

对于函数错误，客户端可以处理错误文档并以用户友好的格式显示错误消息。基于浏览器的应用程序可能会显示错误消息和类型，但忽略堆栈跟踪。AWS CLI 将错误对象保存到输出文件中，而显示从响应标头生成的文档。

```
$ aws lambda invoke --function-name my-function --payload '[1000]' out.json
{
    "StatusCode": 200,
    "FunctionError": "Unhandled",
    "ExecutedVersion": "$LATEST"
}
```

Example out.json

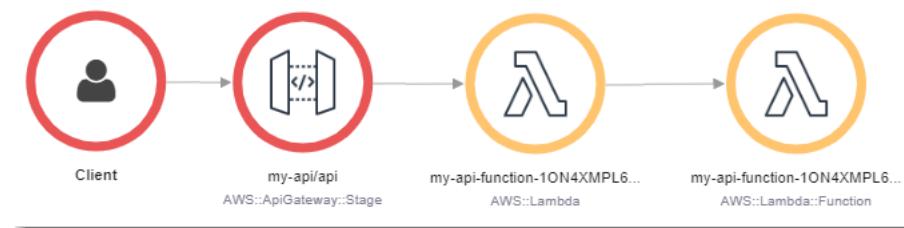
```
{"errorMessage":"Input must be an array that contains 2
numbers.", "errorType":"example.InputLengthException", "stackTrace": [
"example.HandlerDivide.handleRequest(HandlerDivide.java:22)", "example.HandlerDivide.handleRequest(Hand
```

其他 AWS 服务中的错误处理

当 AWS 服务调用您的函数时，服务会选择调用类型和重试行为。AWS 服务可以按计划调用您的函数，以响应资源上的生命周期事件或者针对来自用户的请求提供响应。某些服务异步调用函数并让 Lambda 处理错误，而其他服务则重试或将错误传回给用户。

例如，API 网关 将所有调用和函数错误视为内部错误。如果 Lambda API 拒绝调用请求，则 API 网关 返回 500 错误代码。如果函数运行但返回错误，或返回格式错误的响应，则 API 网关 返回 502。要自定义错误响应，您必须捕获代码中的错误并以所需格式设置响应的格式。

要确定错误来源及其原因，请使用 AWS X-Ray。使用 X-Ray，您可以找出哪些组件遇到错误，并查看有关异常的详细信息。以下示例显示导致来自 API 网关 的 502 响应的函数错误。



通过对您的函数[启用主动跟踪 \(p. 323\)](#)，开始使用 X-Ray。

有关其他服务处理程序错误的详细信息，请参阅[将 AWS Lambda 与其他服务结合使用 \(p. 138\)](#)章节中的主题。

示例应用程序中的错误处理

本指南的 GitHub 存储库包括演示错误使用情况的示例应用程序。每个示例应用程序都包含用于轻松部署和清理的脚本、一个 AWS SAM 模板和支持资源。

Java 示例应用程序

- [java-basic](#) – 具有单元测试和可变日志记录配置的最小 Java 函数。包括 Gradle 和 Maven 版本。
- [java-events](#) – 一个最小的 Java 函数，它将 [aws-lambda-java-events \(p. 302\)](#) 库与不需要 AWS 开发工具包作为依赖项的事件类型（例如 Amazon API Gateway 和 Amazon Simple Notification Service）结合使用。
- [java-events-v1sdk](#) – 一个 Java 函数，它将 [aws-lambda-java-events \(p. 302\)](#) 库与需要 AWS 开发工具包作为依赖项的事件类型（Amazon Simple Storage Service、Amazon DynamoDB 和 Amazon Kinesis）结合使用。
- [blank-java](#) – 一个 Java 函数，具有事件库、高级日志记录配置以及调用 Lambda API 以检索账户设置的适用于 Java 的 AWS 开发工具包 2.x。
- [s3-java](#) – 一个 Java 函数，它处理来自 Amazon S3 的通知事件，并使用 Java 类库 (JCL) 从上传的图像文件创建缩略图。

`java-basic` 函数包括返回自定义运行时异常的处理程序 (`HandlerDivide`)。`HandlerStream` 处理程序实现 `RequestStreamHandler` 并可能引发 `IOException` 检查的异常。

在 AWS Lambda 中检测 Java 代码

Lambda 与 AWS X-Ray 集成使您能够跟踪、调试和优化 Lambda 应用程序。您可以在请求遍历应用程序中的资源（从前端 API 到后端的存储和数据库）时，使用 X-Ray 跟踪请求。只需将 X-Ray 开发工具包库添加到构建配置中，就可以记录您的函数对 AWS 服务进行的任何调用的错误和延迟。

X-Ray 服务地图显示了通过您的应用程序的请求流。来自[错误处理器 \(p. 128\)](#)示例应用程序的以下示例显示了具有两个函数的应用程序。主函数处理事件，有时会返回错误。次函数处理主函数的日志组中显示的错误，并使用 AWS 开发工具包来调用 X-Ray、Amazon S3 和 Amazon CloudWatch Logs。



您可以在函数配置中启用跟踪。

启用活动跟踪

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 AWS X-Ray 下，选择 Active tracing (活动跟踪)。
4. 选择保存。

您的函数需要权限才能将跟踪数据上传到 X-Ray。在 Lambda 控制台中启用活动跟踪后，Lambda 会将所需权限添加到函数的[执行角色 \(p. 30\)](#)。如果没有，请将 `AWSXRayDaemonWriteAccess` 策略添加到执行角色。

X-Ray 应用采样算法确保跟踪有效，同时为应用程序所服务的请求提供代表性样本。默认采样规则是每秒 1 个请求和 5% 的其他请求。

启用活动跟踪后，Lambda 会记录对调用子集的跟踪。Lambda 会记录两个片段，从而在服务地图上创建两个节点。第一个节点表示接收调用请求的 Lambda 服务。第二个节点由函数的[运行时 \(p. 16\)](#)记录。



要记录您的函数对其他资源和服务进行的调用的详细信息，请将适用于 Java 的 X-Ray 开发工具包添加到您的构建配置中。以下示例显示 Gradle 构建配置，其中包括启用适用于 Java 的 AWS 开发工具包 2.x 客户端自动分析的库。

Example `build.gradle` – 跟踪依赖项

```
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'  
    ...  
}
```

您还可以分析 HTTP 客户端、记录 SQL 查询以及使用注释和元数据创建自定义字段。有关更多信息，请参阅 AWS X-Ray 开发人员指南 中的[适用于 Java 的 AWS X-Ray 开发工具包](#)。

使用 Lambda API 启用主动跟踪

要使用 AWS CLI 或 AWS 开发工具包管理跟踪配置，请使用以下 API 操作。

- [UpdateFunctionConfiguration \(p. 560\)](#)
- [GetFunctionConfiguration \(p. 460\)](#)
- [CreateFunction \(p. 421\)](#)

以下示例 AWS CLI 命令对名为 my-function 的函数启用主动跟踪。

```
$ aws lambda update-function-configuration --function-name my-function \  
--tracing-config Mode=Active
```

跟踪模式是发布函数版本时锁定的版本特定配置的一部分。您无法更改已发布版本上的跟踪模式。

使用 AWS CloudFormation 启用主动跟踪

要对 AWS CloudFormation 模板中的 `AWS::Lambda::Function` 资源启用活动跟踪，请使用 `TracingConfig` 属性。

Example `template-std.yml` – 跟踪配置

```
Resources:  
  function:  
    Type: AWS::Lambda::Function  
    Properties:  
      TracingConfig:  
        Mode: Active  
      ...
```

对于 AWS SAM `AWS::Serverless::Function` 资源，请使用 `Tracing` 属性。

Example `template.yml` – 跟踪配置

```
Resources:  
  function:
```

```
Type: AWS::Serverless::Function
Properties:
  CodeUri: build/distributions/blank-java.zip
  Tracing: Active
  ...
  ...
```

如果您使用 X-Ray 开发工具包来分析 AWS 开发工具包客户端和您的函数代码，则您的部署程序包可能会变得相当大。为了避免每次更新函数代码时上传运行时依赖项，请将它们打包到 [Lambda 层 \(p. 68\)](#) 中。以下示例显示存储适用于 Java 的适用于 Java 的开发工具包 和 X-Ray 开发工具包的 AWS::Serverless::LayerVersion 资源。

Example [template.yml](#) – 依赖项层

```
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
```

使用此配置，只有在更改构建依赖项时才会更新库 JAR。当您更新函数代码时，只需上传函数的已编译的类，所以上传时间可以快得多。

为依赖项创建层要求更改构建配置才能在部署之前生成层存档。有关工作示例，请参阅 [java-basic](#) 示例应用程序。

示例应用程序中的跟踪

本指南的 GitHub 存储库包括演示如何使用跟踪的示例应用程序。每个示例应用程序都包含用于轻松部署和清理的脚本、一个 AWS SAM 模板和支持资源。

Java 示例应用程序

- [java-basic](#) – 具有单元测试和可变日志记录配置的最小 Java 函数。包括 Gradle 和 Maven 版本。
- [java-events](#) – 一个最小的 Java 函数，它将 [aws-lambda-java-events \(p. 302\)](#) 库与不需要 AWS 开发工具包作为依赖项的事件类型（例如 Amazon API Gateway 和 Amazon Simple Notification Service）结合使用。
- [java-events-v1sdk](#) – 一个 Java 函数，它将 [aws-lambda-java-events \(p. 302\)](#) 库与需要 AWS 开发工具包作为依赖项的事件类型（Amazon Simple Storage Service、Amazon DynamoDB 和 Amazon Kinesis）结合使用。
- [blank-java](#) – 一个 Java 函数，具有事件库、高级日志记录配置以及调用 Lambda API 以检索账户设置的适用于 Java 的 AWS 开发工具包 2.x。
- [s3-java](#) – 一个 Java 函数，它处理来自 Amazon S3 的通知事件，并使用 Java 类库 (JCL) 从上传的图像文件创建缩略图。

所有示例应用程序都为 Lambda 功能启用了活动跟踪。[blank-java](#) 应用程序显示适用于 Java 的 AWS 开发工具包 2.x 客户端的自动分析、用于测试的段管理、自定义字段以及使用 Lambda 层存储运行时依赖项。



blank-java 示例应用程序中的此示例显示了 Lambda 服务、函数和 Lambda API 的节点。该函数调用 Lambda API 来监控 Lambda 中的存储使用情况。

使用 Eclipse 创建部署程序包

本章节介绍如何使用 Eclipse IDE 和用于 Eclipse 的 Maven 插件将 Java 代码打包到部署程序包中。

Note

AWS SDK Eclipse Toolkit 提供了一个 Eclipse 插件，可用于创建和上传部署程序包，从而创建 Lambda 函数。如果您可以使用 Eclipse IDE 作为开发环境，则使用此插件可以编写 Java 代码、创建并上传部署程序包和创建您的 Lambda 函数。有关更多信息，请参阅 [AWS Toolkit for Eclipse 入门指南](#)。有关使用编写 AWS Lambda 函数的工具包的示例，请参阅 [将 AWS Lambda 与 AWS Toolkit for Eclipse 结合使用](#)。

主题

- [先决条件 \(p. 327\)](#)
- [创建并构建项目 \(p. 327\)](#)

先决条件

安装用于 Eclipse 的 Maven 插件。

1. 启动 Eclipse。从 Eclipse 的 Help 菜单中，选择 Install New Software。
2. 在 Install (安装) 窗口中的 Work with: (使用:) 框中键入 <http://download.eclipse.org/technology/m2e/releases>，然后选择 Add (添加)。
3. 按照步骤操作以完成安装。

创建并构建项目

在该步骤中，启动 Eclipse 并创建一个 Maven 项目。您将添加必要的依赖项并构建该项目。构建将生成一个 .jar，此即部署程序包。

1. 在 Eclipse 中新建一个 Maven 项目。
 - a. 从 File 菜单中，选择 New，然后选择 Project。
 - b. 在 New Project 窗口中，选择 Maven Project。
 - c. 在 New Maven Project 窗口中，选择 Create a simple project，保持其他默认选择不变。
 - d. 在 New Maven Project 的 Configure project 窗口中，键入以下 Artifact 信息：

- Group Id : doc-examples
 - Artifact Id : lambda-java-example
 - Version : 0.0.1-SNAPSHOT
 - Packaging : jar
 - Name : lambda-java-example
2. 在 `pom.xml` 文件中添加 `aws-lambda-java-core` 依赖项。

它提供了 `RequestHandler`、`RequestStreamHandler` 和 `Context` 接口的定义。这让您能够编译用于 AWS Lambda 的代码。

- a. 打开 `pom.xml` 文件的上下文菜单（右键单击），选择 Maven，然后选择 Add Dependency。
- b. 在 Add Dependency 窗口中，键入以下值：

Group Id : com.amazonaws

Artifact Id : aws-lambda-java-core

Version: (版本:) 1.2.0

Note

在按照本指南中的其他教程主题操作时，某些特定的教程可能会要求您添加更多的依赖项。确保根据需要添加这些依赖项。

3. 向项目添加 Java 类。

- a. 在项目的 `src/main/java` 子目录上打开上下文菜单（右键单击），选择 New，然后选择 Class。
- b. 在 New Java Class 窗口中，键入以下值：
 - Package (程序包) : `example`
 - Name (名称) : `Hello`

Note

在按照本指南中的其他教程主题操作时，某些特定的教程可能会建议使用其他程序包名称或类名称。

- c. 添加您的 Java 代码。如果您按照本指南中的其他教程主题操作，则添加所提供的代码。

4. 构建项目。

在 Package Explorer 中，打开该项目的上下文菜单（右键单击），选择 Run As，然后选择 Maven Build...。在 Edit Configuration (编辑配置) 窗口中的 Goals (目标) 框中键入 `package`。

Note

生成的 `.jar lambda-java-example-0.0.1-SNAPSHOT.jar` 不是可用作部署程序包的最终独立 `.jar`。在下一步中，添加 Apache maven-shade-plugin 以创建独立的 `.jar`。有关更多信息，请参阅 [Apache Maven Shade Plugin](#)。

5. 添加 maven-shade-plugin 插件并重新构建。

maven-shade-plugin 将接收 package 目标（生成客户代码 `.jar`）生成的项目 (`jar`) 并创建独立的 `.jar`，该 `.jar` 包含经过编译的客户代码和从 `pom.xml` 解析的依赖项。

- a. 打开 `pom.xml` 文件的上下文菜单（右键单击），选择 Maven，然后选择 Add Plugin。
- b. 在 Add Plugin 窗口中，键入以下值：

- Group Id : org.apache.maven.plugins
- Artifact Id : maven-shade-plugin 328

- Version: (版本:) 3.2.2
- c. 现在，重新构建。

这次，我们像以前一样创建 jar，然后使用 maven-shade-plugin 加入依赖项以生成独立的 .jar。

- i. 打开该项目的上下文菜单（右键单击），选择 Run As，然后选择 Maven build...。
- ii. 在 Edit Configuration 窗口中的 Goals 框中键入 **package shade:shade**。
- iii. 选择 Run。

您可在 /target 子目录中找到生成的独立 .jar（即部署程序包）。

打开 /target 子目录的上下文菜单（右键单击），依次选择 Show In、System Explorer，即可找到该 lambda-java-example-0.0.1-SNAPSHOT.jar。

使用 Go 构建 Lambda 函数

以下部分说明在使用 Go 编写 Lambda 函数代码时如何应用常见的编程模式和核心概念。

Go 运行时

名称	标识符	操作系统
Go 1.x	go1.x	Amazon Linux

AWS Lambda 提供以下适用于 Go 的库：

- [github.com/aws/aws-lambda-go/lambda](#)：适用于 Go 的 Lambda 编程模型的实现。AWS Lambda 使用此程序包调用您的[处理程序 \(p. 331\)](#)。
- [github.com/aws/aws-lambda-go/lambdacontext](#)：用于访问[上下文对象 \(p. 334\)](#)中的执行上下文信息的帮助程序。
- [github.com/aws/aws-lambda-go/events](#)：此库提供常见事件源集成的类型定义。

主题

- [Go 中的 AWS Lambda 部署程序包 \(p. 330\)](#)
- [Go 中的 AWS Lambda 函数处理程序 \(p. 331\)](#)
- [Go 中的 AWS Lambda 上下文对象 \(p. 334\)](#)
- [Go 中的 AWS Lambda 函数日志记录 \(p. 336\)](#)
- [Go 中的 AWS Lambda 函数错误 \(p. 339\)](#)
- [在 AWS Lambda 中检测 Go 代码 \(p. 339\)](#)
- [使用环境变量 \(p. 341\)](#)

Go 中的 AWS Lambda 部署程序包

要创建 Lambda 函数，首先需要创建 Lambda 函数部署程序包（包含代码（Go 可执行文件）和所有依赖项的 .zip 文件）。

在创建部署程序包后，您可直接上传该程序包或先将 .zip 文件上传到要在其中创建 Lambda 函数的 AWS 区域中的 Amazon S3 存储桶，然后指定使用控制台或 AWS CLI 创建 Lambda 函数时的存储桶名称和对象键名称。

使用 go get 下载适用于 Go 的 Lambda 库，并编译您的可执行文件。

```
~/my-function$ go get github.com/aws/aws-lambda-go/lambda
~/my-function$ GOOS=linux go build main.go
```

将 GOOS 设置为 linux 可确保编译的可执行文件与 [Go 运行时 \(p. 108\)](#)兼容（即使您在非 Linux 环境中编译它也是如此）。

通过将可执行文件打包为 ZIP 文件来创建部署包，并使用 AWS CLI 创建函数。处理程序参数必须与包含处理程序的可执行文件的名称匹配。

```
~/my-function$ zip function.zip main
~/my-function$ aws lambda create-function --function-name my-function --runtime go1.x \
```

```
--zip-file fileb://function.zip --handler main \
--role arn:aws:iam::123456789012:role/execution_role
```

在 Windows 上创建部署程序包

要使用 Windows 创建适用于 AWS Lambda 的 .zip，我们建议安装 build-lambda-zip 工具。

Note

如果您尚未完成此操作，则需要安装 [git](#)，然后将 git 可执行文件添加到您的 Windows %PATH% 环境变量。

要下载该工具，请运行以下命令：

```
go.exe get -u github.com/aws/aws-lambda-go/cmd/build-lambda-zip
```

使用您的 GOPATH 中的工具。如果您有 Go 的默认安装，则该工具通常在 %USERPROFILE%\Go\bin 中。否则，请导航到安装 Go 运行时的位置，然后执行以下操作：

在 cmd.exe 中，运行以下命令：

```
set GOOS=linux
go build -o main main.go
%USERPROFILE%\Go\bin\build-lambda-zip.exe -output main.zip main
```

在 Powershell 中，运行以下命令：

```
$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
```

Go 中的 AWS Lambda 函数处理程序

在 [Go](#) 中编写的 Lambda 函数被编写为 Go 可执行文件。在您的 Lambda 函数代码中，您需要包含 [github.com/aws/aws-lambda-go/lambda](#) 程序包，该程序包将实现适用于 Go 的 Lambda 编程模型。此外，您需要实现处理程序函数代码和 main() 函数。

```
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)
```

```
}
```

请注意以下几点：

- package main：在 Go 中，包含 `func main()` 的程序包必须始终名为 `main`。
- import：请使用此包含您的 Lambda 函数需要的库。在此实例中，它包括：
 - 上下文：[Go 中的 AWS Lambda 上下文对象 \(p. 334\)](#)。
 - `fmt`：用于格式化您的函数返回的值的 Go [格式化](#)对象。
 - `github.com/aws/aws-lambda-go/lambda`：如前所述，实现适用于 Go 的 Lambda 编程模型。
- `func HandleRequest(ctx context.Context, name MyEvent) (string, error)`：这是您的 Lambda 处理程序签名且包括将执行的代码。此外，包含的参数表示以下含义：
 - `ctx context.Context`：为您的 Lambda 函数调用提供运行时信息。`ctx` 是您声明的变量，用于利用通过 [Go 中的 AWS Lambda 上下文对象 \(p. 334\)](#) 提供的信息。
 - `name MyEvent`：变量名称为 `name` 的输入类型，其值将在 `return` 语句中返回。
 - `string, error`：返回两个值：成功时的字符串和标准[错误](#)信息。有关自定义错误处理的更多信息，请参阅[Go 中的 AWS Lambda 函数错误 \(p. 339\)](#)。
 - `return fmt.Sprintf("Hello %s!", name), nil`：只返回格式化“Hello”问候语和您在输入事件中提供的姓名。`nil` 表示没有错误，函数已成功执行。
- `func main()`：执行您的 Lambda 函数代码的入口点。该项为必填项。

通过在 `func main(){}{}` 代码括号之间添加 `lambda.Start(HandleRequest)`，您的 Lambda 函数将会执行。按照 Go 语言标准，开括号 `{}` 必须直接置于 `main` 函数签名末尾。

使用结构化类型的 Lambda 函数处理程序

在上述示例中，输入类型是简单的字符串。但是，您也可以将结构化事件传递到您的函数处理程序：

```
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)
}
```

然后，您的请求将如下所示：

```
# request
```

```
{  
    "What is your name?": "Jim",  
    "How old are you?": 33  
}
```

而响应将如下所示：

```
# response  
{  
    "Answer": "Jim is 33 years old!"  
}
```

若要导出，事件结构中的字段名称必须大写。有关来自 AWS 事件源的处理事件的更多信息，请参见 [aws-lambda-go/events](#)。

有效处理程序签名

在 Go 中构建 Lambda 函数处理程序时，您有多个选项，但您必须遵守以下规则：

- 处理程序必须为函数。
- 处理程序可能需要 0 到 2 个参数。如果有两个参数，则第一个参数必须实现 `context.Context`。
- 处理程序可能返回 0 到 2 个参数。如果有一个返回值，则它必须实现 `error`。如果有两个返回值，则第二个值必须实现 `error`。有关实现错误处理信息的更多信息，请参阅[Go 中的 AWS Lambda 函数错误 \(p. 339\)](#)。

下面列出了有效的处理程序签名。`TIn` 和 `TOut` 表示类型与 `encoding/json` 标准库兼容。有关更多信息，请参阅 [func Unmarshal](#)，以了解如何反序列化这些类型。

- `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)`

使用全局状态

您可以声明并修改独立于 Lambda 函数的处理程序代码的全局变量。此外，您的处理程序可能声明一个 `init` 函数，该函数在加载您的处理程序时执行。这在 AWS Lambda 中行为方式相同，正如在标准 Go 程序中一样。您的 Lambda 函数的单个实例将不会同时处理多个事件。

```
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.Println(myObjects)
    return invokeCount, nil
}

func main() {
    lambda.Start(LambdaHandler)
}
```

Go 中的 AWS Lambda 上下文对象

当 Lambda 运行您的函数时，它会将上下文对象传递到[处理程序 \(p. 331\)](#)。此对象提供的方法和属性包含有关调用、函数和执行环境的信息。

Lambda 上下文库提供以下全局变量、方法和属性。

全局变量

- `FunctionName` – Lambda 函数的名称。
- `FunctionVersion` – 函数的[版本 \(p. 63\)](#)。
- `MemoryLimitInMB` – 为函数分配的内存量。
- `LogGroupName` – 函数的日志组。
- `LogStreamName` – 函数实例的日志流。

上下文方法

- `Deadline` – 返回执行超时的日期 (Unix 时间形式，以毫秒为单位)。

上下文属性

- `InvokedFunctionArn` – 用于调用函数的 Amazon 资源名称 (ARN)。指示调用方是否已指定版本或别名。
- `AwsRequestId` – 调用请求的标识符。
- `Identity` – (移动应用程序) 有关授权请求的 Amazon Cognito 身份的信息。
- `ClientContext` – (移动应用程序) 由客户端应用程序向 Lambda 提供的客户端上下文。

访问调用上下文信息

Lambda 函数可以访问有关其环境和调用请求的元数据。这可以在[程序包上下文](#)出访问。如果您的处理程序将 `context.Context` 作为参数包含，则 Lambda 会将有关您的函数的信息插入上下文的 `Value` 属性。请注意，您需要导入 `lambdacontext` 库才能访问 `context.Context` 对象的内容。

```
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)
}
```

在上述示例中，`lc` 是用于使用 `context` 对象捕获的信息的变量，`log.Print(lc.Identity.CognitoIdentityPoolID)` 将输出该信息（在本例中为 `CognitoIdentityPoolID`）。

以下示例介绍了如何使用 `context` 对象来监控执行您的 Lambda 函数需要多长时间。这让您能够分析性能期望并相应地调整您的函数代码（如果需要）。

```
package main

import (
    "context"
    "log"
    "time"
    "github.com/aws/aws-lambda-go/lambda"
)

func LongRunningHandler(ctx context.Context) (string, error) {

    deadline, _ := ctx.Deadline()
    deadline = deadline.Add(-100 * time.Millisecond)
    timeoutChannel := time.After(time.Until(deadline))

    for {

        select {

        case <- timeoutChannel:
            return "Finished before timing out.", nil

        default:
            log.Print("hello!")
            time.Sleep(50 * time.Millisecond)
        }
    }
}

func main() {
    lambda.Start(LongRunningHandler)
}
```

}

Go 中的 AWS Lambda 函数日志记录

您的 Lambda 函数带有一个 CloudWatch Logs 日志组，其中包含您的函数的每个实例的日志流。运行时会将每个调用的详细信息发送到该日志流，然后中继日志和来自您的函数代码的其他输出。

您可以使用 `fmt` 程序包中的方法或写入到 `stdout` 或 `stderr` 的任何日志记录库，从函数代码输出日志。以下示例使用 [日志包](#)。

Example [main.go](#) – 日志记录

```
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 日志格式

```
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:
100 ms Memory Size: 128 MB Max Memory Used: 54 MB Init Duration: 203.69 ms
XRAY TraceId: 1-5e7d7595-212fxmpl9ee07c4884191322 SegmentId: 42ffxmpl0645f474 Sampled: true
```

Go 运行时记录每次调用的 `START`、`END` 和 `REPORT` 行。报告行提供了以下详细信息。

报告日志

- `RequestId` – 调用的唯一请求 ID。
- `Duration (持续时间)` – 函数的处理程序方法处理事件所花费的时间。

- Billed Duration (计费持续时间) – 针对调用计费的时间量。
- Memory Size (内存大小) – 分配给函数的内存量。
- Max Memory Used (最大内存使用量) – 函数使用的内存量。
- Init Duration (初始持续时间) – 对于提供的第一个请求，为运行时在处理程序方法外部加载函数和运行代码所花费的时间。
- XRAY Traceld – 对于跟踪的请求，为 [AWS X-Ray 跟踪编码 \(p. 371\)](#)。
- SegmentId – 对于跟踪的请求，为 X-Ray 分段 ID。
- Sampled (采样) – 对于跟踪的请求，为采样结果。

您可以在 Lambda 控制台中、在 CloudWatch Logs 控制台中或从命令行查看日志。

小节目录

- [在 AWS 管理控制台中查看日志 \(p. 337\)](#)
- [使用 AWS CLI \(p. 337\)](#)
- [删除日志 \(p. 339\)](#)

在 AWS 管理控制台中查看日志

当您在函数配置页上测试函数时，Lambda 控制台会显示日志输出。要查看所有调用的日志，请使用 CloudWatch Logs 控制台。

查看 Lambda 函数的日志

1. 打开 [CloudWatch 控制台的日志页面](#)。
2. 选择您的函数 (`/aws/lambda/function-name`) 的日志组。
3. 选择列表中的第一个流。

每个日志流对应一个[函数实例 \(p. 109\)](#)。当您更新函数以及创建更多实例以处理多个并发调用时，会显示新的流。要找到特定调用的日志，您可以使用 X-Ray 分析您的函数并在跟踪中记录有关请求和日志流的详细信息。如需将日志和跟踪与 X-Ray 相关联的示例应用程序，请参阅[AWS Lambda 错误处理器示例应用程序 \(p. 128\)](#)。

使用 AWS CLI

要从命令行获取调用的日志，请使用 `--log-type` 选项。响应包含一个 `LogResult` 字段，该字段包含来自调用的多达 4 KB 的 base64 编码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

您可以使用 `base64` 实用程序来解码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
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: 100 ms          Memory Size: 128 MB      Max Memory Used: 73 MB
```

base64 实用工具在 Linux、macOS 和 [Ubuntu on Windows](#) 上可用。对于 macOS，命令为 `base64 -D`。

要从命令行获取完整的日志事件，您可以在函数输出中包含日志流名称，如上例中所示。以下示例脚本调用名为 `my-function` 的函数并下载最后 5 个日志事件。

Example get-logs.sh 脚本

此示例要求 `my-function` 返回日志流 ID。

```
#!/bin/bash
aws lambda invoke --function-name my-function --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
```

此脚本使用 `sed` 从输出文件中删除引号，并休眠 15 秒以等待日志可用。输出包括来自 Lambda 的响应，以及来自 `get-log-events` 命令的输出。

```
$ ./get-logs.sh
{
    "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 ...",
            "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: 100 ms \tMemory Size: 128 MB\tMax Memory Used: 75 MB\t\n",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
```

}

删除日志

在删除函数时，不会自动删除日志组。为避免存储无限的日志，请删除日志组或者[配置保留期](#)，在该期限后会自动删除日志。

Go 中的 AWS Lambda 函数错误

您可以创建自定义错误处理机制，直接从您的 Lambda 函数引发异常，并直接进行处理。

以下代码示例演示了如何执行此操作。请注意，Go 中的自定义错误必须导入 errors 模块。

```
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)
}
```

将返回以下内容：

```
{
    "errorMessage": "something went wrong!",
    "errorType": "errorString"
}
```

在 AWS Lambda 中检测 Go 代码

您可以将[适用于 Go 的 X-Ray 开发工具包](#)与您的 Lambda 函数搭配使用。如果您的处理程序包含[Go 中的 AWS Lambda 上下文对象 \(p. 334\)](#)作为其第一个参数，则该对象可传递给 X-Ray 开发工具包。Lambda 通过此上下文传递开发工具包可用于将子分段附加到 Lambda 调用服务分段的值。使用软件开发工具包创建的子分段将显示为您的 Lambda 跟踪的一部分。

安装适用于 Go 的 X-Ray 开发工具包

使用以下命令可安装适用于 Go 的 X-Ray 开发工具包。(该软件开发工具包的非测试依赖项将包含在内)。

```
go get -u github.com/aws/aws-xray-sdk-go/...
```

如果您要包含测试依赖项，请使用以下命令：

```
go get -u -t github.com/aws/aws-xray-sdk-go/...
```

您也可以使用 [Glide](#) 管理依赖项。

```
glide install
```

配置适用于 Go 的 X-Ray 开发工具包

以下代码示例演示了如何在您的 Lambda 函数中配置适用于 Go 的 X-Ray 开发工具包：

```
import (
    "github.com/aws/aws-xray-sdk-go/xray"
)
func myHandlerFunction(ctx context.Context, sample string) {
    xray.Configure(xray.Config{
        LogLevel:      "info",           // default
        ServiceVersion: "1.2.3",
    })
    ... //remaining handler code
}
```

创建子分段

以下代码示例演示了如何启动子分段：

```
// Start a subsegment
ctx, subSeg := xray.BeginSubsegment(ctx, "subsegment-name")
// ...
// Add metadata or annotation here if necessary
// ...
subSeg.Close(nil)
```

Capture

以下代码演示了如何跟踪和捕获关键代码路径：

```
func criticalSection(ctx context.Context) {
    // This example traces a critical code path using a custom subsegment
    xray.Capture(ctx, "MyService.criticalSection", func(ctx1 context.Context) error {
        var err error

        section.Lock()
        result := someLockedResource.Go()
        section.Unlock()

        xray.AddMetadata(ctx1, "ResourceResult", result)
    })
}
```

跟踪 HTTP 请求

如果您想跟踪 HTTP 客户端，也可以使用 `xray.Client()` 方法，如下所示：

```
func myFunction (ctx context.Context) ([]byte, error) {
    resp, err := ctxhttp.Get(ctx, xray.Client(nil), "https://aws.amazon.com")
    if err != nil {
```

```
    return nil, err
}
return ioutil.ReadAll(resp.Body), nil
}
```

使用环境变量

要在 Go 中访问[环境变量 \(p. 49\)](#)，请使用 `Getenv` 函数。

下面介绍了如何完成此步骤。请注意，函数将导入 `fmt` 程序包，以格式化打印的结果，还将导入 `os` 程序包，后者是一个独立于平台的系统界面，可让您访问环境变量。

```
package main

import (
    "fmt"
    "os"
    "github.com/aws/aws-lambda-go/lambda"
)

func main() {
    fmt.Printf("%s is %s. years old\n", os.Getenv("NAME"), os.Getenv("AGE"))
}
```

有关 Lambda 运行时设置的环境变量的列表，请参阅[运行时环境变量 \(p. 51\)](#)。

使用 C# 构建 Lambda 函数

以下部分说明在使用 C# 编写 Lambda 函数代码时如何应用常见的编程模式和核心概念。

AWS Lambda 提供以下适用于 C# 函数的库：

- Amazon.Lambda.Core – 该库提供一个静态 Lambda 记录器、若干序列化接口和一个 context 对象。Context 对象 ([C# 中的 AWS Lambda 上下文对象 \(p. 352\)](#)) 提供有关您的 Lambda 函数的运行时信息。
- Amazon.Lambda.Serialization.Json – 这是在 Amazon.Lambda.Core 中实施序列化接口的实例。
- Amazon.Lambda.Logging.AspNetCore – 这是用于从 ASP.NET 记录日志的库。
- 适用于几项 AWS 服务的事件对象 (POCO) , 其中包括：
 - Amazon.Lambda.APIGatewayEvents
 - Amazon.Lambda.CognitoEvents
 - Amazon.Lambda.ConfigEvents
 - Amazon.Lambda.DynamoDBEvents
 - Amazon.Lambda.KinesisEvents
 - Amazon.Lambda.S3Events
 - Amazon.Lambda.SQSEvents
 - Amazon.Lambda.SNSEvents

这些程序包都可以在 [Nuget 程序包](#) 中找到。

.NET 运行时

名称	标识符	操作系统	
.NET Core 3.1	dotnetcore3.1	Amazon Linux 2	
.NET Core 2.1	dotnetcore2.1	Amazon Linux	

主题

- [C# 中的 AWS Lambda 部署程序包 \(p. 342\)](#)
- [C# 中的 AWS Lambda 函数处理程序 \(p. 347\)](#)
- [C# 中的 AWS Lambda 上下文对象 \(p. 352\)](#)
- [C# 中的 AWS Lambda 函数日志记录 \(p. 352\)](#)
- [C# 中的 AWS Lambda 函数错误 \(p. 355\)](#)

C# 中的 AWS Lambda 部署程序包

.NET Core Lambda 部署程序包是一个 zip 文件，包含您的函数的已编译程序集以及其所有程序集依赖项。该程序包还包含一个 `proj.deps.json` 文件。这将向 .NET Core 运行时告知您的所有函数的依赖项和

`proj.runtimeconfig.json` 文件，后者用于配置 .NET Core 运行时。.NET CLI 的 publish 命令可以创建一个包含所有这些文件的文件夹，但默认情况下 `proj.runtimeconfig.json` 将不会包含在内，因为 Lambda 项目通常被配置为类库。要在 publish 流程中强制写入 `proj.runtimeconfig.json`，请传入命令行参数：`/p:GenerateRuntimeConfigurationFiles=true` to the publish command。

虽然能够使用 dotnet publish 命令创建部署程序包，但我们建议您使用 [AWS Toolkit for Visual Studio \(p. 346\)](#) 或 [.NET Core CLI \(p. 343\)](#) 创建部署程序包。这些工具专门针对 Lambda 进行了优化，以确保 `lambda-project.runtimeconfig.json` 文件存在并优化程序包，包括删除任何并非基于 Linux 的依赖项。

主题

- [.NET Core CLI \(p. 343\)](#)
- [AWS Toolkit for Visual Studio \(p. 346\)](#)

.NET Core CLI

借助 .NET Core CLI，您可以通过跨平台方式创建基于 .NET 的 Lambda 应用程序。本部分假定您已安装 .NET Core CLI。如果您尚未安装，请单击[此处](#)安装。

在 .NET CLI 中，您可以使用 new 命令从命令行创建 .NET 项目。如果要在 Visual Studio 之外创建项目，这种做法非常有用。要查看可用项目类型的列表，请打开命令行并导航到您安装 .NET Core 运行时的位置，然后运行以下命令：

Templates	Short Name	Language	Tags
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			
...			
Examples:			
dotnet new mvc --auth Individual			
dotnet new viewstart			
dotnet new --help			

AWS Lambda 通过 [Amazon.Lambda.Templates](#) NuGet 程序包提供其他模板。要安装此程序包，请运行以下命令：

```
dotnet new -i Amazon.Lambda.Templates
```

一旦安装完成，Lambda 模板将作为 dotnet new 的一部分显示。要检查有关模板的详细信息，请使用 help 选项。

```
dotnet new lambda.EmptyFunction --help
```

`lambda.EmptyFunction` 模板支持以下选项。

- `--name` – 函数的名称。

- **--profile** – 适用于 .NET 的 AWS 开发工具包凭据文件中的配置文件的名称。
- **--region** – 要在其中创建函数的 AWS 区域。

这些选项将保存到名为 aws-lambda-tools-defaults.json 的文件中。

使用 lambda.EmptyFunction 模板创建一个函数项目。

```
dotnet new lambda.EmptyFunction --name MyFunction
```

在 src/myfunction 目录下，检查以下文件：

- **aws-lambda-tools-defaults.json**：这是您部署 Lambda 函数时指定命令行选项的位置。例如：

```
"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**：您的 Lambda 处理程序函数代码。它是一个 C# 模板，该模板包含默认 Amazon.Lambda.Core 库和默认 LambdaSerializer 属性。有关序列化要求和选项的更多信息，请参阅[序列化 Lambda 函数 \(p. 350\)](#)。它还包含一个示例函数，您可以编辑该函数以应用您的 Lambda 函数代码。

```
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**：列出构成您的应用程序的文件和程序集的 [MSBuild](#) 文件。

```
<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：使用此文件记录您的 Lambda 函数。

在 myfunction/test directory, examine the following files:

- myFunction.Tests.csproj 下：如上所述，这是一个 [MSBuild](#) 文件，其中列出了构成您的测试项目的文件和程序集。另请注意，它包含 Amazon.Lambda.Core 库，允许您无缝集成测试您的函数所需的任何 Lambda 模板。

```
<Project Sdk="Microsoft.NET.Sdk">
  ...
  <PackageReference Include="Amazon.Lambda.Core" Version="1.0.0" />
  ...
```

- FunctionTest.cs：src 目录中包含的相同 C# 代码模板文件。编辑此文件，以镜像您的函数的生产代码并对其进行测试，然后将您的 Lambda 函数上传到生产环境。

```
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);
        }
    }
}
```

您的函数一通过其测试，您就可以使用 Amazon.Lambda.Tools .NET Core Global Tool 对其进行构建和部署。要安装 .NET Core Global Tool，请运行以下命令。

```
dotnet tool install -g Amazon.Lambda.Tools
```

如果您已安装该工具，请确保您使用的是包含以下命令的最新版本。

```
dotnet tool update -g Amazon.Lambda.Tools
```

有关 Amazon.Lambda.Tools .NET Core Global 的更多信息，请参阅其 [GitHub 存储库](#)。

安装 Amazon.Lambda.Tools 之后，您可以使用以下命令部署函数：

```
dotnet lambda deploy-function MyFunction --function-role role
```

部署完成后，您可以使用以下命令在生产环境中对其进行重新测试，并将不同的值传递到您的 Lambda 函数处理程序：

```
dotnet lambda invoke-function MyFunction --payload "Just Checking If Everything is OK"
```

假设一切步骤均已成功，您应该会看到以下内容：

```
dotnet lambda invoke-function MyFunction --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: 100 ms      Memory Size:
256 MB      Max Memory Used: 12 MB
```

AWS Toolkit for Visual Studio

您可以使用 AWS Toolkit for Visual Studio 的 Lambda 插件构建基于 .NET 的 Lambda 应用程序。该工具包可作为 [Visual Studio 扩展名](#) 提供。

1. 启动 Microsoft Visual Studio 并选择新建项目。
 - a. 从 File 菜单中，选择 New，然后选择 Project。
 - b. 在新建项目窗口中，选择 AWS Lambda 项目 (.NET Core)，然后选择确定。
 - c. 在选择蓝图窗口中，系统会显示从示例应用程序列表中进行选择的选项，而这些示例应用程序将为您提供相应示例代码，方便您开始着手创建基于 .NET 的 Lambda 应用程序。
 - d. 要从头创建 Lambda 应用程序，请选择 Empty Function (空白函数)，然后选择 Finish (完成)。
2. 检查 aws-lambda-tools-defaults.json 文件，该文件作为项目的一部分创建。您可以在此文件中设置选项，默认情况下由 Lambda 工具读取这些选项。在 Visual Studio 中创建的项目模板使用默认值设置多个此类字段。请注意以下字段：
 - profile – [适用于 .NET 的 AWS 开发工具包凭据文件](#) 中的配置文件的名称。
 - function-handler – 这是指定 function handler 的位置，也就是您无需在向导中设置它的原因。但是，每当您在函数代码中重命名 **Assembly**、**Namespace**、**Class** 或 **Function** 时，您都需要在 aws-lambda-tools-defaults.json file 中更新相应字段。

```
{
  "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. 打开 Function.cs 文件。系统会为您提供一个实施 Lambda 函数处理程序代码的模板。

```

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. 如果您已编写表示您 Lambda 函数的代码，则可以通过以下方式上传该代码：右键单击您的应用程序中的 Project (项目) 节点，然后选择 Publish to AWS Lambda (发布至 AWS Lambda)。
5. 在 Upload Lambda Function 窗口中，键入函数的名称或选择之前发布的函数以重新发布。然后选择下一个
6. 在 Advanced Function Details (高级函数详细信息) 窗口中，配置以下选项：
 - 角色名称 (必需) – AWS Lambda 执行函数时使用的 IAM 角色 (p. 30)。
 - 环境 – Lambda 在执行环境中设置的键值对。使用环境变量 (p. 49) 在代码之外扩展函数的配置。
 - 内存 – 执行期间函数可用的内存量。请选择介于 128 MB 与 3,008 MB (p. 27) 之间的值，以 64 MB 为增量。
 - 超时 – Lambda 在停止函数前允许其运行的时间。默认值为 3 秒。允许的最大值为 900 秒。
 - VPC – 如果您的函数需要通过网络访问无法在 Internet 上获得的资源，请将其配置为连接到 VPC (p. 72)。
 - DLQ – 如果您的函数以异步方式调用，请选择队列或主题 (p. 88) 来接收失败的调用。
 - 启用活动跟踪 – 对传入请求进行采样并使用 AWS X-Ray 跟踪采样的请求 (p. 371)。
7. 选择下一个，然后选择上载，即可部署您的应用程序。

有关更多信息，请参阅[使用 .NET Core CLI 部署 AWS Lambda 项目](#)。

C# 中的 AWS Lambda 函数处理程序

创建 Lambda 函数时，需指定一个处理程序以供 AWS Lambda 服务在代表您执行函数时调用。

将 Lambda 函数处理程序定义为某个类中的实例或静态方法。如果您希望访问 Lambda context 对象，可通过定义类型为 ILambdaContext 的方法参数来实现，该参数是一个您可以用来访问有关当前执行的信息（例如，当前函数的名称、内存限制、剩余执行时间和日志记录）的接口。

```

returnType handler-name(inputType input, ILambdaContext context) {
    ...
}

```

在该语法中，需要注意以下方面：

- **inputType** – 第一个处理程序参数是处理程序的输入，它可以是事件数据（由事件源发布）或您提供的自定义输入（如字符串或任意自定义数据对象）。

- **returnType** – 如果打算同步调用 Lambda 函数（使用 RequestResponse 调用类型），则可以使用任何支持的数据类型返回函数输出。例如，如果使用 Lambda 函数作为移动应用程序后端并同步调用它，则输出数据类型会序列化为 JSON。
- 如果打算异步调用 Lambda 函数（使用 Event 调用类型），则 returnType 应为 void。例如，将 AWS Lambda 搭配 Amazon S3 或 Amazon SNS 之类的事件源使用时，这些事件源会使用 Event 调用类型调用 Lambda 函数。
- **ILambdaContext context** – 处理程序签名中的第二个参数是可选的。它提供对[上下文对象](#) (p. 352) 的访问权限，该对象包含有关函数和请求的信息。

处理流

仅 System.IO.Stream 类型在默认情况下可作为输入参数受支持。

例如，考虑以下 C# 示例代码。

```
using System.IO;

namespace Example
{
    public class Hello
    {
        public Stream MyHandler(Stream stream)
        {
            //function logic
        }
    }
}
```

在此 C# 示例代码中，第一个处理程序参数是处理程序 (MyHandler) 的输入，它可以是事件数据（由 Amazon S3 之类的事件源发布），也可以是您提供的自定义输入（如本示例中的 Stream）或任何自定义数据对象。输出的类型为 Stream。

处理标准数据类型

其他所有类型（如下所列）均需要您指定串行器。

- Primitive .NET 类型（例如 string 或 int）。
- 集合和映射 - IList、IEnumerable、IList<T>、Array、IDictionary、IDictionary< TKey 和 TValue >
- POCO 类型（无格式的旧 CLR 对象）
- 预定义的 AWS 事件类型
- Lambda 将忽略异步调用的返回类型。在这种情况下，可能会将返回类型设置为 void。
- 如果您使用的是 .NET 异步编程，则返回类型可以是 Task 和 Task<T>，并且可使用 async 和 await 关键字。有关更多信息，请参阅[在使用 C# 编写的 AWS Lambda 函数中应用 Async \(p. 351\)](#)。

除非函数输入和输出参数的类型为 System.IO.Stream，否则您需要对这些参数执行序列化。AWS Lambda 提供了可在应用程序的程序集或方法级别应用的默认串行器，或者您也可以通过实施由 Amazon.Lambda.Core 库提供的 ILambdaSerializer 接口定义自己的串行器。有关更多信息，请参阅[C# 中的 AWS Lambda 部署程序包 \(p. 342\)](#)。

要向某个方法添加默认的串行器属性，请先添加对 project.json 文件中 Amazon.Lambda.Serialization.Json 的依赖关系。

```
{
```

```

    "version": "1.0.0-*",
    "dependencies": {
        "Microsoft.NETCore.App": {
            "type": "platform",
            "version": "1.0.1"
        },
        "Amazon.Lambda.Serialization.Json": "1.3.0"
    },
    "frameworks": {
        "netcoreapp1.0": {
            "imports": "dnxcore50"
        }
    }
}

```

以下示例说明了您可以分别针对自己选择的各种方法灵活地指定默认 Json.NET 串行器：

```

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);
    }
}

```

处理程序签名

创建 Lambda 函数时，您必须提供一个处理程序字符串，告知 AWS Lambda 在何处查找要调用的代码。在 C# 中，格式如下：

ASSEMBLY::TYPE::METHOD，其中：

- **ASSEMBLY** 是您的应用程序的 .NET 程序集文件的名称。使用 .NET Core CLI 构建应用程序时，如果未使用 project.json 中的 buildOptions.outputName 设置来设置程序集名称，则 **ASSEMBLY** 名称将为包含 project.json 文件的文件夹的名称。有关更多信息，请参阅 [.NET Core CLI \(p. 343\)](#)。在这种情况下，假设文件夹名称为 HelloWorldApp。
- **TYPE** 是包含 **Namespace** 和 **ClassName** 的处理程序类型的全名。在本例中为 Example.Hello。
- **METHOD** 是函数处理程序的名称，在本例中为 MyHandler。

签名最终将是以下格式：**Assembly::Namespace.ClassName::MethodName**

请考虑以下示例：

```

using System.IO;

namespace Example
{
    public class Hello
    {
        public Stream MyHandler(Stream stream)
        {
            //function logic
        }
    }
}

```

```
}
```

处理程序字符串为 : HelloWorldApp::Example.Hello::MyHandler

Important

如果在处理程序字符串中指定的方法重载，您必须提供 Lambda 应调用的方法的准确签名。如果该解决方法要求在多个（重载）签名中进行选择，AWS Lambda 将拒绝其他有效签名。

序列化 Lambda 函数

对于任何使用 Stream 对象以外的输入或输出类型的 Lambda 函数，您都需要向应用程序中添加一个序列化库。您可以通过下列方式来执行此操作：

- 使用 Amazon.Lambda.Serialization.Json NuGet 程序包。该库使用 JSON.NET 来处理序列化。
- 通过实施 ILambdaSerializer 接口（作为 Amazon.Lambda.Core 库的一部分提供）创建您自己的序列化库。该接口定义了两种方法：
 - T Deserialize<T>(Stream requestStream);

通过实施此方法，您可以将请求负载从 Invoke API 反序列化至传递到 Lambda 函数处理程序的对象中。

- T Serialize<T>(T response, Stream responseStream);。

通过实施此方法，您可以将从 Lambda 函数处理程序中返回的结果序列化到 Invoke API 返回的响应负载中。

您可以使用任意一个串行器，方法是将其作为依赖项添加到您的 MyProject.csproj 文件中。

```
...
<ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="1.0.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="1.3.0" />
</ItemGroup>
```

然后将其添加到您的 AssemblyInfo.cs 文件中。例如，如果您使用的是默认 Json.NET 串行器，则需要添加以下项：

```
[assembly:LambdaSerializer(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]
```

Note

您可以在方法级别定义一个自定义序列化属性，用于覆盖在程序集级别指定的默认串行器。有关更多信息，请参阅 [处理标准数据类型 \(p. 348\)](#)。

Lambda 函数处理程序限制

请注意，处理程序签名存在一些限制。

- 处理程序签名不能是 unsafe 的，且不得使用指针类型，但在处理程序方法及其依赖关系中可以使用 unsafe 上下文。有关更多信息，请参阅 [不安全 \(C# 参考\)](#)。
- 处理程序签名不得使用 params 关键字传递可变数量的参数，也不得将用于支持可变数量参数的 ArgIterator 用作输入或返回参数。

- 处理程序不能是泛型方法（例如 `IList<T> Sort<T>(IList<T> input)`）。
- 不支持使用签名 `async void` 的 `Async` 处理程序。

在使用 C# 编写的 AWS Lambda 函数中应用 Async

如果您知道自己的 Lambda 函数需要长时间的运行过程，例如上传大型文件到 Amazon S3 或者从 DynamoDB 中读取大量记录，则您可以利用 `async/await` 模式。当您使用此签名时，Lambda 会同步执行函数并等待该函数返回响应或执行超时 (p. 47)。

```
public async Task<Response> ProcessS3ImageResizeAsync(SimpleS3Event input)
{
    var response = await client.DoAsyncWork(input);
    return response;
}
```

使用此模式时，您必须谨记以下几项注意事项：

- AWS Lambda 不支持 `async void` 方法。
- 如果您创建了一个 `async` Lambda 函数，但未实施 `await` 运算符，.NET 将发出一个编译器，提醒您注意意外行为。例如，有些 `async` 操作会执行，而有些不会。或者，有些 `async` 操作不会在函数执行完成时结束。

```
public async Task ProcessS3ImageResizeAsync(SimpleS3Event event) // Compiler warning
{
    client.DoAsyncWork(input);
}
```

- 您的 Lambda 函数可包含多个可并行调用的 `async` 调用。您可使用 `Task.WhenAll` 和 `Task.WhenAny` 方法来处理多项任务。要使用 `Task.WhenAll` 方法，您需要将一个操作列表作为阵列传递至该方法。请注意，在以下示例中，如果您忘记在该阵列中包含任何操作，则调用可能会在操作结束之前返回。

```
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);
}
```

要使用 `Task.WhenAny` 方法，您同样需要将一个操作列表作为阵列传递至该方法。该调用将在第一个操作结束时返回，即使其他操作仍在运行也是如此。

```
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);
}
```

C# 中的 AWS Lambda 上下文对象

当 Lambda 运行您的函数时，它会将上下文对象传递到[处理程序 \(p. 347\)](#)。此对象提供了属性与有关调用、函数和执行环境的信息。

上下文属性

- `FunctionName` – Lambda 函数的名称。
- `FunctionVersion` – 函数的[版本 \(p. 63\)](#)。
- `InvokedFunctionArn` – 用于调用函数的 Amazon 资源名称 (ARN)。指示调用方是否已指定版本或别名。
- `MemoryLimitInMB` – 为函数分配的内存量。
- `AwsRequestId` – 调用请求的标识符。
- `LogGroupName` – 函数的日志组。
- `LogStreamName` – 函数实例的日志流。
- `RemainingTime (TimeSpan)` – 在执行超时以前剩余的毫秒数。
- `Identity` – (移动应用程序) 有关授权请求的 Amazon Cognito 身份的信息。
- `ClientContext` – (移动应用程序) 由客户端应用程序向 Lambda 提供的客户端上下文。
- `Logger` - 函数的[记录器对象 \(p. 352\)](#)。

下面的 C# 代码片段显示了一个打印部分上下文信息的简便处理程序函数。

```
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);
}
```

C# 中的 AWS Lambda 函数日志记录

您的 Lambda 函数带有一个 CloudWatch Logs 日志组，其中包含您的函数的每个实例的日志流。运行时会将每个调用的详细信息发送到该日志流，然后中继日志和来自您的函数代码的其他输出。

要从函数代码输出日志，您可以使用[控制台类](#)的方法或使用写入到 `stdout` 或 `stderr` 的任何日志记录库。以下示例使用来自 [Amazon.Lambda.Core \(p. 342\)](#) 库的 `LambdaLogger` 类。

Example [src/blank-csharp/Function.cs](#) – 日志记录

```
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 日志格式

```

START RequestId: d1cf0ccb-xmpl-46e6-950d-04c96c9b1c5d Version: $LATEST
ENVIRONMENT VARIABLES:
{
    "AWS_EXECUTION_ENV": "AWS_Lambda_dotnetcore2.1",
    "AWS_LAMBDA_FUNCTION_MEMORY_SIZE": "256",
    "AWS_LAMBDA_LOG_GROUP_NAME": "/aws/lambda/blank-csharp-function-WU56XMPLV2XA",
    "AWS_LAMBDA_FUNCTION_VERSION": "$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

```

.NET 运行时记录每次调用的 START、END 和 REPORT 行。报告行提供了以下详细信息。

报告日志

- RequestId – 调用的唯一请求 ID。
- Duration (持续时间) – 函数的处理程序方法处理事件所花费的时间。
- Billed Duration (计费持续时间) – 针对调用计费的时间量。
- Memory Size (内存大小) – 分配给函数的内存量。
- Max Memory Used (最大内存使用量) – 函数使用的内存量。
- Init Duration (初始持续时间) – 对于提供的第一个请求，为运行时在处理程序方法外部加载函数和运行代码所花费的时间。
- XRAY Traceld – 对于跟踪的请求，为 AWS X-Ray 跟踪编码 (p. 371)。
- SegmentId – 对于跟踪的请求，为 X-Ray 分段 ID。
- Sampled (采样) – 对于跟踪的请求，为采样结果。

您可以在 Lambda 控制台中、在 CloudWatch Logs 控制台中或从命令行查看日志。

小节目录

- 在 AWS 管理控制台中查看日志 (p. 354)
- 使用 AWS CLI (p. 354)
- 删除日志 (p. 355)

在 AWS 管理控制台中查看日志

当您在函数配置页上测试函数时，Lambda 控制台会显示日志输出。要查看所有调用的日志，请使用 CloudWatch Logs 控制台。

查看 Lambda 函数的日志

1. 打开 [CloudWatch 控制台的日志页面](#)。
2. 选择您的函数 (`/aws/lambda/function-name`) 的日志组。
3. 选择列表中的第一个流。

每个日志流对应一个[函数实例 \(p. 109\)](#)。当您更新函数以及创建更多实例以处理多个并发调用时，会显示新的流。要找到特定调用的日志，您可以使用 X-Ray 分析您的函数并在跟踪中记录有关请求和日志流的详细信息。如需将日志和跟踪与 X-Ray 相关联的示例应用程序，请参阅[AWS Lambda 错误处理器示例应用程序 \(p. 128\)](#)。

使用 AWS CLI

要从命令行获取调用的日志，请使用 `--log-type` 选项。响应包含一个 `LogResult` 字段，该字段包含来自调用的多达 4 KB 的 base64 编码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOiaA4N2QwNDRiOC1mMTU0LTEzTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

您可以使用 `base64` 实用程序来解码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
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: 100 ms      Memory Size: 128 MB      Max Memory Used: 73 MB
```

`base64` 实用工具在 Linux、macOS 和 [Ubuntu on Windows](#) 上可用。对于 macOS，命令为 `base64 -D`。

要从命令行获取完整的日志事件，您可以在函数输出中包含日志流名称，如上例中所示。以下示例脚本调用名为 `my-function` 的函数并下载最后 5 个日志事件。

Example get-logs.sh 脚本

此示例要求 `my-function` 返回日志流 ID。

```
#!/bin/bash
```

```
aws lambda invoke --function-name my-function --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
```

此脚本使用 sed 从输出文件中删除引号，并休眠 15 秒以等待日志可用。输出包括来自 Lambda 的响应，以及来自 get-log-events 命令的输出。

```
$ ./get-logs.sh
{
    "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: 100 ms\tMemory Size: 128 MB\tMax Memory Used: 75 MB\t",
            "ingestionTime": 1559763018353
        }
    ],
    "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
    "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

删除日志

在删除函数时，不会自动删除日志组。为避免存储无限的日志，请删除日志组或者[配置保留期](#)，在该期限后会自动删除日志。

C# 中的 AWS Lambda 函数错误

当 Lambda 函数出现异常时，Lambda 将向您报告异常信息。异常可能发生在两个不同位置：

- 初始化 (Lambda 加载您的代码，验证处理程序字符串并为非静态类创建实例)。

- Lambda 函数调用。

序列化的异常信息将作为已建模的 JSON 对象的负载返回并被输出到 CloudWatch 日志中。

在初始化阶段，无效的处理程序字符串、违规的类型或方法（请参阅 [Lambda 函数处理程序限制 \(p. 350\)](#)），或者其他任何验证方法（例如忘记设置串行器属性以及将 POCO 作为输入或输出类型）都可能引发异常。这些异常的类型为 `LambdaException`。例如：

```
{  
    "errorType": "LambdaException",  
    "errorMessage": "Invalid lambda function handler: 'http://this.is.not.a.valid.handler/'.  
    The valid format is 'ASSEMBLY::TYPE::METHOD'."  
}
```

如果您的构造函数引发异常，则该错误类型也是 `LambdaException`，但构造过程中引发的异常将在本身即为已建模异常对象的 `cause` 属性中提供。

```
{  
    "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()"  
            ]  
        }  
    }  
}
```

如示例所示，内部异常将始终保留（作为 `cause` 属性），并且可进行深层嵌套。

调用期间也会发生异常。在这种情况下，将保留异常类型并将其作为负载直接返回到 CloudWatch 日志中。例如：

```
{  
    "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": {  
    "errorCode": "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": {  
        "errorCode": "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.EndGetResponse(IAsyncResult asyncResult)"  
,  
        "cause": {  
            "errorCode": "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)"  
            ]  
        }  
    }  
}  
}
```

根据调用类型传达错误信息的方法：

- `RequestResponse` 调用类型（即同步执行）：在这种情况下，您会收到错误消息。

例如，使用 Lambda 控制台调用 Lambda 函数时，调用类型始终为 RequestResponse，控制台将在其 Execution result 部分中显示 AWS Lambda 返回的错误信息。

- Event 调用类型（即异步执行）：在这种情况下，AWS Lambda 不返回任何信息。相反，它将错误信息记录到 CloudWatch Logs 和 CloudWatch 指标中。

AWS Lambda 可能会重试失败的 Lambda 函数，具体视事件源而定。有关更多信息，请参阅[AWS Lambda 中的错误处理和自动重试 \(p. 98\)](#)。

使用 PowerShell 构建 Lambda 函数

以下部分说明在使用 PowerShell 编写 Lambda 函数代码时如何应用常见的编程模式和核心概念。

.NET 运行时

名称	标识符	操作系统	
.NET Core 3.1	dotnetcore3.1	Amazon Linux 2	
.NET Core 2.1	dotnetcore2.1	Amazon Linux	

请注意，PowerShell 中的 Lambda 函数要求 PowerShell Core 6.0。不支持 Windows PowerShell。

在您开始之前，您必须先设置 PowerShell 开发环境。有关如何执行此操作的说明，请参阅[设置 PowerShell 开发环境 \(p. 359\)](#)。

要了解如何使用 AWSLambdaPSCore 模块从模板中下载示例 PowerShell 项目、创建 PowerShell 部署程序包，以及在 AWS 云中部署 PowerShell 函数，请参阅[PowerShell 中的 AWS Lambda 部署程序包 \(p. 360\)](#)。

主题

- [设置 PowerShell 开发环境 \(p. 359\)](#)
- [PowerShell 中的 AWS Lambda 部署程序包 \(p. 360\)](#)
- [PowerShell 中的 AWS Lambda 函数处理程序 \(p. 361\)](#)
- [PowerShell 中的 AWS Lambda 上下文对象 \(p. 362\)](#)
- [PowerShell 中的 AWS Lambda 函数日志记录 \(p. 362\)](#)
- [PowerShell 中的 AWS Lambda 函数错误 \(p. 366\)](#)

设置 PowerShell 开发环境

要设置编写 PowerShell 脚本的开发环境，请执行以下操作：

1. 安装正确的 PowerShell 版本 – Lambda 对于 PowerShell 的支持基于跨平台的 PowerShell Core 6.0 版本。这意味着，您可以在 Windows、Linux 或 Mac 上开发 PowerShell Lambda 函数。如果您未安装此版本的 PowerShell，可以在[安装 PowerShell Core](#)中查找相关说明。
2. 安装 .NET Core 3.1 开发工具包 – 由于 PowerShell Core 建立在 .NET Core 的基础之上，因此，Lambda 对于 PowerShell 的支持使用相同的 .NET Core 3.1 Lambda 运行时来开发 .NET Core 和 PowerShell Lambda 函数。新的 Lambda PowerShell 发布 cmdlet 使用 .NET Core 3.1 开发工具包创建 Lambda 部署程序包。Microsoft 网站上的[.NET 下载](#)提供了 .NET Core 3.1 开发工具包。请务必安装开发工具包，而不是运行时安装。
3. 安装 AWSLambdaPSCore 模块 – 您可以从[PowerShell 库](#)安装此模块，也可以通过使用以下 PowerShell Core shell 命令来安装此模块：

```
Install-Module AWSLambdaPSCore -Scope CurrentUser
```

4. (可选) 安装适用于 PowerShell 的 AWS 工具 – 您可以在 PowerShell Core 6.0 中安装模块化 [AWS.Tools](#) 或单模块 [AWSPowerShell.NetCore](#) 版本，以便在您的 PowerShell 环境中使用 Lambda API。有关说明，请参阅[安装适用于 PowerShell 的 AWS 工具](#)。

PowerShell 中的 AWS Lambda 部署程序包

PowerShell Lambda 部署程序包是一个 ZIP 文件，其中包含您的 PowerShell 脚本、PowerShell 脚本所需的 PowerShell 模块，以及托管 PowerShell Core 所需的程序集。

AWSLambdaPSCore 模块具有以下新 cmdlet，可帮助创作和发布 PowerShell Lambda 函数。

AWSLambdaPSCore Cmdlets

- Get-AWSPowerShellLambdaTemplate – 返回入门模板列表。
- New-AWSPowerShellLambda – 根据模板创建初始 PowerShell 脚本。
- Publish-AWSPowerShellLambda – 将给定 PowerShell 脚本发布到 Lambda。
- New-AWSPowerShellLambdaPackage – 创建 Lambda 部署程序包，可以在 CI/CD 系统中用于部署。

为了帮助使用 Lambda 开始编写并调用 PowerShell 脚本，您可以使用 New-AWSPowerShellLambda cmdlet 基于模板创建一个入门脚本。您可以使用 Publish-AWSPowerShellLambda cmdlet 将脚本部署到 AWS Lambda。然后，您可以通过命令行或控制台测试您的脚本。

要创建、上传和测试新的 PowerShell 脚本，请遵循以下步骤：

1. 运行以下命令，以查看可用模板列表：

```
PS C:\> Get-AWSPowerShellLambdaTemplate

Template          Description
-----          -----
Basic            Bare bones script
CodeCommitTrigger Script to process AWS CodeCommit Triggers
...
...
```

2. 运行以下命令，以基于 Basic 模板创建示例脚本：

```
New-AWSPowerShellLambda -ScriptName MyFirstPSScript -Template Basic
```

当前目录的新子目录中创建一个名为 MyFirstPSScript.ps1 的新文件。目录名称基于 -ScriptName 参数。您可以使用 -Directory 参数来选择其他目录。

您可以看到新文件包含以下内容：

```
# PowerShell script file to be executed as a AWS Lambda function.
#
# When executing in Lambda the following variables will be 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 will be 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. 要查看日志消息如何从您的 PowerShell 脚本发送到 CloudWatch Logs，请取消示例脚本的 Write-Host 行的注释。

要演示如何从 Lambda 函数返回数据，请使用 \$PSSessionTable 在脚本末尾添加新的一行。这会将 \$PSSessionTable 添加到 PowerShell 管道。PowerShell 脚本完成后，PowerShell 管道中的最后一个对象是 Lambda 函数的返回数据。\$PSSessionTable 是 PowerShell 全局变量，还提供有关正在运行的环境的信息。

做出这些更改之后，示例脚本的最后两行如下所示：

```
Write-Host (ConvertTo-Json -InputObject $LambdaInput -Compress -Depth 5)  
$PSSessionTable
```

4. 编辑 MyFirstPSScript.ps1 文件后，将目录更改为脚本的位置。然后运行以下命令，将脚本发布到 AWS Lambda：

```
Publish-AWSPowerShellLambda -ScriptPath .\MyFirstPSScript.ps1 -Name MyFirstPSScript -Region us-east-2
```

注意，-Name 参数指定 Lambda 函数名称，该名称将显示在 Lambda 控制台中。您可以使用此函数手动调用脚本。

5. 使用 AWS CLI invoke 命令调用您的函数。

```
> aws lambda invoke --function-name MyFirstPSScript out
```

PowerShell 中的 AWS Lambda 函数处理程序

当调用 Lambda 函数时，Lambda 处理程序会调用 PowerShell 脚本。

调用 PowerShell 脚本时，以下变量已预定义：

- **\$LambdaInput** – 包含处理程序的输入的 PSObject。该输入可以是事件数据（由事件源发布）或您提供的自定义输入（如字符串或任意自定义数据对象）。
- **\$LambdaContext** – 一个 Amazon.Lambda.Core.ILambdaContext 对象，用于访问有关当前执行的信息 — 例如当前函数的名称、内存限制、剩余执行时间和日志记录。

例如，请考虑以下 PowerShell 示例代码。

```
#Requires -Modules @{ModuleName='AWSPowerShell.NetCore';ModuleVersion='3.3.618.0'}  
Write-Host 'Function Name:' $LambdaContext.FunctionName
```

此脚本返回从 \$LambdaContext 变量获得的 FunctionName 属性。

Note

您需要使用 PowerShell 脚本中的 #Requires 语句指示您的脚本所依赖的模块。此语句可执行两个重要任务。1) 它可告知其他开发人员该脚本使用的模块，2) 它可标识 AWS PowerShell 工具在部署过程中使用脚本打包所需要的从属模块。有关 #Requires 语句的更多信息，请参阅[关于 Requires](#)。有关 PowerShell 部署程序包的更多信息，请参阅[PowerShell 中的 AWS Lambda 部署程序包 \(p. 360\)](#)。

当您的 PowerShell Lambda 函数使用 AWS PowerShell cmdlet 时，请务必设置一个 #Requires 语句，使其引用 AWSPowerShell.NetCore 模块，该模块支持 PowerShell Core — 而不是 AWSPowerShell 模块，该模块仅支持 Windows PowerShell。此外，请确保使用 3.3.270.0 版或更

新版本的 `AWSPowerShell.NetCore`，其优化了 cmdlet 导入过程。如果使用较旧版本，冷启动时间较长。有关更多信息，请参阅[适用于 PowerShell 的 AWS 工具](#)。

返回数据

有些 Lambda 调用旨在为调用方返回数据。例如，如果某个调用是为了响应来自 API 网关的 Web 请求，则我们的 Lambda 函数需要返回响应。对于 PowerShell Lambda，添加到 PowerShell 管道的最后一个对象是 Lambda 调用返回的数据。如果对象是字符串，数据将按原样返回。否则，对象将使用 `ConvertTo-Json` cmdlet 转换为 JSON。

例如，请考虑以下 PowerShell 语句，该语句在 PowerShell 管道中添加 `$PSVersionTable`：

```
$PSVersionTable
```

PowerShell 脚本完成后，PowerShell 管道中的最后一个对象是 Lambda 函数的返回数据。`$PSVersionTable` 是 PowerShell 全局变量，还提供有关正在运行的环境的信息。

PowerShell 中的 AWS Lambda 上下文对象

当 Lambda 运行您的函数时，它通过使 `$LambdaContext` 变量可用于[处理程序 \(p. 361\)](#)来传递上下文信息。此变量提供的方法和属性包含有关调用、函数和执行环境的信息。

上下文属性

- `FunctionName` – Lambda 函数的名称。
- `FunctionVersion` – 函数的[版本 \(p. 63\)](#)。
- `InvokedFunctionArn` – 用于调用函数的 Amazon 资源名称 (ARN)。指示调用方是否已指定版本或别名。
- `MemoryLimitInMB` – 为函数分配的内存量。
- `AwsRequestId` – 调用请求的标识符。
- `LogGroupName` – 函数的日志组。
- `LogStreamName` – 函数实例的日志流。
- `RemainingTime` – 在执行超时以前剩余的毫秒数。
- `Identity` – (移动应用程序) 有关授权请求的 Amazon Cognito 身份的信息。
- `ClientContext` – (移动应用程序) 由客户端应用程序向 Lambda 提供的客户端上下文。
- `Logger` – 函数的[记录器对象 \(p. 362\)](#)。

下面的 PowerShell 代码片段显示了一个打印部分上下文信息的简便处理程序函数。

```
#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
```

PowerShell 中的 AWS Lambda 函数日志记录

您的 Lambda 函数带有一个 CloudWatch Logs 日志组，其中包含您的函数的每个实例的日志流。运行时会将每个调用的详细信息发送到该日志流，然后中继日志和来自您的函数代码的其他输出。

要从您的函数代码输出日志，您可以在 [Microsoft.PowerShell.Utility](#) 上使用 cmdlets，或者使用任何写入到 `stdout` 或 `stderr` 的日志记录模块。下面的示例使用了 `Write-Host`。

Example function/Handler.ps1 – 日志记录

```
#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 日志格式

```
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
```

.NET 运行时记录每次调用的 START、END 和 REPORT 行。报告行提供了以下详细信息。

报告日志

- RequestId – 调用的唯一请求 ID。
- Duration (持续时间) – 函数的处理程序方法处理事件所花费的时间。
- Billed Duration (计费持续时间) – 针对调用计费的时间量。
- Memory Size (内存大小) – 分配给函数的内存量。
- Max Memory Used (最大内存使用量) – 函数使用的内存量。
- Init Duration (初始持续时间) – 对于提供的第一个请求，为运行时在处理程序方法外部加载函数和运行代码所花费的时间。

- XRAY TraceId – 对于跟踪的请求，为 [AWS X-Ray 跟踪编码 \(p. 371\)](#)。
- SegmentId – 对于跟踪的请求，为 X-Ray 分段 ID。
- Sampled (采样) – 对于跟踪的请求，为采样结果。

您可以在 Lambda 控制台中、在 CloudWatch Logs 控制台中或从命令行查看日志。

小节目录

- [在 AWS 管理控制台中查看日志 \(p. 364\)](#)
- [使用 AWS CLI \(p. 364\)](#)
- [删除日志 \(p. 365\)](#)

在 AWS 管理控制台中查看日志

当您在函数配置页上测试函数时，Lambda 控制台会显示日志输出。要查看所有调用的日志，请使用 CloudWatch Logs 控制台。

查看 Lambda 函数的日志

1. 打开 [CloudWatch 控制台的日志页面](#)。
2. 选择您的函数 (`/aws/lambda/function-name`) 的日志组。
3. 选择列表中的第一个流。

每个日志流对应一个[函数实例 \(p. 109\)](#)。当您更新函数以及创建更多实例以处理多个并发调用时，会显示新的流。要找到特定调用的日志，您可以使用 X-Ray 分析您的函数并在跟踪中记录有关请求和日志流的详细信息。如需将日志和跟踪与 X-Ray 相关联的示例应用程序，请参阅[AWS Lambda 错误处理器示例应用程序 \(p. 128\)](#)。

使用 AWS CLI

要从命令行获取调用的日志，请使用 `--log-type` 选项。响应包含一个 `LogResult` 字段，该字段包含来自调用的多达 4 KB 的 base64 编码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail
{
    "StatusCode": 200,
    "LogResult":
    "U1RBULQgUmVxdWVzdElkOia4N2QwNDRiOC1mMTU0LTEzZTgtOGNkYS0yOTc0YzVlNGZiMjEgVmVyc2lvb...",
    "ExecutedVersion": "$LATEST"
}
```

您可以使用 base64 实用程序来解码日志。

```
$ aws lambda invoke --function-name my-function out --log-type Tail \
--query 'LogResult' --output text | base64 -d
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: 100 ms          Memory Size: 128 MB      Max Memory Used: 73 MB
```

base64 实用工具在 Linux、macOS 和 [Ubuntu on Windows](#) 上可用。对于 macOS，命令为 `base64 -D`。

要从命令行获取完整的日志事件，您可以在函数输出中包含日志流名称，如上例中所示。以下示例脚本调用名为 my-function 的函数并下载最后 5 个日志事件。

Example get-logs.sh 脚本

此示例要求 my-function 返回日志流 ID。

```
#!/bin/bash
aws lambda invoke --function-name my-function --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
```

此脚本使用 sed 从输出文件中删除引号，并休眠 15 秒以等待日志可用。输出包括来自 Lambda 的响应，以及来自 get-log-events 命令的输出。

```
$ ./get-logs.sh
{
  "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: 100 ms\tMemory Size: 128 MB\tMax Memory Used: 75 MB\t",
      "ingestionTime": 1559763018353
    }
  ],
  "nextForwardToken": "f/34783877304859518393868359594929986069206639495374241795",
  "nextBackwardToken": "b/34783877303811383369537420289090800615709599058929582080"
}
```

删除日志

在删除函数时，不会自动删除日志组。为避免存储无限的日志，请删除日志组或者[配置保留期](#)，在该期限后会自动删除日志。

PowerShell 中的 AWS Lambda 函数错误

如果 Lambda 函数出现终止错误，AWS Lambda 会识别到失败并将错误信息序列化为 JSON 返回。

请考虑以下 PowerShell 脚本示例语句：

```
throw 'The Account is not found'
```

在调用此 Lambda 函数时，它将引发终止错误，并且 AWS Lambda 返回以下错误消息：

```
{
  "errorMessage": "The Account is not found",
  "errorType": "RuntimeException"
}
```

注意 `errorType` 是 `RuntimeException`，这是 PowerShell 引发的默认异常。您可以通过引发错误来使用自定义错误类型，如下所示：

```
throw @{'Exception'='AccountNotFound';'Message'='The Account is not found'}
```

通过将 `errorType` 设置为 `AccountNotFound`，错误消息将进行序列化：

```
{
  "errorMessage": "The Account is not found",
  "errorType": "AccountNotFound"
}
```

如果您不需要错误消息，您可以以错误代码的格式引发字符串。错误代码格式要求字符串以字符开头，且后面只能包含字母和数字，不含空格或符号。

例如，如果您的 Lambda 函数包含以下内容：

```
throw 'AccountNotFound'
```

错误将进行序列化，如下所示：

```
{
  "errorMessage": "AccountNotFound",
  "errorType": "AccountNotFound"
}
```

对基于 Lambda 的应用程序进行监控和问题排查

AWS Lambda 将代表您自动监控 Lambda 函数并通过 Amazon CloudWatch 报告指标。为帮助您监控代码的运行情况，Lambda 会自动追踪请求数、每个请求的执行持续时间和产生错误的请求数。它还会发布相关的 CloudWatch 指标。您可以借助这些指标设置 CloudWatch 自定义警报。有关 CloudWatch 的更多信息，请参阅 [Amazon CloudWatch 用户指南](#)。

您可以使用 AWS Lambda 控制台、CloudWatch 控制台和其他 AWS 资源查看每个 Lambda 函数的请求速率和错误率。以下主题介绍 Lambda CloudWatch 指标及如何访问它们。

- 在 AWS Lambda 控制台中监控函数 (p. 367)
- 使用 AWS Lambda 函数指标 (p. 368)

您可以在代码中插入日志记录语句来帮助验证代码是否按预期运行。Lambda 自动与 Amazon CloudWatch Logs 集成。它将代码中的所有日志推送到与 Lambda 函数 (`/aws/lambda/<function name>`) 关联的 CloudWatch Logs 组。要了解有关日志组以及通过 CloudWatch 控制台访问日志组的更多信息，请参阅 Amazon CloudWatch Logs 用户指南 中的 [使用日志组和日志流](#)。有关如何访问 CloudWatch 日志条目的信息，请参阅 [访问 AWS Lambda 的 Amazon CloudWatch 日志 \(p. 370\)](#)。

Note

如果 Lambda 函数代码正在执行，但几分钟后您仍未看到有任何日志数据生成，则可能是该 Lambda 函数的执行角色未授予将日志数据写入到 CloudWatch Logs 中的权限。有关如何确保正确设置执行角色以授予这些权限的更多信息，请参阅 [AWS Lambda 执行角色 \(p. 30\)](#)。

每项监控服务均提供一个免费套餐。如果您的应用程序超出了免费套餐限制，则定价将基于使用量。有关更多信息，请参阅 [CloudWatch 定价](#) 和 [X-Ray 定价](#)。另请注意，标准存储费率将适用于由 Lambda 服务存储的 CloudWatch 日志。

监控服务遵循以下使用模式：

- 每当 Lambda 函数运行时，AWS Lambda 会向 CloudWatch 报告指标和日志。
- 当您查看 Lambda 函数的“Monitoring (监控)”页面时，CloudWatch Logs Insights 将运行。
- 当您为函数启用 X-Ray 时或当上游服务启用 X-Ray 时，X-Ray 将开始记录跟踪。

在 AWS Lambda 控制台中监控函数

AWS Lambda 将代表您监控函数并将指标发送到 Amazon CloudWatch。指标包括总请求数、持续时间和错误率。Lambda 控制台将为这些指标创建图表，并在每个函数的监控页面上显示它们。

访问监控控制台

1. 打开 [Lambda 控制台](#)。
2. 打开 Lambda 控制台 [函数页面](#)。
3. 选择 Monitoring。

控制台提供了以下图表。

Lambda 监控图表

- 调用数 – 每 5 分钟内调用函数的次数。

- 持续时间 – 平均、最小和最大执行时间。
- 错误数和成功率 (%) – 错误的数量，以及完成且没有错误的执行的百分比。
- 限制 – 由于并发限制而导致执行失败的次数。
- IteratorAge – 对于流事件源，当 Lambda 接收该源并调用函数时，批处理中的最后一个项的存在时间。
- Async delivery failures (异步传输失败次数) – Lambda 尝试写入到目标或死信队列时发生的错误的数量。
- Concurrent executions (并发执行数) – 正在处理事件的函数实例的数目。

要查看 CloudWatch 中图表的定义，请从图表右上角的菜单中选择在指标中查看。有关 Lambda 记录的指标的更多信息，请参阅[使用 AWS Lambda 函数指标 \(p. 368\)](#)。

此外，控制台还显示来自 CloudWatch Logs Insights 的报告，这些报告是根据函数日志中的信息编译的。您可以将这些报告添加到 CloudWatch Logs 控制台中的自定义控制面板。使用查询开始构建您自己的报告。

要查看查询，请从报告右上角的菜单中选择在 CloudWatch Logs Insights 中查看。

使用 AWS Lambda 函数指标

当您的函数完成对事件的处理时，Lambda 会将有关调用的指标发送到 Amazon CloudWatch。您可以在 CloudWatch 控制台中使用这些指标构建图表和控制面板，并设置警报以响应利用率、性能或错误率的变化。使用维度可以按函数名称、别名或版本对函数指标进行筛选和排序。

在 CloudWatch 控制台中查看指标

1. 打开 [Amazon CloudWatch 控制台的指标页面 \(AWS/Lambda 命名空间\)](#)。
2. 选择维度。
 - 按函数名称 (FunctionName) – 查看函数的所有版本和别名的聚合指标。
 - 按资源 (Resource) – 查看函数的版本或别名的指标。
 - 按已执行版本 (ExecutedVersion) – 查看别名和版本组合的指标。使用 ExecutedVersion 维度可以比较函数的两个版本的错误率，这两个版本都是[加权别名 \(p. 65\)](#)的目标。
 - 跨所有函数 (无) – 查看当前 AWS 区域中所有函数的聚合指标。
3. 选择指标以将其添加到图表。

默认情况下，图表将使用所有指标的 Sum 统计数据。要选择其他统计数据并自定义图表，请使用 Graphed metrics (图表化指标) 选项卡上的选项。

指标上的时间戳反映函数被调用的时间。根据执行的持续时间，这可能是发出指标前的几分钟。例如，如果您的函数的超时值为 10 分钟，请查看 10 分钟之前的情况，以获取准确的指标。

有关 CloudWatch 的更多信息，请参阅[Amazon CloudWatch 用户指南](#)。

小目录

- [使用调用指标 \(p. 368\)](#)
- [使用性能指标 \(p. 369\)](#)
- [使用并发指标 \(p. 369\)](#)

使用调用指标

调用指标是调用结果的二进制指示器。例如，如果函数返回一个错误，则 Lambda 会发送值为 1 的 Errors 指标。要获取每分钟发生的函数错误数，请查看一分钟时段内的 Errors 指标的总数。

应使用 Sum 统计数据查看以下指标。

调用指标

- **Invocations** – 函数代码的执行次数，包括成功的执行和导致出现函数错误的执行。如果调用请求受到限制或导致出现[调用错误 \(p. 484\)](#)，则不会记录调用。这等于计费请求的数目。
- **Errors** – 导致出现函数错误的调用的次数。函数错误包括您的代码所引发的异常和 Lambda 运行时所引发的异常。运行时返回因超时和配置错误等问题导致的错误。要计算错误率，请将 Errors 的值除以 Invocations 的值。
- **DeadLetterErrors** – 对于[异步调用 \(p. 83\)](#)，为 Lambda 尝试将事件发送到死信队列但失败的次数。权限错误、资源误配或大小限制可能会致发生死信错误。
- **DestinationDeliveryFailures** – 对于异步调用，Lambda 尝试将事件发送到[目标 \(p. 24\)](#)但失败的次数。权限错误、资源误配或大小限制可能会导致发生传输错误。
- **Throttles** – 受限制的调用请求数。当所有函数实例都在处理请求并且没有用于扩展的并发性时，Lambda 将拒绝其他请求，并出现[TooManyRequestsException \(p. 484\)](#)。受限制的请求和其他调用错误不会计为 Invocations 或 Errors。
- **ProvisionedConcurrencyInvocations** – 根据预置的并发性 (p. 54) 执行函数代码的次数。
- **ProvisionedConcurrencySpilloverInvocations** – 在使用所有预置的并发性时基于标准并发性执行函数代码的次数。

使用性能指标

性能指标提供了有关单个调用的性能详细信息。例如，Duration 指标指示函数处理事件所花费的时间量（以毫秒为单位）。要了解函数处理事件的速度，请使用 Average 或 Maximum 统计数据查看这些指标。

性能指标

- **Duration** – 函数代码处理事件所花费的时间量。对于由函数实例处理的第一个事件，该时间量包括[初始化时间 \(p. 17\)](#)。调用的计费持续时间是已舍入到最近的 100 毫秒的 Duration 值。
- **IteratorAge** – 对于从流读取的[事件源映射 \(p. 90\)](#)，为事件中最后一条记录的期限。期限是指流接收记录的时间到事件源映射将事件发送到函数的时间之间的时间量。

Duration 还支持[百分位数统计数据](#)。使用百分位数可排除偏离平均统计数据和最大统计数据的异常值。例如，P95 统计数据显示 95% 的执行的最长持续时间，并排除最慢的 5% 的执行。

使用并发指标

Lambda 将并发指标报告为跨函数、版本、别名或 AWS 区域处理事件的实例数的总计数。要查看接近并发限制的程度，请使用 Max 统计数据查看这些指标。

并发指标

- **ConcurrentExecutions** – 正在处理事件的函数实例的数目。如果此数目达到区域的[并发执行限制 \(p. 27\)](#)或您在函数上配置的[预留并发限制 \(p. 54\)](#)，则其他调用请求将受到限制。
- **ProvisionedConcurrentExecutions** – 根据[预配置并发 \(p. 54\)](#)处理事件的函数实例的数目。对于具有预配置并发的别名或版本的每次调用，Lambda 都会发出当前计数。
- **ProvisionedConcurrencyUtilization** – 对于版本或别名，将 ProvisionedConcurrentExecutions 值除以分配的预置并发总数。例如，.5 指明有 50% 的已分配预配置并发正在使用中。
- **UnreservedConcurrentExecutions** – 对于 AWS 区域，为由不具有预留并发的函数处理的事件数。

访问 AWS Lambda 的 Amazon CloudWatch 日志

AWS Lambda 会自动替您监控 Lambda 函数，并通过 Amazon CloudWatch 报告各项指标。为帮助您诊断函数中的问题，Lambda 会记录您的函数处理的所有请求，并通过 Amazon CloudWatch Logs 自动存储您的代码生成的日志。

您可以在代码中插入日志记录语句来帮助验证代码是否按预期运行。Lambda 自动与 CloudWatch Logs 集成，并将您的代码的所有日志推送到与 Lambda 函数关联的 CloudWatch Logs 组（即名为 /aws/lambda/<###>#> 的组）。有关日志组和通过 CloudWatch 控制台访问它们的更多信息，请参阅 Amazon CloudWatch 用户指南 中的 [监控系统、应用程序和自定义日志文件](#)。

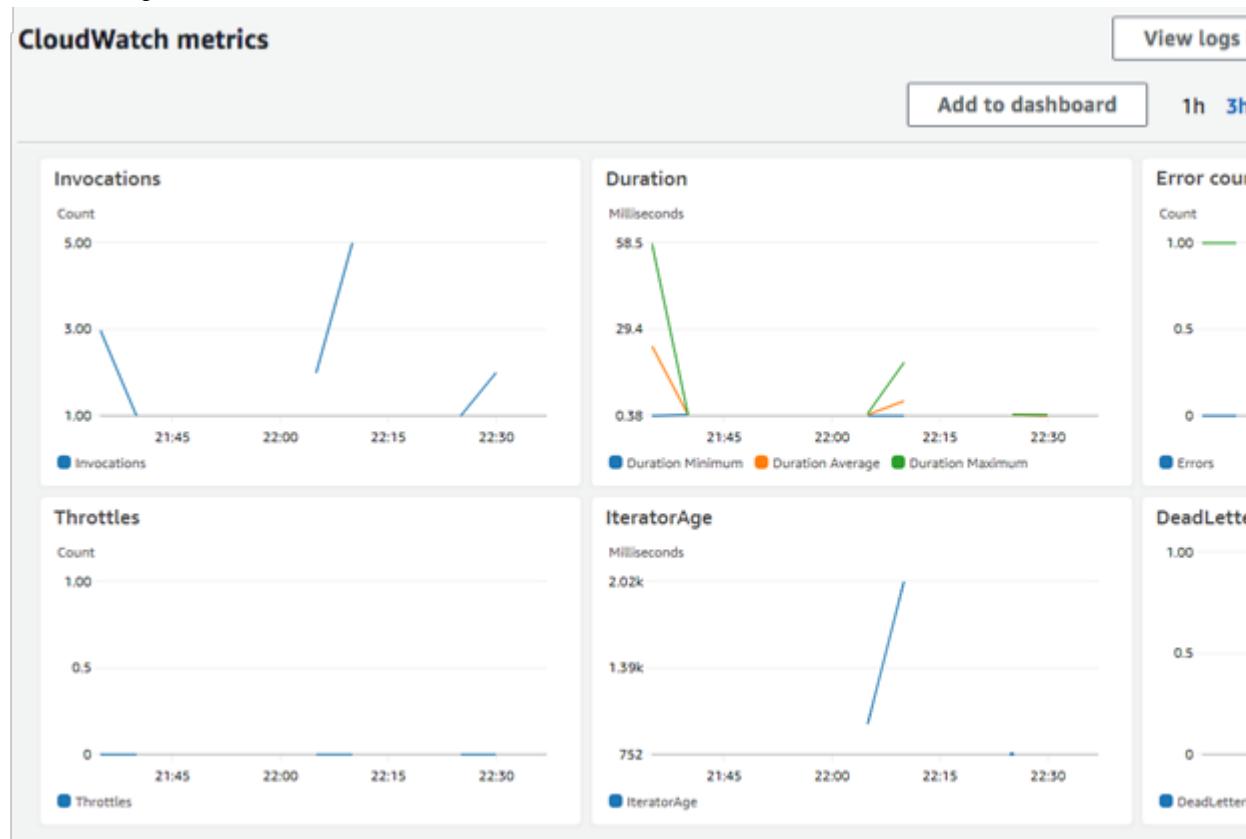
您可以借助 Lambda 控制台、CloudWatch 控制台、AWS CLI 或 CloudWatch API 查看 Lambda 日志。下面的流程介绍如何使用 Lambda 控制台查看日志。

Note

使用 Lambda 日志没有额外的费用；不过，会收取标准 CloudWatch Logs 费用。有关更多信息，请参阅 [CloudWatch 定价](#)。

使用 Lambda 控制台查看日志

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 选择 Monitoring。



所示为 Lambda 函数指标的图形化表示。

4. 选择查看 CloudWatch 中的日志。

Lambda 使用您的函数的权限将日志上传到 CloudWatch Logs。如果您未在控制台中看到日志，请检查您的[执行角色权限 \(p. 30\)](#)。

使用 AWS X-Ray

您可以使用 AWS X-Ray 来可视化应用程序的组件、确定性能瓶颈以及对导致错误的请求进行故障排除。您的 Lambda 函数会将跟踪数据发送到 X-Ray，X-Ray 将处理这些数据以生成服务映射和可搜索的跟踪摘要。

如果您在调用函数的服务中启用了 X-Ray 跟踪，则 X-Ray 会自动将跟踪发送到 Lambda。上游服务（例如 Amazon API Gateway）或通过 X-Ray 开发工具包检测的托管于 Amazon EC2 上的应用程序将对传入请求进行采样，并添加跟踪头来指示 Lambda 是否发送跟踪。有关支持活动检测的服务的完整列表，请参阅 AWS X-Ray 开发人员指南中的[受支持的 AWS 服务](#)。

要跟踪没有跟踪标头的请求，请在函数的配置中启用活动跟踪。

您可以在函数配置中启用跟踪。

启用活动跟踪

1. 打开 Lambda 控制台 [函数页面](#)。
2. 选择函数。
3. 在 AWS X-Ray 下，选择 Active tracing (活动跟踪)。
4. 选择保存。

您的函数需要权限才能将跟踪数据上传到 X-Ray。在 Lambda 控制台中启用活动跟踪后，Lambda 会将所需权限添加到函数的[执行角色 \(p. 30\)](#)。如果没有，请将 [AWSXRayDaemonWriteAccess](#) 策略添加到执行角色。

X-Ray 应用采样算法确保跟踪有效，同时为应用程序所服务的请求提供代表性样本。默认采样规则是每秒 1 个请求和 5% 的其他请求。

利用 AWS X-Ray 跟踪基于 Lambda 的应用程序

AWS X-Ray 是一种 AWS 服务，使您可以通过 AWS Lambda 应用程序检测、分析和优化性能问题。X-Ray 从 Lambda 服务和组成您的应用程序的所有上游或下游服务中收集元数据。X-Ray 使用这些元数据生成详细的服务图形，用于说明性能瓶颈，延迟峰值，以及影响 Lambda 应用程序性能的其他问题。

使用 [AWS X-Ray 服务地图上的 Lambda \(p. 371\)](#) 识别出有问题的资源或组件后，您可以进行放大，查看请求的可视化形式。此可视化形式涵盖的时间范围从事件源触发 Lambda 函数开始，直到函数执行完成。X-Ray 可以为您提供函数操作的分析结果，如 Lambda 函数对其他服务进行的下游调用的信息。此外，X-Ray 与 Lambda 的集成还便于您了解 AWS Lambda 服务开销。它通过显示请求的停留时间和调用数量等具体信息来实现此功能。

Note

只有目前与 X-Ray 集成的服务可在您的 Lambda 跟踪之外显示为独立跟踪。有关当前支持 X-Ray 的服务列表，请参阅[将 AWS X-Ray 与其他 AWS 服务集成](#)。

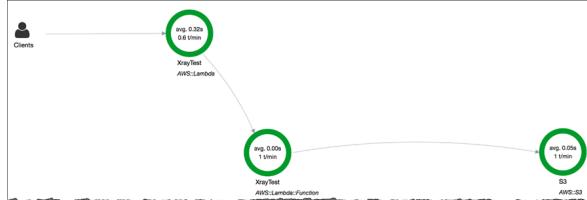
AWS X-Ray 服务地图上的 Lambda

针对 Lambda 处理的请求，X-Ray 在服务地图上显示三种类型的节点：

- Lambda 服务 (AWS::Lambda) – 此类节点表示请求用于 Lambda 服务的时间。Lambda 首次接收请求时开始计时，直到请求离开 Lambda 服务时计时结束。

- Lambda 函数 (AWS::Lambda::Function) – 此类节点表示 Lambda 函数的执行时间。
- 下游服务调用 – 在这种类型中，Lambda 函数发出的每个下游服务调用均表示为一个单独的节点。

下图中的节点（从左到右）依次代表：Lambda 服务、用户函数和对 Amazon S3 的下游调用：



有关更多信息，请参阅[查看服务地图](#)。

Lambda 作为 AWS X-Ray 跟踪

您可以在服务地图中进行放大，查看您的 Lambda 函数的跟踪视图。跟踪将显示与函数调用相关的深度信息，以分段和子分段表示。

- Lambda 服务分段 – 此分段表示不同的信息，具体取决于调用函数的事件源：
 - 同步和流事件源 – 此服务分段计算从 Lambda 服务接收请求/事件，直到请求离开 Lambda 服务（完成请求的最终调用）所经历的时间。
 - 异步 – 此服务分段表示响应时间，即 Lambda 服务向客户端返回 202 响应所需的时间。

Lambda 服务分段可以包含两种类型的子分段：

- 停留时间（仅限异步调用）– 表示函数在被调用之前在 Lambda 服务中花费的时间。这个子分段在 Lambda 服务接收请求/事件时开始，在首次调用 Lambda 函数时结束。
- 尝试 – 表示单次调用尝试，包括 Lambda 服务引起的任何开销。开销示例包括初始化函数代码所花费的时间和函数执行时间。
- Lambda 函数分段 – 表示函数针对给定调用尝试的执行时间。该分段于函数处理程序启动时开始，函数终止时结束。该分段可以包含三种类型的子分段：
 - 初始化 – 运行函数 `initialization` 代码（定义为 Lambda 处理程序或静态初始化程序的外部代码）所花费的时间。
 - 下游调用 – Lambda 函数的代码对其他 AWS 服务或下游 HTTP 请求的调用。
 - 自定义子分段 – 可以使用 X-Ray 开发工具包添加到 Lambda 函数分段中的自定义子分段。

Lambda 函数分段由 Lambda 服务代表客户生成。它覆盖在 Lambda 函数代码中创建的任何分段，包括 X-Ray 开发工具包生成的通过中间件捕获请求的分段。如果分段是在 Lambda 函数代码中创建的，它将作为 Facade 分段实现。Facade 分段只允许在其中添加和删除自定义子分段；所有其他分段操作都被视为无操作。

对于每个跟踪调用，Lambda 会发送 Lambda 服务分段及其所有子分段。无论您使用何种运行时，都会发出此分段及其子分段。

从 Lambda 函数发送跟踪分段

对于每个跟踪调用，Lambda 会发送 Lambda 服务分段及其所有子分段。此外，Lambda 将发送 Lambda 函数分段和 `init` 子分段。发送这些分段无需考虑函数的运行时，也不需要更改代码或其他库。如果您希望 Lambda 函数的 X-Ray 跟踪包含自定义分段、注释或来自下游调用的子分段，可能需要包含其他库并对代码添加注释。

请注意，任何检测代码都必须在 Lambda 函数处理程序内而不是在初始化代码中实现。

以下示例介绍了如何在支持的运行时内执行此操作：

- 在 AWS Lambda 中检测 Python 代码 (p. 288)
- 在 AWS Lambda 中检测 Node.js 代码 (p. 275)
- 在 AWS Lambda 中检测 Java 代码 (p. 323)
- 在 AWS Lambda 中检测 Go 代码 (p. 339)

Lambda 环境中的 AWS X-Ray 守护程序

[AWS X-Ray 守护程序](#)是收集原始分段数据并将其中继到 AWS X-Ray 服务的软件应用程序。守护程序与 AWS X-Ray 开发工具包结合使用，使得开发工具包发送的数据可以到达 X-Ray 服务。

当您跟踪 Lambda 函数时，X-Ray 守护程序会自动在 Lambda 环境中运行，收集跟踪数据并发送到 X-Ray。进行跟踪时，X-Ray 守护程序会占用最多 16 MB 或 3% 的函数内存分配。例如，如果您为 Lambda 函数分配了 128 MB 的内存，X-Ray 守护程序会占用 16 MB 的函数内存分配。如果您为 Lambda 函数分配了 1024 MB 内存，分配给 X-Ray 守护程序的内存为 31 MB (3%)。有关更多信息，请参阅 [AWS X-Ray 守护程序](#)。

Note

Lambda 将尝试终止 X-Ray 守护程序，以免超出函数的内存限制。例如，假设您为 Lambda 函数分配了 128 MB 内存，这就意味着 X-Ray 守护程序将获得 16 MB。这样您的 Lambda 函数获得的内存分配为 112 MB。但是，如果您的函数超过 112 MB，X-Ray 守护程序将被终止，以避免引发内存不足错误。

使用环境变量与 AWS X-Ray 通信

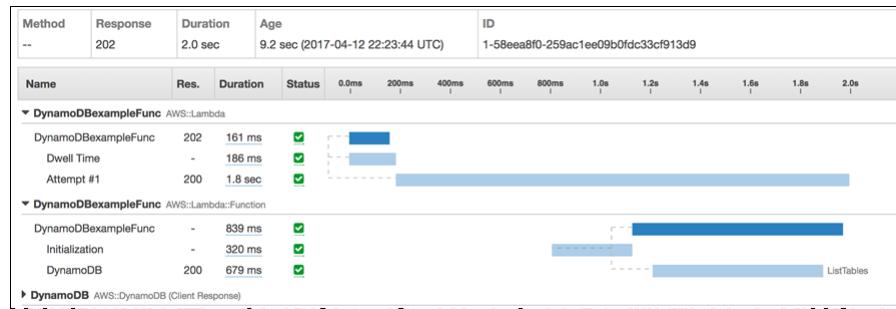
AWS Lambda 使用环境变量以便于与 X-Ray 守护程序进行通信和配置 X-Ray 开发工具包。

- `_X_AMZN_TRACE_ID`：包含跟踪标头，其中包括采样决策、跟踪 ID 和父分段 ID。（要了解有关这些属性的更多信息，请参阅[跟踪标头](#)。）如果调用您的函数时 Lambda 收到跟踪标头，该标头将用于填充 `_X_AMZN_TRACE_ID` 环境变量。如果 Lambda 未收到跟踪标头，将为您生成一个跟踪标头。
- `AWS_XRAY_CONTEXT_MISSING`：您的函数尝试记录 X-Ray 数据，但跟踪标头不可用时，X-Ray 开发工具包使用此变量确定其行为。默认情况下，Lambda 将此值设为 `LOG_ERROR`。
- `AWS_XRAY_DAEMON_ADDRESS`：此环境变量公开了 X-Ray 守护程序的地址，格式为：`IP_ADDRESS:PORT`。您可以使用 X-Ray 守护程序的地址，直接将跟踪数据发送到 X-Ray 守护程序，而无需使用 X-Ray 开发工具包。

在 AWS X-Ray 控制台中跟踪 Lambda：示例

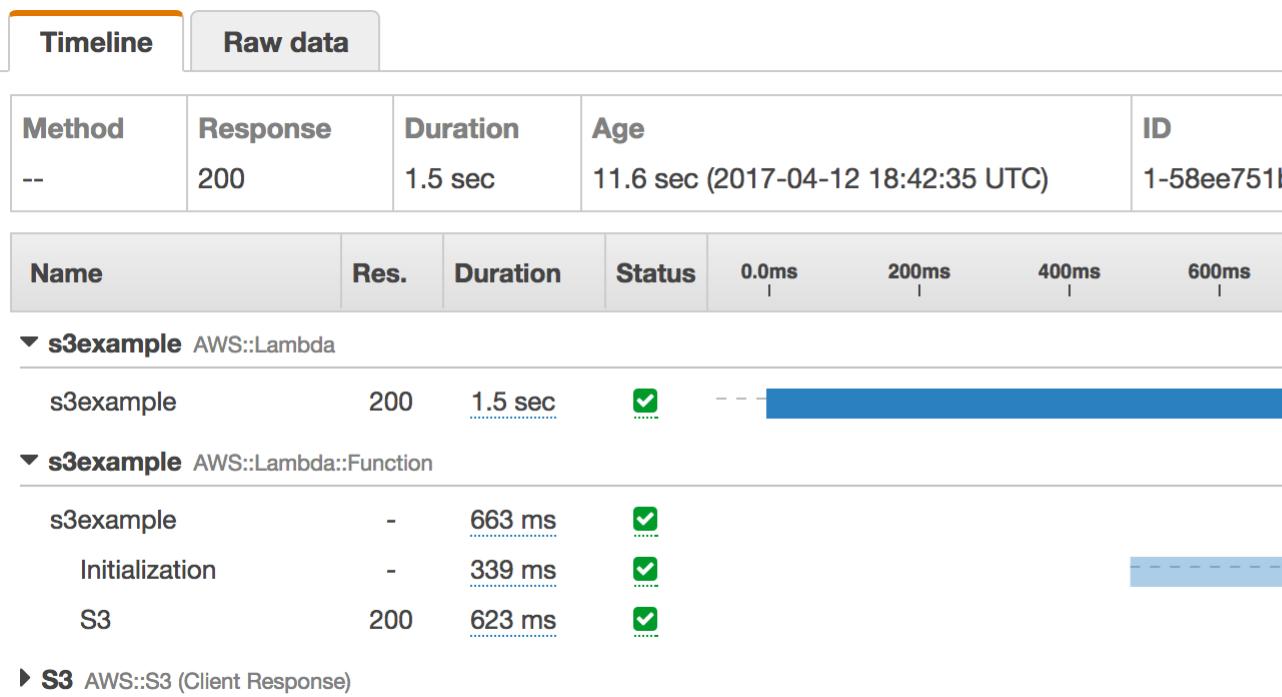
以下示例对两个不同的 Lambda 函数进行了 Lambda 跟踪。每个跟踪展示不同调用类型的跟踪结构：异步和同步。

- 异步 – 以下示例展示具有一个成功调用和一个对 DynamoDB 进行下游调用的异步 Lambda 请求。



Lambda 服务分段封装响应时间，即将响应（例如 202）返回至客户端所用的时间。其中包括在 Lambda 服务队列中花费的时间（停留时间）的子分段和每次调用尝试的子分段。（以上示例中只出现了一次调用尝试。）服务分段中的每次尝试子分段具有相应的用户函数分段。在此示例中，用户函数分段包含两个子分段：初始化子分段，表示函数在处理程序之前运行的初始化代码，以及下游调用子分段，表示对 DynamoDB 的 ListTables 调用。

- 每个调用子分段和每个下游调用都会显示状态代码和错误消息。
- 同步 – 以下示例展示对 Amazon S3 进行下游调用的同步请求。



Lambda 服务分段捕获请求在 Lambda 服务中花费的全部时间。该服务分段将有一个相应的用户函数分段。在此示例中，用户函数分段包含一个表示函数初始化代码（在处理程序之前运行的代码）的子分段；以及表示对 Amazon S3 的 PutObject 调用的子分段。

Note

如果您要跟踪 HTTP 调用，需要使用 HTTP 客户端。有关更多信息，请参阅[使用适用于 Java 的 X-Ray 开发工具包跟踪对下游 HTTP Web 服务的调用](#)或[使用适用于 Node.js 的 X-Ray 开发工具包跟踪对下游 HTTP Web 服务的调用](#)。

使用 AWS CloudTrail 记录 AWS Lambda API 调用

AWS Lambda 与 AWS CloudTrail 集成，后者是一项服务，该服务提供由用户、角色或 AWS Lambda 中的 AWS 服务执行的操作的记录。CloudTrail 将 AWS Lambda 的 API 调用作为事件捕获。捕获的调用包含来自 AWS Lambda 控制台和代码的 AWS Lambda API 操作调用。如果您创建跟踪，则可以使 CloudTrail 事件持续传送到 Amazon S3 存储桶（包括 AWS Lambda 的事件）。如果您不配置跟踪，则仍可在 CloudTrail 控制台的 Event history (事件历史记录) 中查看最新事件。通过使用 CloudTrail 收集的信息，您可以确定向 AWS Lambda 发出了什么请求、发出请求的 IP 地址、何人发出的请求、请求的发出时间以及其他详细信息。

要了解有关 CloudTrail 的更多信息（包括如何配置和启用），请参阅 [AWS CloudTrail User Guide](#)。

CloudTrail 中的 AWS Lambda 信息

在您创建 AWS 账户时，即针对该账户启用了 CloudTrail。当 AWS Lambda 中发生受支持的事件活动时，该活动将记录在 CloudTrail 事件中，并与其他 AWS 服务事件一同保存在 Event history (事件历史记录) 中。您可以在 AWS 账户中查看、搜索和下载最新事件。有关更多信息，请参阅[使用 CloudTrail 事件历史记录查看事件](#)。

要持续记录 AWS 账户中的事件（包括 AWS Lambda 的事件），请创建跟踪。通过跟踪，CloudTrail 可将日志文件传送至 Amazon S3 存储桶。默认情况下，在控制台中创建跟踪时，此跟踪应用于所有 AWS 区域。此跟踪在 AWS 分区中记录来自所有区域的事件，并将日志文件传送至您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，进一步分析在 CloudTrail 日志中收集的事件数据并采取行动。有关更多信息，请参阅下列内容：

- [创建跟踪概述](#)
- [CloudTrail 支持的服务和集成](#)
- [为 CloudTrail 配置 Amazon SNS 通知](#)
- [接收多个区域中的 CloudTrail 日志文件](#)和从多个账户中接收 CloudTrail 日志文件

AWS Lambda 支持在 CloudTrail 日志文件中将以下操作记录为事件：

- [AddPermission \(p. 407\)](#)
- [CreateEventSourceMapping \(p. 415\)](#)
- [CreateFunction \(p. 421\)](#)

([CreateFunction](#) 的 CloudTrail 日志省略了 `ZipFile` 参数。)
- [DeleteEventSourceMapping \(p. 432\)](#)
- [DeleteFunction \(p. 436\)](#)
- [GetEventSourceMapping \(p. 451\)](#)
- [GetFunction \(p. 455\)](#)
- [GetFunctionConfiguration \(p. 460\)](#)
- [GetPolicy \(p. 477\)](#)
- [ListEventSourceMappings \(p. 492\)](#)
- [ListFunctions \(p. 498\)](#)
- [RemovePermission \(p. 537\)](#)
- [UpdateEventSourceMapping \(p. 547\)](#)
- [UpdateFunctionCode \(p. 553\)](#)

([UpdateFunctionCode](#) 的 CloudTrail 日志省略了 `ZipFile` 参数。)
- [UpdateFunctionConfiguration \(p. 560\)](#)

每个日志条目都包含有关生成请求的人员的信息。日志中的用户身份信息有助于确定请求是由根或 IAM 用户证书发出，通过某个角色或联合用户的临时安全证书发出，还是由其他 AWS 服务发出。有关更多信息，请参阅[CloudTrail 事件参考](#)中的 `userIdentity` 字段。

日志文件可以在存储桶中存储任意长时间，不过您也可以定义 Amazon S3 生命周期规则以自动存档或删除日志文件。默认情况下，将使用 Amazon S3 服务器端加密 (SSE) 对日志文件进行加密。

如果需要针对日志文件传输快速采取措施，可选择让 CloudTrail 在传输新日志文件时发布 Amazon SNS 通知。有关更多信息，请参阅[为 CloudTrail 配置 Amazon SNS 通知](#)。

您也可以将多个 AWS 区域和多个 AWS 账户的 AWS Lambda 日志文件聚合到单个 S3 存储桶中。有关更多信息，请参阅[使用 CloudTrail 日志文件](#)。

了解 AWS Lambda 日志文件条目

CloudTrail 日志文件可包含一个或多个日志条目，每个条目由多个 JSON 格式的事件组成。一个日志条目表示来自任何源的一个请求，包括有关所请求的操作、所有参数以及操作的日期和时间等信息。日志条目不一定具有任何特定顺序。也即，它们不是公用 API 调用的有序堆栈跟踪。

下面的示例介绍 `GetFunction` 和 `DeleteFunction` 操作的 CloudTrail 日志条目。

```
{  
    "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": "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` 可能包括日期和版本信息（如 `"GetFunction20150331"`），但它仍在引用同一公用 API。有关更多信息，请参阅 AWS CloudTrail 用户指南 中的 [CloudTrail 事件历史记录所支持的服务](#)。

使用 CloudTrail 跟踪函数调用

CloudTrail 还会记录数据事件。您可以启用数据事件日志记录，以便在每次调用 Lambda 函数时记录一个事件。这可帮助您了解哪些身份正在调用函数以及调用频率。有关此选项的更多信息，请参阅[记录跟踪的数据事件](#)。

AWS Lambda 中的安全性

AWS 的云安全性的优先级最高。作为 AWS 客户，您将从专为满足大多数安全敏感型组织的要求而打造的数据中心和网络架构中受益。

安全性是 AWS 和您的共同责任。责任共担模型将其描述为云的 安全性和云中的 安全性：

- 云的安全性 – AWS 负责保护在 AWS 云中运行 AWS 服务的基础设施。AWS 还向您提供可安全使用的服务。作为 [AWS 合规性计划](#) 的一部分，第三方审计人员将定期测试和验证安全性的有效性。要了解适用于 AWS Lambda 的合规性计划，请参阅[合规性计划范围内的 AWS 服务](#)。
- 云中的安全性 – 您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

此文档将帮助您了解如何在使用 Lambda 时应用责任共担模型。以下主题说明如何配置 Lambda 以实现您的安全性和合规性目标。您还将了解如何使用其他 AWS 服务来帮助您监控和保护您的 Lambda 资源。

主题

- [AWS Lambda 中的数据保护 \(p. 378\)](#)
- [适用于 AWS Lambda 的 Identity and Access Management \(p. 379\)](#)
- [AWS Lambda 的合规性验证 \(p. 385\)](#)
- [AWS Lambda 中的弹性 \(p. 386\)](#)
- [AWS Lambda 中的基础设施安全性 \(p. 386\)](#)
- [AWS Lambda 中的配置和漏洞分析 \(p. 386\)](#)

AWS Lambda 中的数据保护

AWS Lambda 符合 AWS 责任共担模型，此模型包含适用于数据保护的法规和准则。AWS 负责保护运行所有 AWS 服务的全球基础设施。AWS 保持对此基础设施上托管的数据的控制，包括用于处理客户内容和个人数据的安全配置控制。充当数据控制者或数据处理者的 AWS 客户和 APN 合作伙伴对他们在 AWS 云中放置的任何个人数据承担责任。

出于数据保护的目的，我们建议您保护 AWS 账户凭证并使用 AWS Identity and Access Management (IAM) 设置单个用户账户，以便仅向每个用户提供履行其工作职责所需的权限。我们还建议您通过以下方式保护您的数据：

- 对每个账户使用 Multi-Factor Authentication (MFA)。
- 使用 SSL/TLS 与 AWS 资源进行通信。
- 使用 AWS CloudTrail 设置 API 和用户活动日志记录。
- 使用 AWS 加密解决方案以及 AWS 服务中的所有默认安全控制。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的个人数据。

我们强烈建议您切勿将敏感的可识别信息（例如您客户的账号）放入自由格式字段或元数据（例如函数名称和标签）。这包括使用控制台、API、AWS CLI 或 AWS 开发工具包处理 Lambda 或其他 AWS 服务时。您输入到元数据的任何数据都可能被选取以包含在诊断日志中。当您向外部服务器提供 URL 时，请勿在 URL 中包含凭证信息来验证您对该服务器的请求。

有关数据保护的更多信息，请参阅 AWS 安全性博客上的 [AWS 责任共担模型和 GDPR 博客文章](#)。

小目录

- [传输中加密 \(p. 379\)](#)

- 静态加密 (p. 379)

传输中加密

Lambda API 终端节点仅支持基于 HTTPS 的安全连接。使用 AWS 管理控制台、AWS 开发工具包或 Lambda API 管理 Lambda 资源时，所有通信都使用传输层安全性 (TLS) 进行加密。

有关 API 终端节点的完整列表，请参阅 AWS General Reference 中的 [AWS 区域和终端节点](#)。

静态加密

您可以使用环境变量安全地存储密钥，以便与 Lambda 函数结合使用。Lambda 始终加密静态环境变量。

此外，您可以使用以下功能来自定义环境变量的加密方式。

- 密钥配置 – 在每个函数级别，您可以将 Lambda 配置为使用您在 AWS Key Management Service 中创建和管理的加密密钥。这些称为客户管理的客户主密钥 (CMK) 或客户托管密钥。如果您未配置客户托管密钥，Lambda 将使用 Lambda 在您的账户中创建的名为 aws/lambda 的 AWS 托管 CMK。
- 加密帮助程序 – 在将环境变量值发送到 Lambda 之前，Lambda 控制台允许您在客户端对其进行加密。这可通过防止在 Lambda 控制台中或 Lambda API 返回的函数配置中以未加密形式显示密钥来进一步增强安全性。控制台还提供了示例代码，您可以调整这些代码来在函数处理程序中对值进行解密。

有关更多信息，请参阅 [使用 AWS Lambda 环境变量 \(p. 49\)](#)。

Lambda 始终对您上传到 Lambda 的文件进行加密，包括 [部署包 \(p. 19\)](#) 和 [层存档 \(p. 68\)](#)。

默认情况下，Amazon CloudWatch Logs 和 AWS X-Ray 也对数据进行加密，并可配置为使用客户托管密钥。有关详细信息，请参阅 [加密 CloudWatch Logs 中的日志数据](#) 和 [AWS X-Ray 中的数据保护](#)。

适用于 AWS Lambda 的 Identity and Access Management

AWS Identity and Access Management (IAM) 是一项 AWS 服务，可帮助管理员安全地控制对 AWS 资源的访问。IAM 管理员控制谁可以通过身份验证（登录）和授权（具有权限）以使用 Lambda 资源。IAM 是一项无需额外费用即可使用的 AWS 服务。

主题

- [受众 \(p. 379\)](#)
- [使用身份进行身份验证 \(p. 380\)](#)
- [使用策略管理访问 \(p. 381\)](#)
- [AWS Lambda 如何与 IAM 协同工作 \(p. 382\)](#)
- [AWS Lambda 基于身份的策略示例 \(p. 382\)](#)
- [故障排除 AWS Lambda 身份和访问 \(p. 384\)](#)

受众

如何使用 AWS Identity and Access Management (IAM) 因您可以在 Lambda 中执行的操作而异。

服务用户 – 如果您使用 Lambda 服务来完成工作，则您的管理员会为您提供所需的凭证和权限。当您使用更多 Lambda 功能来完成工作时，您可能需要额外权限。了解如何管理访问权限可帮助您向管理员请求适合的权限。如果您无法访问 Lambda 中的一项功能，请参阅 [故障排除 AWS Lambda 身份和访问 \(p. 384\)](#)。

服务管理员 – 如果您在公司负责管理 Lambda 资源，则您可能具有 Lambda 的完全访问权限。您有责任确定您的员工应访问哪些 Lambda 功能和资源。然后，您必须向 IAM 管理员提交请求以更改您的服务用户的权限。检查此页上的信息，了解 IAM 的基本概念。要了解有关您的公司如何将 IAM 与 Lambda 搭配使用的更多信息，请参阅[AWS Lambda 如何与 IAM 协同工作 \(p. 382\)](#)。

IAM 管理员 – 如果您是 IAM 管理员，您可能希望了解有关您可以如何编写策略以管理 Lambda 访问权限的详细信息。要查看您可在 IAM 中使用的基于身份的 Lambda 示例策略，请参阅[AWS Lambda 基于身份的策略示例 \(p. 382\)](#)。

使用身份进行身份验证

身份验证是您使用身份凭证登录 AWS 的方法。有关使用 AWS 管理控制台登录的更多信息，请参阅 IAM 用户指南 中的 [IAM 控制台和登录页面](#)。

您必须以 AWS 账户根用户、IAM 用户身份或通过代入 IAM 角色进行身份验证（登录到 AWS）。您还可以使用公司的单一登录身份验证方法，甚至使用 Google 或 Facebook 登录。在这些案例中，您的管理员以前使用 IAM 角色设置了联合身份验证。在您使用来自其他公司的凭证访问 AWS 时，您间接地代入了角色。

要直接登录到 [AWS 管理控制台](#)，请使用您的密码和根用户电子邮件或 IAM 用户名。您可以使用根用户或 IAM 用户访问密钥以编程方式访问 AWS。AWS 提供了开发工具包和命令行工具，可使用您的凭证对您的请求进行加密签名。如果您不使用 AWS 工具，则必须自行对请求签名。使用签名版本 4（用于对入站 API 请求进行验证的协议）完成此操作。有关验证请求的更多信息，请参阅 AWS General Reference 中的[签名版本 4 签名流程](#)。

无论使用何种身份验证方法，您可能还需要提供其他安全信息。例如，AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅 IAM 用户指南 中的[在 AWS 中使用多重身份验证 \(MFA\)](#)。

AWS 账户根用户

当您首次创建 AWS 账户时，最初使用的是一个对账户中所有 AWS 服务和资源有完全访问权限的单点登录身份。此身份称为 AWS 账户根用户，可使用您创建账户时所用的电子邮件地址和密码登录来获得此身份。强烈建议您不使用根用户执行日常任务，即使是管理任务。请遵守[仅将根用户用于创建首个 IAM 用户的最佳实践](#)。然后请妥善保存根用户凭证，仅用它们执行少数账户和服务管理任务。

IAM 用户和群组

[IAM 用户](#) 是 AWS 账户内对某个人员或应用程序具有特定权限的一个身份。IAM 用户可以拥有长期凭证，例如用户名和密码或一组访问密钥。要了解如何生成访问密钥，请参阅 IAM 用户指南 中的[管理 IAM 用户的访问密钥](#)。为 IAM 用户生成访问密钥时，请确保查看并安全保存密钥对。您以后无法找回秘密访问密钥，而是必须生成新的访问密钥对。

[IAM 组](#) 是指定一个 IAM 用户集合的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您有一个名为 IAMAdmins 的组并为该组授予管理 IAM 资源的权限。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅 IAM 用户指南 中的[何时创建 IAM 用户（而不是角色）](#)。

IAM 角色

[IAM 角色](#) 是 AWS 账户中具有特定权限的实体。它类似于 IAM 用户，但未与特定人员关联。您可以通过[切换角色](#)，在 AWS 管理控制台中暂时代入 IAM 角色。您可以调用 AWS CLI 或 AWS API 操作或使用自定义 URL 以代入角色。有关使用角色方法的更多信息，请参阅 IAM 用户指南 中的[使用 IAM 角色](#)。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 临时 IAM 用户权限 – IAM 用户可代入 IAM 角色，暂时获得针对特定任务的不同权限。

- 联合身份用户访问 – 您也可以不创建 IAM 用户，而是使用来自 AWS Directory Service、您的企业用户目录或 Web 身份提供商的现有身份。这些用户被称为联合身份用户。在通过[身份提供商](#)请求访问权限时，AWS 将为联合身份用户分配角色。有关联合身份用户的更多信息，请参阅 IAM 用户指南 中的[联合身份用户和角色](#)。
- 跨账户访问 – 您可以使用 IAM 角色允许其他账户中的某个人（可信任委托人）访问您账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅 IAM 用户指南 中的[IAM 角色与基于资源的策略有何不同](#)。
- AWS 服务访问 – 服务角色是服务代表您在您的账户中执行操作而担任的 IAM 角色。在设置一些 AWS 服务环境时，您必须为服务定义要代入的角色。这个服务角色必须包含该服务访问所需的 AWS 资源会用到的所有必要权限。服务角色因服务而异，但只要您满足服务记录在案的要求，许多服务都允许您选择权限。服务角色只在您的账户内提供访问权限，不能用于为访问其他账户中的服务授权。您可以在 IAM 中创建、修改和删除服务角色。例如，您可以创建一个角色，此角色允许 Amazon Redshift 代表您访问 Amazon S3 存储桶，然后将该存储桶中的数据加载到 Amazon Redshift 集群中。有关更多信息，请参阅 IAM 用户指南 中的[创建角色以向 AWS 服务委派权限](#)。
- 在 Amazon EC2 上运行的应用程序 – 对于在 EC2 实例上运行、并发出 AWS CLI 或 AWS API 请求的应用程序，您可以使用 IAM 角色管理它们的临时凭证。这优先于在 EC2 实例中存储访问密钥。要将 AWS 角色分配给 EC2 实例并使其对该实例的所有应用程序可用，您可以创建一个附加到实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅 IAM 用户指南 中的[使用 IAM 角色向在 Amazon EC2 实例上运行的应用程序授予权限](#)。

要了解是否使用 IAM 角色，请参阅 IAM 用户指南 中的[何时创建 IAM 角色（而不是用户）](#)。

使用策略管理访问

您将创建策略并将其附加到 IAM 身份或 AWS 资源，以便控制 AWS 中的访问。策略是 AWS 中的对象；在与身份或资源相关联时，策略定义它们的权限。在某个实体（根用户、IAM 用户或 IAM 角色）发出请求时，AWS 将评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略在 AWS 中存储为 JSON 文档。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南 中的[JSON 策略概述](#)。

IAM 管理员可以使用策略来指定哪些用户有权访问 AWS 资源，以及他们可以对这些资源执行哪些操作。每个 IAM 实体（用户或角色）一开始都没有权限。换言之，默认情况下，用户什么都不能做，甚至不能更改他们自己的密码。要为用户授予执行某些操作的权限，管理员必须将权限策略附加到用户。或者，管理员可以将用户添加到具有预期权限的组中。当管理员为某个组授予访问权限时，该组内的全部用户都会获得这些访问权限。

IAM 策略定义操作的权限，无论您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。具有该策略的用户可以从 AWS 管理控制台、AWS CLI 或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、角色或组）的 JSON 权限策略文档。这些策略控制身份可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅 IAM 用户指南 中的[创建 IAM 策略](#)。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入单个用户、组或角色中。托管策略是可以附加到 AWS 账户中的多个用户、组和角色的独立策略。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管策略或内联策略之间进行选择，请参阅 IAM 用户指南 中的[在托管策略与内联策略之间进行选择](#)。

基于资源的策略

基于资源的策略是附加到资源（如 Amazon S3 存储桶）的 JSON 策略文档。服务管理员可以使用这些策略来定义指定的委托人（账户成员、用户或角色）可以对该资源以及在什么条件下执行哪些操作。基于资源的策略是内联策略。没有基于托管资源的策略。

访问控制列表 (ACL)

访问控制列表 (ACL) 是一种策略类型，用于控制哪些委托人（账户成员、用户或角色）有权访问资源。ACL 类似于基于资源的策略，但它们不使用 JSON 策略文档格式。Amazon S3、AWS WAF 和 Amazon VPC 是支持 ACL 的服务示例。要了解有关 ACL 的更多信息，请参阅 Amazon Simple Storage Service 开发人员指南中的[访问控制列表 \(ACL\) 概述](#)。

其他策略类型

AWS 支持额外的、不太常用的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界 – 权限边界是一项高级功能，借助该功能，您可以设置基于身份的策略可以授予 IAM 实体的最大权限（IAM 用户或角色）。您可为实体设置权限边界。这些结果权限是实体的基于身份的策略及其权限边界的交集。在 Principal 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅 IAM 用户指南 中的[IAM 实体的权限边界](#)。
- 服务控制策略 (SCP) – SCP 是 JSON 策略，指定了组织或组织单位 (OU) 在 AWS Organizations 中的最大权限。AWS Organizations 是一项服务，用于分组和集中管理您的企业拥有的多个 AWS 账户。如果在组织内启用了所有功能，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中实体（包括每个 AWS 账户根用户）的权限。有关组织和 SCP 的更多信息，请参阅 AWS Organizations 用户指南中的[SCP 工作原理](#)。
- 会话策略 – 会话策略是当您以编程方式为角色或联合身份用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅 IAM 用户指南 中的[会话策略](#)。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解 AWS 如何确定在涉及多个策略类型时是否允许请求，请参阅 IAM 用户指南 中的[策略评估逻辑](#)。

AWS Lambda 如何与 IAM 协同工作

使用 IAM 来管理 Lambda 的访问权限之前，您应该了解哪些 IAM 功能可与 Lambda 协同工作。要概括了解 Lambda 及其他 AWS 服务如何与 IAM 协同工作，请参阅 IAM 用户指南 中的[可与 IAM 协同工作的 AWS 服务](#)。

有关 Lambda 使用的权限、策略和角色的概述，请参阅 [AWS Lambda 权限 \(p. 30\)](#)。

AWS Lambda 基于身份的策略示例

默认情况下，IAM 用户和角色没有创建或修改 Lambda 资源的权限。它们还无法使用 AWS 管理控制台、AWS CLI 或 AWS API 执行任务。IAM 管理员必须创建 IAM 策略，为用户和角色授予权限，以便对他们所需的指定资源执行特定的 API 操作。然后，管理员必须将这些策略附加到需要这些权限的 IAM 用户或组。

要了解如何使用这些示例 JSON 策略文档创建 IAM 基于身份的策略，请参阅 IAM 用户指南 中的[JSON 选项卡上的创建策略](#)。

主题

- [策略最佳实践 \(p. 383\)](#)
- [使用 Lambda 控制台 \(p. 383\)](#)
- [允许用户查看他们自己的权限 \(p. 383\)](#)

策略最佳实践

基于身份的策略非常强大。它们确定某个人是否可以创建、访问或删除您账户中的 Lambda 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下准则和建议：

- 开始使用 AWS 托管策略 – 要快速开始使用 Lambda，请使用 AWS 托管策略，为您的员工授予他们所需的权限。这些策略已在您的账户中提供，并由 AWS 维护和更新。有关更多信息，请参阅 IAM 用户指南 中的[利用 AWS 托管策略开始使用权限](#)。
- 授予最低权限 – 创建自定义策略时，仅授予执行任务所需的许可。最开始只授予最低权限，然后根据需要授予其他权限。这样做比起一开始就授予过于宽松的权限而后再尝试收紧权限来说更为安全。有关更多信息，请参阅 IAM 用户指南 中的[授予最小权限](#)。
- 为敏感操作启用 MFA – 为增强安全性，要求 IAM 用户使用多重身份验证 (MFA) 来访问敏感资源或 API 操作。有关更多信息，请参阅 IAM 用户指南 中的[在 AWS 中使用多重身份验证 \(MFA\)](#)。
- 使用策略条件来增强安全性 – 在切实可行的范围内，定义基于身份的策略在哪些情况下允许访问资源。例如，您可编写条件来指定请求必须来自允许的 IP 地址范围。您也可以编写条件，以便仅允许指定日期或时间范围内的请求，或者要求使用 SSL 或 MFA。有关更多信息，请参阅 IAM 用户指南 中的[IAM JSON 策略元素：Condition](#)。

使用 Lambda 控制台

要访问 AWS Lambda 控制台，您必须拥有一组最低的权限。这些权限必须允许您列出和查看有关您的 AWS 账户中的 Lambda 资源的详细信息。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（IAM 用户或角色），控制台将无法按预期正常运行。

有关授予函数开发的最小访问权限的示例策略，请参阅[函数开发 \(p. 38\)](#)。除了 Lambda API 以外，Lambda 控制台还使用其他服务来显示触发器配置，并允许您添加新触发器。如果您的用户将 Lambda 与其他服务结合使用，则他们还需要这些服务的访问权限。有关配置其他服务及 Lambda 的详细信息，请参阅[将 AWS Lambda 与其他服务结合使用 \(p. 138\)](#)。

允许用户查看他们自己的权限

此示例显示您可以如何创建策略，以便允许 IAM 用户查看附加到其用户身份的内联和托管策略。此策略包括在控制台上完成此操作或者以编程方式使用 AWS CLI 或 AWS API 所需的权限。

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

```
    "iam>ListPolicyVersions",
    "iam>ListPolicies",
    "iam>ListUsers"
],
"Resource": "*"
}
}
```

故障排除 AWS Lambda 身份和访问

使用以下信息可帮助您诊断和修复在使用 Lambda 和 IAM 时可能遇到的常见问题。

主题

- [我无权在 Lambda 中执行操作 \(p. 384\)](#)
- [我无权执行 iam:PassRole \(p. 384\)](#)
- [我想要查看我的访问密钥 \(p. 384\)](#)
- [我是管理员并希望允许其他人访问 Lambda \(p. 385\)](#)
- [我想要允许我的 AWS 账户之外的用户访问我的 Lambda 资源 \(p. 385\)](#)

我无权在 Lambda 中执行操作

如果 AWS 管理控制台 告诉您，您无权执行某个操作，则必须联系您的管理员寻求帮助。您的管理员是指为您提供用户名和密码的那个人。

当 mateojackson IAM 用户尝试使用控制台查看有关函数的详细信息，但不具有 lambda:GetFunction 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
lambda:GetFunction on resource: my-function
```

在这种情况下，Mateo 请求他的管理员更新其策略，以允许他使用 lambda:GetFunction 操作访问 my-function 资源。

我无权执行 iam:PassRole

如果您收到错误消息，提示您无权执行 iam:PassRole 操作，则必须联系您的管理员寻求帮助。您的管理员是指为您提供用户名和密码的那个人。请求那个人更新您的策略，以便允许您将角色传递给 Lambda。

有些 AWS 服务允许您将现有角色传递到该服务，而不是创建新服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 marymajor 的 IAM 用户尝试使用控制台在 Lambda 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

在这种情况下，Mary 请求她的管理员来更新其策略，以允许她执行 iam:PassRole 操作。

我想要查看我的访问密钥

创建 IAM 用户访问密钥之后，您可以随时查看您的访问密钥 ID。但是，您无法再查看您的秘密访问密钥。如果您丢失了私有密钥，则必须创建一个新的访问密钥对。

访问密钥包含两部分：访问密钥 ID（例如 AKIAIOSFODNN7EXAMPLE）和秘密访问密钥（例如 wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY）。与用户名和密码一样，您必须同时使用访问密钥 ID 和秘密访问密钥对请求执行身份验证。像对用户名和密码一样，安全地管理访问密钥。

Important

请不要向第三方提供访问密钥，甚至为了帮助[找到您的规范用户 ID](#)也不能提供。如果您这样做，可能会向某人提供对您的账户的永久访问权限。

当您创建访问密钥对时，系统会提示您将访问密钥 ID 和秘密访问密钥保存在一个安全位置。秘密访问密钥仅在您创建它时可用。如果您丢失了秘密访问密钥，则必须向您的 IAM 用户添加新的访问密钥。您最多可拥有两个访问密钥。如果您已有两个密钥，则必须删除一个密钥对，然后再创建新的密钥。要查看说明，请参阅 IAM 用户指南中的[管理访问密钥](#)。

我是管理员并希望允许其他人访问 Lambda

要允许其他人访问 Lambda，您必须为需要访问权限的人员或应用程序创建 IAM 实体（用户或角色）。他们（它们）将使用该实体的凭证访问 AWS。然后，您必须将策略附加到实体，以便在 Lambda 中为他们（它们）授予正确的权限。

要立即开始使用，请参阅 IAM 用户指南中的[创建您的第一个 IAM 委托用户和组](#)。

我想要允许我的 AWS 账户之外的用户访问我的 Lambda 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表 (ACL) 的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 Lambda 是否支持这些功能，请参阅[AWS Lambda 如何与 IAM 协同工作 \(p. 382\)](#)。
- 要了解如何向您拥有的 AWS 账户中的资源提供访问权限，请参阅 IAM 用户指南中的[对您拥有的 AWS 账户中的 IAM 用户提供访问权限](#)。
- 要了解如何向第三方 AWS 账户提供对您的资源的访问权限，请参阅 IAM 用户指南中的[向第三方拥有的 AWS 账户提供访问权限](#)。
- 要了解如何通过联合身份验证提供访问权限，请参阅 IAM 用户指南中的[向经过外部身份验证的用户 \(联合身份验证\) 提供访问权限](#)。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅 IAM 用户指南中的[IAM 角色和基于资源的策略有何不同](#)。

AWS Lambda 的合规性验证

作为多个 AWS 合规性计划的一部分，第三方审核员将评估 AWS Lambda 的安全性和合规性。其中包括 SOC、PCI、FedRAMP、HIPAA 及其他。

有关特定合规性计划范围内的 AWS 服务的列表，请参阅[合规性计划范围内的 AWS 服务](#)。有关一般信息，请参阅[AWS 合规性计划](#)。

您可以使用 AWS Artifact 下载第三方审计报告。有关更多信息，请参阅[下载 AWS Artifact 中的报告](#)。

您使用 Lambda 的合规性责任取决于您数据的敏感度、贵公司的合规性目标以及适用的法律法规。AWS 提供以下资源来帮助满足合规性：

- [安全性与合规性快速入门指南](#) – 这些部署指南讨论了架构注意事项，并提供了在 AWS 上部署基于安全性和合规性的基准环境的步骤。
- [《设计符合 HIPAA 安全性和合规性要求的架构》白皮书](#) – 此白皮书介绍公司如何使用 AWS 创建符合 HIPAA 标准的应用程序。

- [AWS 合规性资源](#) – 此业务手册和指南集合可能适用于您的行业和位置。
- [AWS Config](#) – 此 AWS 服务评估您的资源配置对内部实践、行业指南和法规的遵循情况。
- [AWS Security Hub](#) – 此 AWS 服务提供了 AWS 中安全状态的全面视图，可帮助您检查是否符合安全行业标准和最佳实践。

AWS Lambda 中的弹性

AWS 全球基础设施围绕 AWS 区域和可用区构建。AWS 区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅 [AWS 全球基础设施](#)。

除了 AWS 全球基础设施之外，Lambda 还提供了多种功能，以帮助支持您的数据弹性和备份需求。

- 版本控制 – 您可以在 Lambda 中使用版本控制，以便在开发时保存函数的代码和配置。与别名相配合，您可以使用版本控制来执行蓝/绿和滚动部署。有关详细信息，请参阅[AWS Lambda 函数版本 \(p. 63\)](#)。
- 扩展 – 当您的函数在处理请求时收到新的请求时，Lambda 会启动一个新的函数实例来处理增加的负载。Lambda 自动扩展以处理每个区域的 1,000 个并发执行（可根据需要提高该[限制 \(p. 27\)](#)）。有关详细信息，请参阅[AWS Lambda 函数扩展 \(p. 94\)](#)。
- 高可用性 – Lambda 在多个可用区中运行您的函数，以确保在单一区域中服务中断时能够处理事件。如果将函数配置为连接到账户中的 Virtual Private Cloud (VPC)，请在多个可用区中指定子网以确保高可用性。有关详细信息，请参阅[配置 Lambda 函数以访问 VPC 中的资源 \(p. 72\)](#)。
- 预留并发 – 要确保您的函数能够始终扩展以处理更多请求，您可以为其预留并发。为函数设置预留并发可确保其能够扩展（但不超出）指定数量的并发调用。这可确保您不会因为其他函数使用了所有可用的并发而丢失请求。有关详细信息，请参阅[管理 Lambda 函数的并发 \(p. 54\)](#)。
- 重试 – 对于异步调用和由其他服务触发的调用子集，Lambda 会在遇到错误时自动重试（重试之间有延迟）。同步调用函数的其他客户端和 AWS 服务负责执行重试。有关详细信息，请参阅[AWS Lambda 中的错误处理和自动重试 \(p. 98\)](#)。
- 死信队列 – 对于异步调用，如果所有重试都失败，您可以配置 Lambda 将请求发送到死信队列。死信队列是 Amazon SNS 主题或 Amazon SQS 队列，它接收事件以进行故障排除或重新处理。有关详细信息，请参阅 [AWS Lambda 函数死信队列 \(p. 88\)](#)。

AWS Lambda 中的基础设施安全性

作为一项托管服务，AWS Lambda 由 [Amazon Web Services：安全流程概述](#) 白皮书中所述的 AWS 全球网络安全程序提供保护。

您可以使用 AWS 发布的 API 调用通过网络访问 Lambda。客户端必须支持传输层安全性 (TLS) 1.0 或更高版本。建议使用 TLS 1.2 或更高版本。客户端还必须支持具有完全向前保密 (PFS) 的密码套件，例如 Ephemeral Diffie-Hellman (DHE) 或 Elliptic Curve Ephemeral Diffie-Hellman (ECDHE)。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 委托人关联的秘密访问密钥来对请求进行签名。或者，您可以使用 [AWS Security Token Service](#) (AWS STS) 生成临时安全凭证来对请求进行签名。

AWS Lambda 中的配置和漏洞分析

AWS Lambda 提供在基于 Amazon Linux 的执行环境中运行您的函数代码的[运行时 \(p. 108\)](#)。Lambda 负责确保运行时和执行环境中的软件保持最新状态、发布适用于新语言和框架的新运行时和在不再支持底层软件时弃用运行时。

如果您的函数使用了其他的库，则由您负责更新这些库。您可以在[部署包 \(p. 19\)](#)或附加到函数的层 (p. 68)中包含其他库。您还可以构建[自定义运行时 \(p. 111\)](#)，并使用层与其他账户共享它们。

当运行时上的软件或其执行环境的使用寿命结束时，Lambda 将弃用运行时。当 Lambda 弃用运行时时，您负责将自己的函数迁移到相同语言或框架的受支持运行时。有关详细信息，请参阅[运行时支持策略 \(p. 110\)](#)。

排查 AWS Lambda 中的问题

以下主题为您在使用 Lambda API、控制台或工具时可能遇到的错误和问题提供故障排除建议。如果您发现某个问题未在此处列出，可以使用此页上的 Feedback 按钮来报告。

有关更多故障排除建议和常见支持问题的答案，请访问 [AWS 知识中心](#)。

主题

- [解决 AWS Lambda 中的部署问题 \(p. 388\)](#)
- [AWS Lambda 中的调用问题疑难解答 \(p. 390\)](#)
- [AWS Lambda 中的执行问题疑难解答 \(p. 391\)](#)
- [AWS Lambda 中的联网问题疑难解答 \(p. 392\)](#)

解决 AWS Lambda 中的部署问题

更新函数时，Lambda 使用更新的代码或设置来启动函数的新实例，从而部署更改。部署错误会阻止新版本的使用，并可能由于部署程序包、代码、权限或工具中的问题而引起。

当您使用 Lambda API 或客户端（如 AWS CLI）将更新直接部署到函数时，您可以直接在输出中查看 Lambda 中的错误。如果您使用 AWS CloudFormation、AWS CodeDeploy 或 AWS CodePipeline 等服务，请在该服务的日志或事件流中查找来自 Lambda 的响应。

错误：EACCES: permission denied, open '/var/task/index.js' (EACCES: 权限被拒绝，打开“/var/task/index.js”)

错误：cannot load such file -- function (无法加载此文件 -- 函数)

错误：[Errno 13] Permission denied: '/var/task/function.py' ([Errno 13] 权限被拒绝：“/var/task/function.py”)

Lambda 运行时需要权限才能读取部署包中的文件。您可以使用 `chmod` 命令更改文件模式。以下示例命令使当前目录中的所有文件和文件夹都可供任何用户使用。

```
my-function$ chmod 644 $(find . -type f)
my-function$ chmod 755 $(find . -type d)
```

错误：An error occurred (RequestEntityTooLargeException) when calling the UpdateFunctionCode operation (在调用 UpdateFunctionCode 操作时出错 (RequestEntityTooLargeException))

在将部署包或图层存档直接上传到 Lambda 时，ZIP 文件的大小最多为 50 MB。要上传一个较大的文件，请将此文件存储在 Amazon S3 中并使用 [S3Bucket 和 S3Key \(p. 554\)](#) 参数。

Note

当您直接使用 AWS CLI、AWS 开发工具包或通过其他方式上传文件时，二进制 ZIP 文件将转换为 base64，其大小将增加约 30%。为了支持这一点以及请求中其他参数的大小，Lambda 应用的实际请求大小限制会更大。因此，50 MB 的限制是近似值。

错误：GetObject 时发生错误。S3 错误代码：PermanentRedirect。S3 错误消息：存储桶位于此区域中：us-east-2。请使用此区域重试请求

当您从 Amazon S3 存储桶上传函数的部署包时，存储桶必须与函数位于同一区域。当您在 [UpdateFunctionCode \(p. 553\)](#) 调用中指定 Amazon S3 对象或在 AWS CLI 或 AWS SAM CLI 中使用包并部署命令时，可能会出现此问题。为您在其中开发应用程序的每个区域创建部署构件存储桶。

错误：找不到模块“function”

错误：cannot load such file -- function (无法加载此文件 -- 函数)

错误：无法导入模块“function”

错误：找不到类：function.Handler

错误：fork/exec/var/task/function：没有这样的文件或目录

错误：无法从程序集“Function”加载类型“Function.Handler”。

函数处理程序配置中文件或类的名称与您的代码不匹配。有关更多信息，请参阅以下条目。

错误：index.handler 未定义或未导出

错误：模块“function”上缺少处理程序“handler”

错误：#<LambdaHandler:0x000055b76ccef98> 的未定义的方法“handler”

错误：在类函数 function.Handler 上没有找到具有正确方法签名的名为 handleRequest 的公共方法。

错误：无法从程序集“Function”的类型“Function.Handler”中找到方法“handleRequest”

函数处理程序配置中的处理程序方法的名称与您的代码不匹配。每个运行时为处理程序定义一个命名约定，例如 *filename.methodname*。处理程序是函数代码中的方法，在调用函数时由运行时执行该方法。

对于某些语言，Lambda 提供了一个包含一个接口的库，该接口需要具有特定名称的处理程序方法。有关每种语言的处理程序命名的详细信息，请参阅以下主题。

- 使用 Node.js 构建 Lambda 函数 (p. 265)
- 使用 Python 构建 Lambda 函数 (p. 277)
- 使用 Ruby 构建 Lambda 函数 (p. 291)
- 使用 Java 构建 Lambda 函数 (p. 301)
- 使用 Go 构建 Lambda 函数 (p. 330)
- 使用 C# 构建 Lambda 函数 (p. 342)
- 使用 PowerShell 构建 Lambda 函数 (p. 359)

错误：InvalidParameterValueException：Lambda 无法配置您的环境变量，因为您提供的环境变量超过了 4KB 限制。测量的字符串：{"A1":"uSFeY5cyPiPn7AtnX5BsM...

错误：RequestEntityTooLargeException：对 UpdateFunctionConfiguration 操作的请求必须小于 5120 字节

存储在函数配置中的变量对象的最大大小不得超过 4096 个字节。这包括密钥名称、值、引号、逗号和括号。HTTP 请求正文的总大小也受到限制。

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

在此示例中，对象为 39 个字符，并且它在存储（不含空格）为字符串 `{"BUCKET": "my-bucket", "KEY": "file.txt"}` 时占用 39 个字节。环境变量值中的每个标准 ASCII 字符使用一个字节。每个扩展 ASCII 和 Unicode 字符可以使用 2 到 4 个字节。

错误：`InvalidParameterValueException`: Lambda 无法配置您的环境变量，因为您提供的环境变量包含目前不支持修改的保留键。

Lambda 保留一些环境变量键供内部使用。例如，运行时使用 `AWS_REGION` 来确定当前区域且它不能被覆盖。运行时使用其他变量（如 `PATH`），但这些变量可以在函数配置中扩展。有关完整列表，请参阅[运行时环境变量 \(p. 51\)](#)。

AWS Lambda 中的调用问题疑难解答

当您调用 Lambda 函数时，Lambda 在将事件发送到函数或（对于异步调用）发送到事件队列之前，先验证请求并检查扩展容量。调用错误可能是由请求参数、事件结构、函数设置、用户权限、资源权限或限制中的问题引起的。

如果您直接调用函数，则会在 Lambda 的响应中看到调用错误。如果您使用事件源映射或通过其他服务异步调用函数，则可能会在日志、死信队列或失败事件目标中找到错误。错误处理选项和重试行为因调用函数的方式和错误类型而异。

有关 `Invoke` 操作可以返回的错误类型的列表，请参阅[Invoke \(p. 482\)](#)。

错误：`User: arn:aws:iam::123456789012:user/developer` is not authorized to perform: `lambda:InvokeFunction` on resource: `my-function` (用户: `arn:aws:iam::123456789012:user/developer` 未获得授权，无法对资源执行 `lambda:InvokeFunction: my-function`)

您的 IAM 用户或您代入的角色需要权限才能调用函数。此要求还适用于 Lambda 函数以及其他调用函数的计算资源。将 `AWSLambdaRole` 托管策略或允许对目标函数执行 `lambda:InvokeFunction` 操作的自定义策略添加到 IAM 用户。

Note

与 Lambda 中的其他 API 操作不同，IAM (`lambda:InvokeFunction`) 中的操作名称不匹配用于调用函数的 API 操作 (`Invoke`) 的名称。

有关更多信息，请参阅[AWS Lambda 权限 \(p. 30\)](#)。

错误：`ResourceConflictException`：此时无法执行该操作。该函数目前处于以下状态：待定

如果您在创建时将函数连接到 VPC，则该函数会在 Lambda 创建弹性网络接口时进入 `Pending` 状态。在此期间，您无法调用或修改函数。如果您在创建后将函数连接到 VPC，则可以在更新处于待处理状态时调用该函数，但无法修改其代码或配置。

有关更多信息，请参阅[使用 Lambda API 监控函数的状态 \(p. 92\)](#)。

错误：一个函数卡在 `Pending` 状态达几分钟。

如果某个函数卡在 `Pending` 状态超过六分钟，请调用以下 API 操作之一来取消阻止它。

- [UpdateFunctionCode \(p. 553\)](#)
- [UpdateFunctionConfiguration \(p. 560\)](#)
- [PublishVersion \(p. 518\)](#)

Lambda 取消处于暂停状态的操作并将函数置于 `Failed` 状态。然后，您可以删除该函数并重新创建它，或尝试其他更新。

问题：一个函数使用了所有可用的并发，导致其他函数被限制。

要将某个区域中的可用并发划分为池，请使用[预留并发 \(p. 54\)](#)。预留并发可确保函数始终可以扩展到向其分配的并发，并且不会扩展到向其分配的并发之外。

问题：您可以直接调用您的函数，但是当另一个服务或账户调用它时，它不会运行。

您向[其他服务 \(p. 138\)](#)和账户授予权限以在函数的[基于资源的策略 \(p. 33\)](#)中调用函数。如果调用方是另一个账户，则该用户还需要[函数调用权限 \(p. 37\)](#)。

问题：函数在循环中被连续调用。

当您的函数在触发它的同一 AWS 服务中管理资源时，通常会发生这种情况。例如，可以创建一个函数来将对象存储在 Amazon S3 存储桶中，该存储桶配置了一个[再次调用函数的通知 \(p. 233\)](#)。要使函数停止运行，请在[函数配置页面 \(p. 47\)](#)上选择 Throttle（限制）。然后，确定导致递归调用的代码路径或配置错误。

错误：KMSDisabledException : Lambda 无法解密环境变量，因为使用的 KMS 密钥已被禁用。请检查功能的 KMS 密钥设置。

如果您的 KMS 密钥被禁用或者撤销了允许 Lambda 使用该密钥的授权，则可能会发生此错误。如果缺少授权，请将函数配置为使用其他密钥。然后，重新分配自定义密钥以重新创建授权。

AWS Lambda 中的执行问题疑难解答

当 Lambda 运行时执行函数代码时，可能会在已经处理事件一段时间的函数实例上处理事件，或者可能需要初始化一个新实例。在函数初始化期间、处理程序代码处理事件时或者当函数返回（或无法返回）响应时，可能会发生错误。

函数执行错误可能是由您的代码、函数配置、下游资源或权限中的问题引起。如果您直接调用函数，则会在 Lambda 的响应中看到函数错误。如果您使用事件源映射或通过其他服务异步调用函数，则可能会在日志、死信队列或失败时的目标中找到错误。错误处理选项和重试行为因调用函数的方式和错误类型而异。

当您的函数代码或 Lambda 运行时返回错误时，来自 Lambda 的响应中的状态代码为“200 OK”。响应中是否存在错误由名为 X-Amz-Function-Error 的标头指示。400 和 500 系列状态代码保留用于[调用错误 \(p. 390\)](#)。

问题：函数执行时间太长。

如果您的代码在 Lambda 中运行所花费的时间长于在本地计算机上运行所花费的时间，则可能受到对该函数可用的内存或处理能力的限制。[使用额外内存配置函数 \(p. 47\)](#)以增加内存和 CPU。

问题：日志未显示在 CloudWatch Logs 中。

问题：跟踪未显示在 AWS X-Ray 中。

您的函数需要权限才能调用 CloudWatch Logs 和 X-Ray。更新函数的[执行角色 \(p. 30\)](#)以向其授予权限。添加以下托管策略以启用日志和跟踪。

- AWSLambdaBasicExecutionRole
- AWSXRayDaemonWriteAccess

当您向函数添加权限时，也要更新其代码或配置。这将强制停止并替换正在运行的带过期凭证的函数实例。

问题：(Node.js) 函数在代码完成执行之前返回

许多库（包括 AWS 开发工具包）都是异步操作。当您进行网络调用或执行其他需要等待响应的操作时，库返回一个名为 promise 的对象，该对象在后台跟踪操作的进度。

要等待 promise 解析为响应，请使用 `await` 关键字。这会阻止您的处理程序代码执行，直到将 promise 解析为包含响应的对象。如果您不需要在代码中使用来自响应的数据，则可以直接将 promise 返回到运行时。

一些库不返回 promise，但可以包装到返回 promise 的代码中。有关更多信息，请参阅[Node.js 中的 AWS Lambda 函数处理程序 \(p. 266\)](#)。

问题：运行时包含的 AWS 开发工具包不是最新版本

问题：运行时中包含的 AWS 开发工具包自动更新

脚本语言的运行时包括 AWS 开发工具包，并定期更新到最新版本。每个运行时的当前版本在[运行时页面 \(p. 108\)](#)上列出。要使用较新版本的 AWS 开发工具包，或将函数锁定为特定版本，您可以将库与函数代码捆绑起来，或[创建 Lambda 层 \(p. 68\)](#)。有关创建具有依赖项的部署程序包的详细信息，请参阅以下主题：

- [Node.js 中的 AWS Lambda 部署程序包 \(p. 268\)](#)
- [Python 中的 AWS Lambda 部署程序包 \(p. 279\)](#)
- [Ruby 中的 AWS Lambda 部署程序包 \(p. 293\)](#)
- [Java 中的 AWS Lambda 部署程序包 \(p. 302\)](#)
- [Go 中的 AWS Lambda 部署程序包 \(p. 330\)](#)
- [C# 中的 AWS Lambda 部署程序包 \(p. 342\)](#)
- [PowerShell 中的 AWS Lambda 部署程序包 \(p. 360\)](#)

问题：(Python) 无法从部署程序包中正确加载某些库

具有使用 C 或 C ++ 编写的扩展模块的库，必须在与 Lambda (Amazon Linux) 具有相同处理器体系结构的环境中进行编译。有关更多信息，请参阅[Python 中的 AWS Lambda 部署程序包 \(p. 279\)](#)。

AWS Lambda 中的联网问题疑难解答

默认情况下，Lambda 在具有与 AWS 服务和 Internet 连接的内部 Virtual Private Cloud (VPC) 中运行您的函数。要访问本地网络资源，您可以将[函数配置为连接到您账户中的 VPC \(p. 72\)](#)。使用此功能时，您可以管理函数的 Internet 访问以及与 VPC 资源的网络连接。

网络连接错误可能是由于路由配置、安全组规则、角色权限、网络地址转换或资源可用性（例如 IP 地址或网络接口）中的问题造成。它们可能会导致特定错误，或者如果请求无法到达目标，则会导致超时。

问题：连接到 VPC 后，函数失去 Internet 访问权限

错误：错误：连接 ETIMEDOUT 176.32.98.189:443

错误：错误：任务在 10.00 秒后超时

当您将函数连接到 VPC 时，所有出站请求都会通过您的 VPC。要连接到 Internet，请将 VPC 配置为将出站流量从函数的子网发送到公有子网中的 NAT 网关。有关更多信息和示例 VPC 配置，请参阅[对 VPC 连接函数的 Internet 和服务访问 \(p. 74\)](#)。

问题：函数需要在不使用 Internet 的情况下访问 AWS 服务

要从没有 Internet 访问权限的私有子网连接到 AWS 服务，请使用 VPC 终端节点。有关包含 DynamoDB 和 Amazon S3 的 VPC 终端节点的示例模板，请参阅[??? \(p. 74\)](#)。

错误：ENILimitReachedException：函数的 VPC 已达到弹性网络接口限制。

将函数连接到 VPC 时，Lambda 为附加到函数的每个子网和安全组的组合创建一个弹性网络接口。这些网络接口限制为每个 VPC 250，但可以增加此限制。要请求提高限制，请使用[支持中心控制台](#)。

AWS Lambda 版本

下表介绍了 2018 年 5 月之后对 AWS Lambda 开发人员指南的重要更改。如需有关文档更新的通知，您可以订阅 [RSS 源](#)。

update-history-change	update-history-description	update-history-date
在 AWS Lambda 中支持 .NET Core 3.1.0 运行时	AWS Lambda 现在支持 .NET Core 3.1.0 运行时。有关详细信息，请参阅 .NET Core CLI 。	March 31, 2020
对 API 网关 HTTP API 的支持	更新和扩展了将 Lambda 与 API 网关 结合使用的文档，包括对 HTTP API 的支持。添加了使用 AWS CloudFormation 创建 API 和函数的示例应用程序。有关详细信息，请参阅 将 AWS Lambda 与 Amazon API Gateway 结合使用 。	March 23, 2020
Ruby 2.7	已提供适用于 Ruby 2.7 的新运行时 ruby2.7，它是第一个使用 Amazon Linux 2 的 Ruby 运行时。有关详细信息，请参阅 使用 Ruby 构建 Lambda 函数 。	February 19, 2020
并发指标	AWS Lambda 现在报告所有函数、别名和版本的 ConcurrentExecutions 指标。您可以在函数的监控页面上查看此指标的图表。以前，仅在账户级别和为使用预留并发的函数报告 ConcurrentExecutions。有关详细信息，请参阅 AWS Lambda 函数指标 。	February 18, 2020
更新功能状态	默认情况下，现在对所有函数强制执行函数状态。当您将函数连接到 VPC 时，Lambda 会创建共享的弹性网络接口。这样，您的函数就可以在不创建额外的网络接口的情况下向上扩展。在此期间，您无法对函数执行其他操作，包括更新其配置和发布版本。在某些情况下，调用也会受到影响。有关函数的当前状态的详细信息可从 Lambda API 中获取。 此更新正在分阶段发布。有关详细信息，请参阅 AWS 计算博客上的 VPC 网络的已更新 Lambda 状态生命周期 。有关状态的更多信息，请参阅 AWS Lambda 函数状态 。	January 24, 2020

对函数配置 API 输出的更新	对于连接到 VPC 的函数，将原因代码添加到 StateReasonCode (InvalidSubnet、InvalidSecurityGroup) 和 LastUpdateStatusReasonCode (SubnetOutOfIPAddresses、InvalidSubnet、InvalidLastUpdateStatus)。有关状态的更多信息，请参阅 AWS Lambda 函数状态 。	January 20, 2020
预配置并发	现在，您可为函数版本或别名分配预配置并发。预配置并发使函数能够在延迟不发生波动的情况下进行扩展。有关详细信息，请参阅 管理 Lambda 函数的并发 。	December 3, 2019
创建数据库代理	现在，您可以使用 Lambda 控制台为 Lambda 函数创建数据库代理。数据库代理使函数能够在不耗尽数据库连接的情况下达到高并发级别。有关详细信息，请参阅 配置 Lambda 函数的数据库访问 。	December 3, 2019
对持续时间指标的百分位支持	现在，您可以基于百分位筛选持续时间指标。有关详细信息，请参阅 AWS Lambda 指标 。	November 26, 2019
流事件源的错误处理	从流读取的事件源映射可以使新的配置选项。您可以配置 DynamoDB 流 和 Kinesis 流 事件源映射，以便限制重试并设置最长记录期限。出现错误时，您可以将事件源映射配置为在重试之前拆分批次，并将失败批次的调用记录发送到队列或主题。有关详细信息，请参阅 AWS Lambda 事件源映射 。	November 25, 2019
异步调用目标	现在，您可以配置 Lambda 以将异步调用记录发送给另一个服务。调用记录包含有关事件、上下文和函数响应的详细信息。您可以将调用记录发送到 SQS 队列、SNS 主题、Lambda 函数或 EventBridge 事件总线。有关详细信息，请参阅 配置异步调用目标 。	November 25, 2019
异步调用的错误处理选项	异步调用可以使用新的配置选项。您可以配置 Lambda 以限制重试并设置最长事件期限。有关详细信息，请参阅 配置异步调用的错误处理 。	November 25, 2019
增加流事件源的并发	DynamoDB 流 和 Kinesis 流 事件源映射的新选项使您能够一次处理每个分片中的多个批次。当您增加每个分片的并发批次数时，函数的并发最多可以是流中分片数的 10 倍。有关详细信息，请参阅 AWS Lambda 事件源映射 。	November 25, 2019

函数状态	当您创建或更新函数时，它会进入挂起状态，同时 Lambda 会预配置资源来为其提供支持。如果您将函数连接到 VPC，Lambda 可以立即创建共享的弹性网络接口，而不是在调用函数时创建网络接口。这将为连接 VPC 的函数带来更好的性能，但可能需要更新您的自动化。有关详细信息，请参阅 AWS Lambda 函数状态 。	November 25, 2019
Node.js、Python 和 Java 的新运行时	新运行时可用于 Node.js 12、Python 3.8 和 Java 11。有关详细信息，请参阅 AWS Lambda 运行时 。	November 18, 2019
Amazon SQS 事件源的 FIFO 队列支持	现在，您可以创建从先进先出 (FIFO) 队列读取的事件源映射。以前只支持标准队列。有关详细信息，请参阅 将 AWS Lambda 与 Amazon SQS 结合使用 。	November 18, 2019
在 Lambda 控制台中创建应用程序	现在可在 Lambda 控制台中创建应用程序。有关说明，请参阅 在 Lambda 控制台中创建具有持续交付功能的应用程序 。	October 31, 2019
在 Lambda 控制台中创建应用程序（测试版）	现在，您可以在 Lambda 控制台中创建具有集成式持续交付管道的 Lambda 应用程序。该控制台提供了示例应用程序，可将这些应用程序用作您自己的项目的起点。选择 AWS CodeCommit 或 GitHub 来实施源代码控制。每次将更改推送到存储库时，包含的管道都会自动构建并部署它们。有关说明，请参阅 在 Lambda 控制台中创建具有持续交付功能的应用程序 。	October 3, 2019
针对 VPC 连接函数的性能改进	Lambda 现在使用一种新的弹性网络接口，该接口由虚拟私有云 (VPC) 子网中的所有函数共享。当您将一个函数连接到 VPC 时，Lambda 为每个所选安全组和子网组合创建一个网络接口。当共享网络接口可用时，此函数在扩展时不再需要创建其他网络接口。这大大缩短了启动时间。有关详细信息，请参阅 配置 Lambda 函数以访问 VPC 中的资源 。	September 3, 2019

流批量设置	现在，您可以为 Amazon DynamoDB 和 Amazon Kinesis 事件源映射配置一个批处理时间窗口。配置一个最高 5 分钟批处理时间窗口以便在完整批处理可用之前对传入的记录进行缓存。这可以在流处于不活动状态时减少您函数的调用次数。	August 29, 2019
CloudWatch Logs Insights 集成	Lambda 控制台中的监控页面现在包括来自 Amazon CloudWatch Logs Insights 的报告。有关详细信息，请参阅 在 AWS Lambda 控制台中监控函数 。	June 18, 2019
Amazon Linux 2018.03	Lambda 执行环境正在更新为使用 Amazon Linux 2018.03。有关详细信息，请参阅 执行环境 。	May 21, 2019
Node.js 10	针对 Node.js 10 和 nodejs10.x 提供了新的运行时。此运行时使用 Node.js 10.15，并将定期使用 Node.js 10 的最新小版本进行更新。Node.js 10 也是可使用 Amazon Linux 2 的首个运行时。有关详细信息，请参阅 使用 Node.js 构建 Lambda 函数 。	May 13, 2019
GetLayerVersionByArn API	使用 GetLayerVersionByArn API 下载层版本信息，使用版本 ARN 作为输入。与 GetLayerVersion 相比，GetLayerVersionByArn 让您可以直接使用 ARN 而不是将其解析以获取层名称和版本号。	April 25, 2019
自定义运行时层	构建自定义运行时以采用您的常用编程语言运行 Lambda 函数。有关详细信息，请参阅 自定义 AWS Lambda 运行时 。	November 29, 2018
Ruby	利用 Lambda 层，您可以从您的函数代码单独打包并部署库、自定义运行时及其他依赖项。与其他账户或在全球范围内共享您的层。有关详细信息，请参阅 AWS Lambda 层 。	November 29, 2018
Application Load Balancer 触发器	Elastic Load Balancing 现在支持 Lambda 函数作为 Application Load Balancer 的目标。有关详细信息，请参阅 将 Lambda 与 Application Load Balancer 结合使用 。	November 29, 2018

使用 Kinesis HTTP/2 流使用者作为触发器	您可以使用 Kinesis HTTP/2 数据流使用者将事件发送到 AWS Lambda。流使用者具有来自数据流中每个分片的专用读取吞吐量，并使用 HTTP/2 来最大程度地降低延迟。有关详细信息，请参阅 将 AWS Lambda 与 Kinesis 结合使用 。	November 19, 2018
Python 3.7	AWS Lambda 现在通过一个新运行时支持 Python 3.7。有关更多信息，请参阅 使用 Python 构建 Lambda 函数 。	November 19, 2018
异步函数调用的负载限制提高	异步调用的最大负载大小从 128 KB 增加到 256 KB，这与 Amazon SNS 触发器的最大消息大小相匹配。有关详细信息，请参阅 AWS Lambda 限制 。	November 16, 2018
AWS GovCloud (美国东部) 区域	AWS Lambda 现已在 AWS GovCloud (美国东部) 区域提供。有关详细信息，请参阅 AWS 博客上的 AWS GovCloud (美国东部) 现已开放 。	November 12, 2018
已将 AWS SAM 主题移至单独的开发人员指南	许多主题都重点说明了如何使用 AWS 无服务器应用程序模型 (AWS SAM) 构建无服务器应用程序。这些主题已移至 AWS 无服务器应用程序模型 开发人员指南 。	October 25, 2018
在控制台中查看 Lambda 应用程序	您可以在 Lambda 控制台中的 应用程序 页面上查看 Lambda 应用程序的状态。此页面显示了 AWS CloudFormation 堆栈的状态。它包括页面的链接，您可以在这些页面中查看堆栈资源的更多信息。您还可以查看应用程序的聚合指标并创建自定义监控控制面板。	October 11, 2018
函数执行超时限制	要允许长时间运行的函数，最大可配置执行超时从 5 分钟增加到 15 分钟。有关详细信息，请参阅 AWS Lambda 限制 。	October 10, 2018
AWS Lambda 支持 PowerShell Core 语言	AWS Lambda 现在支持 PowerShell Core 语言。有关更多信息，请参阅 在 PowerShell 中编写 Lambda 函数的编程模型 。	September 11, 2018
支持 AWS Lambda 中的 .NET Core 2.1.0 运行时	AWS Lambda 现在支持 .NET Core 2.1.0 运行时。有关更多信息，请参阅 .NET Core CLI 。	July 9, 2018
现在可通过 RSS 更新	现在您可以订阅 RSS 源来接收有关 AWS Lambda 开发人员指南的通知。	July 5, 2018

支持 Amazon SQS 作为事件源	AWS Lambda 现在支持 Amazon Simple Queue Service (Amazon SQS) 作为事件源。有关更多信息，请参阅 调用 Lambda 函数 。	June 28, 2018
中国 (宁夏) 区域	AWS Lambda 现于中国 (宁夏) 区域中可用。有关 Lambda 区域和终端节点的更多信息，请参阅 AWS General Reference 中的 区域和终端节点 。	June 28, 2018

早期更新

下表描述 2018 年 6 月之前发布的每个 AWS Lambda 开发人员指南中的重要变化。

更改	描述	日期
Node.js 运行时 8.10 的运行时支持	AWS Lambda 现在支持 Node.js 运行时 v8.10。有关更多信息，请参阅 使用 Node.js 构建 Lambda 函数 (p. 265) 。	2018 年 4 月 2 日
函数和别名修订 ID	AWS Lambda 现在支持您的函数版本和别名上的修订 ID。当您更新您的函数版本或别名资源时，您可以使用这些 ID 跟踪和应用条件更新。	2018 年 1 月 25 日
对 Go 和 .NET 2.0 的运行时支持	AWS Lambda 增加了对 Go 和 .NET 2.0 的运行时支持。有关更多信息，请参阅 使用 Go 构建 Lambda 函数 (p. 330) 和 使用 C# 构建 Lambda 函数 (p. 342) 。	2018 年 1 月 15 日
控制台再设计	AWS Lambda 引入了一个新的 Lambda 控制台以简化您的体验，并添加了一个 Cloud9 代码编辑器以使您能够更好地调试和修改函数代码。有关更多信息，请参阅 使用 AWS Lambda 控制台编辑器创建函数 (p. 5) 。	2017 年 11 月 30 日
设置单个函数的并发限制	AWS Lambda 现在支持设置单个函数的并发限制。有关更多信息，请参阅 管理 Lambda 函数的并发 (p. 54) 。	2017 年 11 月 30 日
使用别名转移流量	AWS Lambda 现在支持使用别名转移流量。有关更多信息，请参阅 Lambda 函数的滚动部署 (p. 132) 。	2017 年 11 月 28 日
逐步代码部署	AWS Lambda 现在支持通过使用 Code Deploy 安全部署新版本的 Lambda 函数。有关更多信息，请参阅 逐步代码部署 。	2017 年 11 月 28 日
中国 (北京) 区域	AWS Lambda 现于中国 (北京) 区域中可用。有关 Lambda 区域和终端节点的更多信息，请参阅 AWS General Reference 中的 区域和终端节点 。	2017 年 11 月 9 日
推出 SAM Local	AWS Lambda 推出 SAM Local (现在叫做 SAM CLI)，这是一种 AWS CLI 工具，在将无服务应用程序上传到 Lambda 运行时前，为您提供在本地开发、测试和分析它们的环境。有关更多信息，请参阅 测试和调试无服务器应用程序 。	2017 年 8 月 11 日
加拿大 (中部) 区域	AWS Lambda 现于加拿大 (中部) 区域中可用。有关 Lambda 区域和终端节点的更多信息，请参阅 AWS General Reference 中的 区域和终端节点 。	2017 年 6 月 22 日

更改	描述	日期
南美洲 (圣保罗) 区域	AWS Lambda 现于南美洲 (圣保罗) 区域中可用。有关 Lambda 区域和终端节点的更多信息 , 请参阅 AWS General Reference 中的 区域和终端节点 。	2017 年 6 月 6 日
AWS Lambda 支持 AWS X-Ray。	Lambda 引入了对 X-Ray 的支持 , 这样您就可以通过 Lambda 应用程序检测、分析和优化性能问题。有关更多信息 , 请参阅 使用 AWS X-Ray (p. 371) 。	2017 年 4 月 19 日
亚太地区 (孟买) 区域	AWS Lambda 现于亚太地区 (孟买) 区域中可用。有关 Lambda 区域和终端节点的更多信息 , 请参阅 AWS General Reference 中的 区域和终端节点 。	2017 年 3 月 28 日
AWS Lambda 现在支持 Node.js 运行时 v6.10	AWS Lambda 增加了对 Node.js 运行时 v6.10 的支持。有关更多信息 , 请参阅 使用 Node.js 构建 Lambda 函数 (p. 265) 。	2017 年 3 月 22 日
欧洲 (伦敦) 区域	AWS Lambda 现于欧洲 (伦敦) 区域中可用。有关 Lambda 区域和终端节点的更多信息 , 请参阅 AWS General Reference 中的 区域和终端节点 。	2017 年 2 月 1 日
AWS Lambda 支持 .NET 运行时、Lambda@Edge (预览版) 、死信队列和无服务器应用程序自动部署。	<p>AWS Lambda 增加对 C# 的支持。有关更多信息 , 请参阅 使用 C# 构建 Lambda 函数 (p. 342)。</p> <p>Lambda@Edge 能使您在 AWS 边缘站点上运行 Lambda 函数以响应 CloudFront 事件。有关更多信息 , 请参阅 将 AWS Lambda 与 CloudFront Lambda@Edge 结合使用 (p. 174)。</p>	2016 年 12 月 3 日
AWS Lambda 可将 Amazon Lex 添加为受支持的事件源。	使用 Lambda 和 Amazon Lex , 您可以为 Slack 和 Facebook 等各种服务快速构建聊天机器人。有关更多信息 , 请参阅 将 AWS Lambda 与 Amazon Lex 结合使用 (p. 226) 。	2016 年 11 月 30 日
美国西部 (加利福利亚北部) 区域	AWS Lambda 现于美国西部 (加利福利亚北部) 区域中可用。有关 Lambda 区域和终端节点的更多信息 , 请参阅 AWS General Reference 中的 区域和终端节点 。	2016 年 11 月 21 日
引入 AWS 无服务器应用程序模型以创建和部署基于 Lambda 的应用程序 , 以及将环境变量用于 Lambda 函数配置设置。	<p>AWS 无服务器应用程序模型 : 您可以使用 AWS SAM 定义用于在无服务器应用程序内表示资源的语法。要部署您的应用程序 , 只需在 AWS CloudFormation 模板文件 (在 JSON 或 YAML 中写入) 中作为应用程序的一部分来指定资源及其相关权限策略 , 打包您的部署项目 , 然后部署该模板。有关更多信息 , 请参阅 AWS Lambda 应用程序 (p. 120)。</p> <p>环境变量 : 您可以使用环境变量为 Lambda 函数指定函数代码以外的配置设置。有关更多信息 , 请参阅 使用 AWS Lambda 环境变量 (p. 49)。</p>	2016 年 11 月 18 日
亚太区域 (首尔)	AWS Lambda 现于亚太区域 (首尔) 中可用。有关 Lambda 区域和终端节点的更多信息 , 请参阅 AWS General Reference 中的 区域和终端节点 。	2016 年 8 月 29 日
亚太区域 (悉尼)	Lambda 现于亚太区域 (悉尼) 中可用。有关 Lambda 区域和终端节点的更多信息 , 请参阅 AWS General Reference 中的 区域和终端节点 。	2016 年 6 月 23 日
对 Lambda 控制台的更新	已更新 Lambda 控制台以简化角色创建过程。有关更多信息 , 请参阅 使用控制台创建 Lambda 函数 (p. 3) 。	2016 年 6 月 23 日

更改	描述	日期
AWS Lambda 现在支持 Node.js 运行时 v4.3	AWS Lambda 增加了对 Node.js 运行时 v4.3 的支持。有关更多信息，请参阅 使用 Node.js 构建 Lambda 函数 (p. 265) 。	2016 年 4 月 7 日
欧洲（法兰克福）区域	Lambda 目前在欧洲（法兰克福）区域中可用。有关 Lambda 区域和终端节点的更多信息，请参阅 AWS General Reference 中的 区域和终端节点 。	2016 年 3 月 14 日
VPC 支持	您现在可以配置 Lambda 函数来访问您的 VPC 中的资源。有关更多信息，请参阅 配置 Lambda 函数以访问 VPC 中的资源 (p. 72) 。	2016 年 2 月 11 日
已更新 AWS Lambda 运行时。	更新了 执行环境 (p. 108) 。	2015 年 11 月 4 日
版本控制支持、用于开发 Lambda 函数代码的 Python、计划的事件和执行时间增加	<p>您现在可以使用 Python 开发您的 Lambda 函数代码。有关更多信息，请参阅使用 Python 构建 Lambda 函数 (p. 277)。</p> <p>版本控制：您可以保留 Lambda 函数的一个或多个版本。利用版本控制，您可以控制在不同的环境（例如，开发、测试或生产环境）中执行的 Lambda 函数版本。有关更多信息，请参阅AWS Lambda 函数版本 (p. 63)。</p> <p>计划的事件：您也可以使用 AWS Lambda 控制台将 AWS Lambda 设置为定期调用您的代码。您可以指定一个固定速率（小时数、天数或周数）或指定一个 cron 表达式。有关示例，请参阅配合使用 AWS Lambda 和 Amazon CloudWatch Events (p. 167)。</p> <p>执行时间增加：您现在可以设置您的 Lambda 函数运行最多五分钟以允许更长时间运行的函数，例如大量数据注入和处理作业。</p>	2015 年 10 月 8 日
对于 DynamoDB 流的支持	DynamoDB 流现在普遍可用，您可以在 DynamoDB 可用的所有区域使用它。您可以为自己的表启用 DynamoDB 流，并使用 Lambda 函数作为该表的触发器。触发器是为响应对 DynamoDB 表做出的更新而采取的自定义操作。有关示例演练的信息，请参阅 教程：将 AWS Lambda 与 Amazon DynamoDB 流结合使用 (p. 190) 。	2015 年 7 月 14 日
AWS Lambda 现在支持通过兼容 REST 的客户端调用 Lambda 函数。	<p>以前，要从 Web、移动设备或 IoT 应用程序调用 Lambda 函数，您需要 AWS 开发工具包（例如：适用于 Java 的 AWS 开发工具包、适用于 Android 的 AWS 开发工具包或适用于 iOS 的 AWS 开发工具包）。现在，AWS Lambda 支持在兼容 REST 的客户端上通过可借助 Amazon API Gateway 创建的自定义 API 调用 Lambda 函数。您可以向 Lambda 函数终端节点 URL 发送请求。您可以在该终端节点上配置安全性以允许开放性访问，利用 AWS Identity and Access Management (IAM) 授权访问，或使用 API 密钥限制其他人对您的 Lambda 函数的访问。</p> <p>有关示例入门练习，请参阅配合使用 AWS Lambda 和 Amazon API Gateway (p. 141)。</p> <p>有关 Amazon API Gateway 的更多信息，请参阅 https://aws.amazon.com/api-gateway/。</p>	2015 年 7 月 09 日

更改	描述	日期
AWS Lambda 控制台现在可提供蓝图，以轻松地创建 Lambda 函数并测试它们。	AWS Lambda 控制台提供了一组蓝图。每个蓝图是您的 Lambda 函数提供了示例事件源配置和示例代码，您可以使用它们轻松地创建基于 Lambda 的应用程序。所有 AWS Lambda 入门练习现在都使用这些蓝图。有关更多信息，请参阅 开始使用 AWS Lambda (p. 3) 。	2015 年 7 月 09 日
AWS Lambda 现在支持使用 Java 编写 Lambda 函数。	您现在可以使用 Java 编写 Lambda 代码。有关更多信息，请参阅 使用 Java 构建 Lambda 函数 (p. 301) 。	2015 年 6 月 15 日
在创建或更新 Lambda 函数时，AWS Lambda 现在支持以函数 .zip 的形式指定 Amazon S3 对象。	可以将 Lambda 函数部署程序包 (.zip 文件) 上传到要创建 Lambda 函数的同一区域中的 Amazon S3 存储桶中。然后，您可以在创建或更新 Lambda 函数时指定存储桶名称和对象键名称。	2015 年 5 月 28 日
AWS Lambda 现在普遍可用且增加了对移动后端的支持	<p>AWS Lambda 现在可普遍用于生产环境。此外，该版本还推出了一些新的功能，让使用 AWS Lambda 构建手机、平板电脑和物联网 (IoT) 后端变得更加简单（可自动扩展而无需预置或管理基础设施）。AWS Lambda 现在支持实时（同步）和异步事件。其他功能包括更简单的事件源配置和管理。引入了针对 Lambda 函数的资源策略，简化了权限模型和编程模型。</p> <p>文档进行了相应的更新。有关信息，请参阅以下主题：</p> <ul style="list-style-type: none"> 开始使用 AWS Lambda (p. 3) AWS Lambda 	2015 年 4 月 9 日
预览版	AWS Lambda 开发人员指南 预览版。	2014 年 11 月 13 日

API 参考

本节包含 AWS Lambda API 参考文档。在执行 API 调用时，您需要提供签名以验证请求。AWS Lambda 支持签名版本 4。有关更多信息，请参阅 Amazon Web Services 一般参考中的[签名版本 4 签名流程](#)。

有关该服务的概述，请参阅[什么是 AWS Lambda？\(p. 1\)](#)。

可以使用 AWS CLI 探索 AWS Lambda API。本指南提供了几个使用 AWS CLI 的教程。

主题

- [Actions \(p. 402\)](#)
- [Data Types \(p. 572\)](#)

Actions

The following actions are supported:

- [AddLayerVersionPermission \(p. 404\)](#)
- [AddPermission \(p. 407\)](#)
- [CreateAlias \(p. 411\)](#)
- [CreateEventSourceMapping \(p. 415\)](#)
- [CreateFunction \(p. 421\)](#)
- [DeleteAlias \(p. 430\)](#)
- [DeleteEventSourceMapping \(p. 432\)](#)
- [DeleteFunction \(p. 436\)](#)
- [DeleteFunctionConcurrency \(p. 438\)](#)
- [DeleteFunctionEventInvokeConfig \(p. 440\)](#)
- [DeleteLayerVersion \(p. 442\)](#)
- [DeleteProvisionedConcurrencyConfig \(p. 444\)](#)
- [GetAccountSettings \(p. 446\)](#)
- [GetAlias \(p. 448\)](#)
- [GetEventSourceMapping \(p. 451\)](#)
- [GetFunction \(p. 455\)](#)
- [GetFunctionConcurrency \(p. 458\)](#)
- [GetFunctionConfiguration \(p. 460\)](#)
- [GetFunctionEventInvokeConfig \(p. 466\)](#)
- [GetLayerVersion \(p. 469\)](#)
- [GetLayerVersionByArn \(p. 472\)](#)
- [GetLayerVersionPolicy \(p. 475\)](#)
- [GetPolicy \(p. 477\)](#)
- [GetProvisionedConcurrencyConfig \(p. 479\)](#)
- [Invoke \(p. 482\)](#)
- [InvokeAsync \(p. 487\)](#)
- [ListAliases \(p. 489\)](#)

- [ListEventSourceMappings \(p. 492\)](#)
- [ListFunctionEventInvokeConfigs \(p. 495\)](#)
- [ListFunctions \(p. 498\)](#)
- [ListLayers \(p. 501\)](#)
- [ListLayerVersions \(p. 503\)](#)
- [ListProvisionedConcurrencyConfigs \(p. 506\)](#)
- [ListTags \(p. 509\)](#)
- [ListVersionsByFunction \(p. 511\)](#)
- [PublishLayerVersion \(p. 514\)](#)
- [PublishVersion \(p. 518\)](#)
- [PutFunctionConcurrency \(p. 525\)](#)
- [PutFunctionEventInvokeConfig \(p. 528\)](#)
- [PutProvisionedConcurrencyConfig \(p. 532\)](#)
- [RemoveLayerVersionPermission \(p. 535\)](#)
- [RemovePermission \(p. 537\)](#)
- [TagResource \(p. 539\)](#)
- [UntagResource \(p. 541\)](#)
- [UpdateAlias \(p. 543\)](#)
- [UpdateEventSourceMapping \(p. 547\)](#)
- [UpdateFunctionCode \(p. 553\)](#)
- [UpdateFunctionConfiguration \(p. 560\)](#)
- [UpdateFunctionEventInvokeConfig \(p. 569\)](#)

AddLayerVersionPermission

Adds permissions to the resource-based policy of a version of an AWS Lambda layer. Use this action to grant layer usage permission to other accounts. You can grant permission to a single account, all AWS accounts, or all accounts in an organization.

To revoke permission, call [RemoveLayerVersionPermission \(p. 535\)](#) with the statement ID that you specified when you added it.

Request Syntax

```
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"
}
```

URI Request Parameters

The request requires the following URI parameters.

[LayerName \(p. 404\)](#)

The name or Amazon Resource Name (ARN) of the layer.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (`arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\:\d{12}:layer:[a-zA-Z0-9-_+]|[a-zA-Z0-9-_]+`)

[RevisionId \(p. 404\)](#)

Only update the policy if the revision ID matches the ID specified. Use this option to avoid modifying a policy that has changed since you last read it.

[VersionNumber \(p. 404\)](#)

The version number.

Request Body

The request accepts the following data in JSON format.

[Action \(p. 404\)](#)

The API action that grants access to the layer. For example, `lambda:GetLayerVersion`.

Type: String

Pattern: `lambda:GetLayerVersion`

Required: Yes

[OrganizationId \(p. 404\)](#)

With the principal set to `*`, grant permission to all accounts in the specified organization.

Type: String

Pattern: o-[a-zA-Z0-9]{10,32}

Required: No

[Principal \(p. 404\)](#)

An account ID, or * to grant permission to all AWS accounts.

Type: String

Pattern: \d{12}|*|arn:(aws[a-zA-Z-]*):iam::\d{12}:root

Required: Yes

[StatementId \(p. 404\)](#)

An identifier that distinguishes the policy from others on the same layer version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ([a-zA-Z0-9-_]+)

Required: Yes

Response Syntax

```
HTTP/1.1 201
Content-type: application/json

{
    "RevisionId": "string",
    "Statement": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

[RevisionId \(p. 405\)](#)

A unique identifier for the current revision of the policy.

Type: String

[Statement \(p. 405\)](#)

The permission statement.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400
`PolicyLengthExceededException`

The permissions policy for the resource is too large. [Learn more](#)

HTTP Status Code: 400
`PreconditionFailedException`

The `RevisionId` provided does not match the latest `RevisionId` for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest `RevisionId` for your resource.

HTTP Status Code: 412
`ResourceConflictException`

The resource already exists, or another operation is in progress.

HTTP Status Code: 409
`ResourceNotFoundException`

The resource specified in the request does not exist.

HTTP Status Code: 404
`ServiceException`

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
`TooManyRequestsException`

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

AddPermission

Grants an AWS service or another account permission to use a function. You can apply the policy at the function level, or specify a qualifier to restrict access to a single version or alias. If you use a qualifier, the invoker must use the full Amazon Resource Name (ARN) of that version or alias to invoke the function.

To grant permission to another account, specify the account ID as the `Principal`. For AWS services, the principal is a domain-style identifier defined by the service, like `s3.amazonaws.com` or `sns.amazonaws.com`. For AWS services, you can also specify the ARN of the associated resource as the `SourceArn`. If you grant permission to a service principal without specifying the source, other accounts could potentially configure resources in their account to invoke your Lambda function.

This action adds a statement to a resource-based permissions policy for the function. For more information about function policies, see [Lambda Function Policies](#).

Request Syntax

```
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"
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 407\)](#)

The name of the Lambda function, version, or alias.

Name formats

- Function name - `my-function` (name-only), `my-function:v1` (with alias).
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- Partial ARN - `123456789012:function:my-function`.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?`

[Qualifier \(p. 407\)](#)

Specify a version or alias to add permissions to a published version of the function.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (`|[a-zA-Z0-9$-_]+`)

Request Body

The request accepts the following data in JSON format.

[Action \(p. 407\)](#)

The action that the principal can use on the function. For example, `lambda:InvokeFunction` or `lambda:GetFunction`.

Type: String

Pattern: (`lambda:[*]` | `lambda:[a-zA-Z]+[*]`)

Required: Yes

[EventSourceToken \(p. 407\)](#)

For Alexa Smart Home functions, a token that must be supplied by the invoker.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Pattern: [a-zA-Z0-9._\-\-]+

Required: No

[Principal \(p. 407\)](#)

The AWS service or account that invokes the function. If you specify a service, use `SourceArn` or `SourceAccount` to limit who can invoke the function through that service.

Type: String

Pattern: .*

Required: Yes

[RevisionId \(p. 407\)](#)

Only update the policy if the revision ID matches the ID that's specified. Use this option to avoid modifying a policy that has changed since you last read it.

Type: String

Required: No

[SourceAccount \(p. 407\)](#)

For Amazon S3, the ID of the account that owns the resource. Use this together with `SourceArn` to ensure that the resource is owned by the specified account. It is possible for an Amazon S3 bucket to be deleted by its owner and recreated by another account.

Type: String

Pattern: \d{12}

Required: No

[SourceArn \(p. 407\)](#)

For AWS services, the ARN of the AWS resource that invokes the function. For example, an Amazon S3 bucket or Amazon SNS topic.

Type: String

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+:(a-z{2}(-gov)?-[a-zA-Z]+\d{1})?:(\d{12})?:(.*?)

Required: No

[StatementId \(p. 407\)](#)

A statement identifier that differentiates the statement from others in the same policy.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ([a-zA-Z0-9_-]+)

Required: Yes

Response Syntax

```
HTTP/1.1 201
Content-type: application/json

{
    "Statement": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

[Statement \(p. 409\)](#)

The permission statement that's added to the function policy.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

PolicyLengthExceededException

The permissions policy for the resource is too large. [Learn more](#)

HTTP Status Code: 400

PreconditionFailedException

The RevisionId provided does not match the latest RevisionId for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest RevisionId for your resource.

HTTP Status Code: 412

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

CreateAlias

Creates an [alias](#) for a Lambda function version. Use aliases to provide clients with a function identifier that you can update to invoke a different version.

You can also map an alias to split invocation requests between two versions. Use the `RoutingConfig` parameter to specify a second version and the percentage of invocation requests that it receives.

Request Syntax

```
POST /2015-03-31/functions/FunctionName/aliases HTTP/1.1
Content-type: application/json

{
  "Description": "string",
  "FunctionVersion": "string",
  "Name": "string",
  "RoutingConfig": {
    "AdditionalVersionWeights": {
      "string" : number
    }
  }
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 411)

The name of the Lambda function.

Name formats

- Function name - `MyFunction`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- Partial ARN - `123456789012:function:MyFunction`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?`

Request Body

The request accepts the following data in JSON format.

[Description](#) (p. 411)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[FunctionVersion \(p. 411\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

Required: Yes

[Name \(p. 411\)](#)

The name of the alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\\$)([a-zA-Z0-9-_]+)

Required: Yes

[RoutingConfig \(p. 411\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 578\)](#) object

Required: No

Response Syntax

```
HTTP/1.1 201
Content-type: application/json

{
    "AliasArn": "string",
    "Description": "string",
    "FunctionVersion": "string",
    "Name": "string",
    "RevisionId": "string",
    "RoutingConfig": {
        "AdditionalVersionWeights": {
            "string" : number
        }
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

[AliasArn \(p. 412\)](#)

The Amazon Resource Name (ARN) of the alias.

Type: String

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 \(p. 412\)](#)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 412\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[Name \(p. 412\)](#)

The name of the alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\\$)([a-zA-Z0-9-_]+)

[RevisionId \(p. 412\)](#)

A unique identifier that changes when you update the alias.

Type: String

[RoutingConfig \(p. 412\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 578\)](#) object

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceConflictException](#)

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

[ResourceNotFoundException](#)

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

CreateEventSourceMapping

Creates a mapping between an event source and an AWS Lambda function. Lambda reads items from the event source and triggers the function.

For details about each event source type, see the following topics.

- [Using AWS Lambda with Amazon DynamoDB](#)
- [Using AWS Lambda with Amazon Kinesis](#)
- [Using AWS Lambda with Amazon SQS](#)

The following error handling options are only available for stream sources (DynamoDB and Kinesis):

- `BisectBatchOnFunctionError` - If the function returns an error, split the batch in two and retry.
- `DestinationConfig` - Send discarded records to an Amazon SQS queue or Amazon SNS topic.
- `MaximumRecordAgeInSeconds` - Discard records older than the specified age.
- `MaximumRetryAttempts` - Discard records after the specified number of retries.
- `ParallelizationFactor` - Process multiple batches from each shard concurrently.

Request Syntax

```
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",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "StartingPosition": "string",
    "StartingPositionTimestamp": number
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

[BatchSize \(p. 415\)](#)

The maximum number of items to retrieve in a single batch.

- Amazon Kinesis - Default 100. Max 10,000.
- Amazon DynamoDB Streams - Default 100. Max 1,000.
- Amazon Simple Queue Service - Default 10. Max 10.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

Required: No

[BisectBatchOnFunctionError \(p. 415\)](#)

(Streams) If the function returns an error, split the batch in two and retry.

Type: Boolean

Required: No

[DestinationConfig \(p. 415\)](#)

(Streams) An Amazon SQS queue or Amazon SNS topic destination for discarded records.

Type: [DestinationConfig \(p. 581\)](#) object

Required: No

[Enabled \(p. 415\)](#)

Disables the event source mapping to pause polling and invocation.

Type: Boolean

Required: No

[EventSourceArn \(p. 415\)](#)

The Amazon Resource Name (ARN) of the event source.

- Amazon Kinesis - The ARN of the data stream or a stream consumer.
- Amazon DynamoDB Streams - The ARN of the stream.
- Amazon Simple Queue Service - The ARN of the queue.

Type: String

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12})?:(.*)

Required: Yes

[FunctionName \(p. 415\)](#)

The name of the Lambda function.

Name formats

- Function name - MyFunction.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- Version or Alias ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction:PROD.
- Partial ARN - 123456789012:function:MyFunction.

The length constraint applies only to the full ARN. If you specify only the function name, it's limited to 64 characters in length.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

Required: Yes

[MaximumBatchingWindowInSeconds \(p. 415\)](#)

(Streams) The maximum amount of time to gather records before invoking the function, in seconds.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 300.

Required: No

[MaximumRecordAgeInSeconds \(p. 415\)](#)

(Streams) The maximum age of a record that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 604800.

Required: No

[MaximumRetryAttempts \(p. 415\)](#)

(Streams) The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000.

Required: No

[ParallelizationFactor \(p. 415\)](#)

(Streams) The number of batches to process from each shard concurrently.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

Required: No

[StartingPosition \(p. 415\)](#)

The position in a stream from which to start reading. Required for Amazon Kinesis and Amazon DynamoDB Streams sources. AT_TIMESTAMP is only supported for Amazon Kinesis streams.

Type: String

Valid Values: TRIM_HORIZON | LATEST | AT_TIMESTAMP

Required: No

[StartingPositionTimestamp \(p. 415\)](#)

With StartingPosition set to AT_TIMESTAMP, the time from which to start reading, in Unix time seconds.

Type: Timestamp

Required: No

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
    "BatchSize": number,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "State": "string",
    "StateTransitionReason": "string",
    "UUID": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

[BatchSize \(p. 418\)](#)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[BisectBatchOnFunctionError \(p. 418\)](#)

(Streams) If the function returns an error, split the batch in two and retry.

Type: Boolean

[DestinationConfig \(p. 418\)](#)

(Streams) An Amazon SQS queue or Amazon SNS topic destination for discarded records.

Type: [DestinationConfig \(p. 581\)](#) object

[EventSourceArn \(p. 418\)](#)

The Amazon Resource Name (ARN) of the event source.

Type: String

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+:(a-z{2}(-gov)?-[a-z]+\d{1})?:(\d{12})?:(.*?)

[FunctionArn \(p. 418\)](#)

The ARN of the Lambda function.

Type: String

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 \(p. 418\)](#)

The date that the event source mapping was last updated, or its state changed, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 418\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[MaximumBatchingWindowInSeconds \(p. 418\)](#)

(Streams) The maximum amount of time to gather records before invoking the function, in seconds.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 300.

[MaximumRecordAgeInSeconds \(p. 418\)](#)

(Streams) The maximum age of a record that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 604800.

[MaximumRetryAttempts \(p. 418\)](#)

(Streams) The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000.

[ParallelizationFactor \(p. 418\)](#)

(Streams) The number of batches to process from each shard concurrently.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

[State \(p. 418\)](#)

The state of the event source mapping. It can be one of the following: Creating, Enabling, Enabled, Disabling, Disabled, Updating, or Deleting.

Type: String

[StateTransitionReason \(p. 418\)](#)

Indicates whether the last change to the event source mapping was made by a user, or by the Lambda service.

Type: String
[UUID \(p. 418\)](#)

The identifier of the event source mapping.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

CreateFunction

Creates a Lambda function. To create a function, you need a [deployment package](#) and an [execution role](#). The deployment package contains your function code. The execution role grants the function permission to use AWS services, such as Amazon CloudWatch Logs for log streaming and AWS X-Ray for request tracing.

When you create a function, Lambda provisions an instance of the function and its supporting resources. If your function connects to a VPC, this process can take a minute or so. During this time, you can't invoke or modify the function. The `State`, `StateReason`, and `StateReasonCode` fields in the response from [GetFunctionConfiguration \(p. 460\)](#) indicate when the function is ready to invoke. For more information, see [Function States](#).

A function has an unpublished version, and can have published versions and aliases. The unpublished version changes when you update your function's code and configuration. A published version is a snapshot of your function code and configuration that can't be changed. An alias is a named resource that maps to a version, and can be changed to map to a different version. Use the `Publish` parameter to create version 1 of your function from its initial configuration.

The other parameters let you configure version-specific and function-level settings. You can modify version-specific settings later with [UpdateFunctionConfiguration \(p. 560\)](#). Function-level settings apply to both the unpublished and published versions of the function, and include tags ([TagResource \(p. 539\)](#)) and per-function concurrency limits ([PutFunctionConcurrency \(p. 525\)](#)).

If another account or an AWS service invokes your function, use [AddPermission \(p. 407\)](#) to grant permission by creating a resource-based IAM policy. You can grant permissions at the function level, on a version, or on an alias.

To invoke your function directly, use [Invoke \(p. 482\)](#). To invoke your function in response to events in other AWS services, create an event source mapping ([CreateEventSourceMapping \(p. 415\)](#)), or configure a function trigger in the other service. For more information, see [Invoking Functions](#).

Request Syntax

```
POST /2015-03-31/functions HTTP/1.1
Content-type: application/json

{
  "Code": {
    "S3Bucket": "string",
    "S3Key": "string",
    "S3ObjectVersion": "string",
    "ZipFile": blob
  },
  "DeadLetterConfig": {
    "TargetArn": "string"
  },
  "Description": "string",
  "Environment": {
    "Variables": {
      "string" : "string"
    }
  },
  "FunctionName": "string",
  "Handler": "string",
  "KMSKeyArn": "string",
  "Layers": [ "string" ],
  "MemorySize": number,
  "Publish": boolean,
  "Role": "string",
  "Timeout": number
}
```

```
"Runtime": "string",
"Tags": {
    "string" : "string"
},
"Timeout": number,
"TracingConfig": {
    "Mode": "string"
},
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ]
}
}
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request accepts the following data in JSON format.

[Code \(p. 421\)](#)

The code for the function.

Type: [FunctionCode \(p. 588\)](#) object

Required: Yes

[DeadLetterConfig \(p. 421\)](#)

A dead letter queue configuration that specifies the queue or topic where Lambda sends asynchronous events when they fail processing. For more information, see [Dead Letter Queues](#).

Type: [DeadLetterConfig \(p. 580\)](#) object

Required: No

[Description \(p. 421\)](#)

A description of the function.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[Environment \(p. 421\)](#)

Environment variables that are accessible from function code during execution.

Type: [Environment \(p. 582\)](#) object

Required: No

[FunctionName \(p. 421\)](#)

The name of the Lambda function.

Name formats

- Function name - my-function.

- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

Required: Yes

[Handler \(p. 421\)](#)

The name of the method within your code that Lambda calls to execute your function. The format includes the file name. It can also include namespaces and other qualifiers, depending on the runtime. For more information, see [Programming Model](#).

Type: String

Length Constraints: Maximum length of 128.

Pattern: [^\s]+

Required: Yes

[KMSKeyArn \(p. 421\)](#)

The ARN of the AWS Key Management Service (AWS KMS) key that's used to encrypt your function's environment variables. If it's not provided, AWS Lambda uses a default service key.

Type: String

Pattern: (arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:[^.])|()

Required: No

[Layers \(p. 421\)](#)

A list of [function layers](#) to add to the function's execution environment. Specify each layer by its ARN, including the version.

Type: Array of strings

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

Required: No

[MemorySize \(p. 421\)](#)

The amount of memory that your function has access to. Increasing the function's memory also increases its CPU allocation. The default value is 128 MB. The value must be a multiple of 64 MB.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

Required: No

[Publish \(p. 421\)](#)

Set to true to publish the first version of the function during creation.

Type: Boolean

Required: No

[Role \(p. 421\)](#)

The Amazon Resource Name (ARN) of the function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@-_/.]+

Required: Yes

[Runtime \(p. 421\)](#)

The identifier of the function's [runtime](#).

Type: String

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

Required: Yes

[Tags \(p. 421\)](#)

A list of [tags](#) to apply to the function.

Type: String to string map

Required: No

[Timeout \(p. 421\)](#)

The amount of time that Lambda allows a function to run before stopping it. The default is 3 seconds. The maximum allowed value is 900 seconds.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

[TracingConfig \(p. 421\)](#)

Set `Mode` to `Active` to sample and trace a subset of incoming requests with AWS X-Ray.

Type: [TracingConfig \(p. 607\)](#) object

Required: No

[VpcConfig \(p. 421\)](#)

For network connectivity to AWS resources in a VPC, specify a list of security groups and subnets in the VPC. When you connect a function to a VPC, it can only access resources and the internet through that VPC. For more information, see [VPC Settings](#).

Type: [VpcConfig \(p. 609\)](#) object

Required: No

Response Syntax

```
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"
        }
    },
    "FunctionArn": "string",
    "FunctionName": "string",
    "Handler": "string",
    "KMSKeyArn": "string",
    "LastModified": "string",
    "LastUpdateStatus": "string",
    "LastUpdateStatusReason": "string",
    "LastUpdateStatusReasonCode": "string",
    "Layers": [
        {
            "Arn": "string",
            "CodeSize": number
        }
    ],
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "string",
    "State": "string",
    "StateReason": "string",
    "StateReasonCode": "string",
    "Timeout": number,
    "TracingConfig": {
        "Mode": "string"
    },
    "Version": "string",
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
        "SubnetIds": [ "string" ],
        "VpcId": "string"
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

[CodeSha256 \(p. 425\)](#)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize \(p. 425\)](#)

The size of the function's deployment package, in bytes.

Type: Long

[DeadLetterConfig \(p. 425\)](#)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 580\)](#) object

[Description \(p. 425\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 425\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 584\)](#) object

[FunctionArn \(p. 425\)](#)

The function's Amazon Resource Name (ARN).

Type: String

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 \(p. 425\)](#)

The name of the function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 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 \(p. 425\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 425\)](#)

The KMS key that's used to encrypt the function's environment variables. This key is only returned if you've configured a customer managed CMK.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:[.*])|()`

[LastModified \(p. 425\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[LastUpdateStatus \(p. 425\)](#)

The status of the last update that was performed on the function. This is first set to `Successful` after function creation completes.

Type: String

Valid Values: `Successful` | `Failed` | `InProgress`

[LastUpdateStatusReason \(p. 425\)](#)

The reason for the last update that was performed on the function.

Type: String

[LastUpdateStatusReasonCode \(p. 425\)](#)

The reason code for the last update that was performed on the function.

Type: String

Valid Values: `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup`

[Layers \(p. 425\)](#)

The function's `layers`.

Type: Array of [Layer \(p. 597\)](#) objects

[MasterArn \(p. 425\)](#)

For Lambda@Edge functions, the ARN of the master function.

Type: String

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 \(p. 425\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 425\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 425\)](#)

The function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\\-/]+
[Runtime \(p. 425\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

[State \(p. 425\)](#)

The current state of the function. When the state is Inactive, you can reactivate the function by invoking it.

Type: String

Valid Values: Pending | Active | Inactive | Failed

[StateReason \(p. 425\)](#)

The reason for the function's current state.

Type: String

[StateReasonCode \(p. 425\)](#)

The reason code for the function's current state. When the code is Creating, you can't invoke or modify the function.

Type: String

Valid Values: Idle | Creating | Restoring | EniLimitExceeded | InsufficientRolePermissions | InvalidConfiguration | InternalError | SubnetOutOfIPAddresses | InvalidSubnet | InvalidSecurityGroup

[Timeout \(p. 425\)](#)

The amount of time that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 425\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 608\)](#) object

[Version \(p. 425\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 425\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 610\)](#) object

Errors

CodeStorageExceededException

You have exceeded your maximum total code size per account. [Learn more](#)

HTTP Status Code: 400

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteAlias

Deletes a Lambda function [alias](#).

Request Syntax

```
DELETE /2015-03-31/functions/FunctionName/aliases/Name HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 430)

The name of the Lambda function.

Name formats

- Function name - MyFunction.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- Partial ARN - 123456789012:function:MyFunction.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

[Name](#) (p. 430)

The name of the alias.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\\$)([a-zA-Z0-9-_]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

- HTTP Status Code: 400
ResourceConflictException

The resource already exists, or another operation is in progress.
- HTTP Status Code: 409
ServiceException

The AWS Lambda service encountered an internal error.
- HTTP Status Code: 500
TooManyRequestsException

The request throughput limit was exceeded.
- HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteEventSourceMapping

Deletes an [event source mapping](#). You can get the identifier of a mapping from the output of [ListEventSourceMappings](#) (p. 492).

When you delete an event source mapping, it enters a `Deleting` state and might not be completely deleted for several seconds.

Request Syntax

```
DELETE /2015-03-31/event-source-mappings/UUID HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[UUID](#) (p. 432)

The identifier of the event source mapping.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
    "BatchSize": number,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "State": "string",
    "StateTransitionReason": "string",
    "UUID": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

[BatchSize \(p. 432\)](#)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[BisectBatchOnFunctionError \(p. 432\)](#)

(Streams) If the function returns an error, split the batch in two and retry.

Type: Boolean

[DestinationConfig \(p. 432\)](#)

(Streams) An Amazon SQS queue or Amazon SNS topic destination for discarded records.

Type: [DestinationConfig \(p. 581\)](#) object

[EventSourceArn \(p. 432\)](#)

The Amazon Resource Name (ARN) of the event source.

Type: String

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+:([a-z]{2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.*)

[FunctionArn \(p. 432\)](#)

The ARN of the Lambda function.

Type: String

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 \(p. 432\)](#)

The date that the event source mapping was last updated, or its state changed, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 432\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[MaximumBatchingWindowInSeconds \(p. 432\)](#)

(Streams) The maximum amount of time to gather records before invoking the function, in seconds.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 300.

[MaximumRecordAgeInSeconds \(p. 432\)](#)

(Streams) The maximum age of a record that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 604800.

[MaximumRetryAttempts \(p. 432\)](#)

(Streams) The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000.

[ParallelizationFactor \(p. 432\)](#)

(Streams) The number of batches to process from each shard concurrently.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

[State \(p. 432\)](#)

The state of the event source mapping. It can be one of the following: Creating, Enabling, Enabled, Disabling, Disabled, Updating, or Deleting.

Type: String

[StateTransitionReason \(p. 432\)](#)

Indicates whether the last change to the event source mapping was made by a user, or by the Lambda service.

Type: String

[UUID \(p. 432\)](#)

The identifier of the event source mapping.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceInUseException](#)

The operation conflicts with the resource's availability. For example, you attempted to update an EventSource Mapping in CREATING, or tried to delete a EventSource mapping currently in the UPDATING state.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

[TooManyRequestsException](#)

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteFunction

Deletes a Lambda function. To delete a specific function version, use the [Qualifier](#) parameter. Otherwise, all versions and aliases are deleted.

To delete Lambda event source mappings that invoke a function, use [DeleteEventSourceMapping \(p. 432\)](#). For AWS services and resources that invoke your function directly, delete the trigger in the service where you originally configured it.

Request Syntax

```
DELETE /2015-03-31/functions/FunctionName?Qualifier=Qualifier HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 436\)](#)

The name of the Lambda function or version.

Name formats

- Function name - my-function (name-only), my-function:1 (with version).
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

[Qualifier \(p. 436\)](#)

Specify a version to delete. You can't delete a version that's referenced by an alias.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteFunctionConcurrency

Removes a concurrent execution limit from a function.

Request Syntax

```
DELETE /2017-10-31/functions/FunctionName/concurrency HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 438)

The name of the Lambda function.

Name formats

- Function name - my-function.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteFunctionEventInvokeConfig

Deletes the configuration for asynchronous invocation for a function, version, or alias.

To configure options for asynchronous invocation, use [PutFunctionEventInvokeConfig \(p. 528\)](#).

Request Syntax

```
DELETE /2019-09-25/functions/FunctionName/event-invoke-config?Qualifier=Qualifier HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 440\)](#)

The name of the Lambda function, version, or alias.

Name formats

- Function name - my-function (name-only), my-function:v1 (with alias).
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

[Qualifier \(p. 440\)](#)

A version number or alias name.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400
ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

DeleteLayerVersion

Deletes a version of an AWS Lambda layer. Deleted versions can no longer be viewed or added to functions. To avoid breaking functions, a copy of the version remains in Lambda until no functions refer to it.

Request Syntax

```
DELETE /2018-10-31/layers/LayerName/versions/VersionNumber HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[LayerName \(p. 442\)](#)

The name or Amazon Resource Name (ARN) of the layer.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\:\d{12}:layer:[a-zA-Z0-9-_+]| [a-zA-Z0-9-_]+)

[VersionNumber \(p. 442\)](#)

The version number.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

DeleteProvisionedConcurrencyConfig

Deletes the provisioned concurrency configuration for a function.

Request Syntax

```
DELETE /2019-09-30/functions/FunctionName/provisioned-concurrency?Qualifier=Qualifier
HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 444)

The name of the Lambda function.

Name formats

- Function name - `my-function`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- Partial ARN - `123456789012:function:my-function`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?`

Qualifier (p. 444)

The version number or alias name.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(|[a-zA-Z0-9$-_]+)`

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

- HTTP Status Code: 400
`ResourceConflictException`

The resource already exists, or another operation is in progress.
- HTTP Status Code: 409
`ResourceNotFoundException`

The resource specified in the request does not exist.
- HTTP Status Code: 404
`ServiceException`

The AWS Lambda service encountered an internal error.
- HTTP Status Code: 500
`TooManyRequestsException`

The request throughput limit was exceeded.
- HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetAccountSettings

Retrieves details about your account's [limits](#) and usage in an AWS Region.

Request Syntax

```
GET /2016-08-19/account-settings/ HTTP/1.1
```

URI Request Parameters

The request does not use any URI parameters.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "AccountLimit": {
    "CodeSizeUnzipped": number,
    "CodeSizeZipped": number,
    "ConcurrentExecutions": number,
    "TotalCodeSize": number,
    "UnreservedConcurrentExecutions": number
  },
  "AccountUsage": {
    "FunctionCount": number,
    "TotalCodeSize": number
  }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[AccountLimit \(p. 446\)](#)

Limits that are related to concurrency and code storage.

Type: [AccountLimit \(p. 574\)](#) object

[AccountUsage \(p. 446\)](#)

The number of functions and amount of storage in use.

Type: [AccountUsage \(p. 575\)](#) object

Errors

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
`TooManyRequestsException`

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetAlias

Returns details about a Lambda function [alias](#).

Request Syntax

```
GET /2015-03-31/functions/FunctionName/aliases/Name HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 448)

The name of the Lambda function.

Name formats

- Function name - `MyFunction`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- Partial ARN - `123456789012:function:MyFunction`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?`

[Name](#) (p. 448)

The name of the alias.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(?!^[\d-])^[\w-]{1,128}$`

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "AliasArn": "string",
    "Description": "string",
    "FunctionVersion": "string",
    "Name": "string",
    "RevisionId": "string",
    "RoutingConfig": {
        "AdditionalVersionWeights": {
            "string": number
        }
    }
}
```

```
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[AliasArn \(p. 448\)](#)

The Amazon Resource Name (ARN) of the alias.

Type: String

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 \(p. 448\)](#)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 448\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$\$LATEST|[0-9]+)`

[Name \(p. 448\)](#)

The name of the alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(?!^[\0-9]+\$)([a-zA-Z0-9-_]+)`

[RevisionId \(p. 448\)](#)

A unique identifier that changes when you update the alias.

Type: String

[RoutingConfig \(p. 448\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 578\)](#) object

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400
ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetEventSourceMapping

Returns details about an event source mapping. You can get the identifier of a mapping from the output of [ListEventSourceMappings](#) (p. 492).

Request Syntax

```
GET /2015-03-31/event-source-mappings/UUID HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[UUID](#) (p. 451)

The identifier of the event source mapping.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "BatchSize": number,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "State": "string",
    "StateTransitionReason": "string",
    "UUID": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[BatchSize \(p. 451\)](#)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[BisectBatchOnFunctionError \(p. 451\)](#)

(Streams) If the function returns an error, split the batch in two and retry.

Type: Boolean

[DestinationConfig \(p. 451\)](#)

(Streams) An Amazon SQS queue or Amazon SNS topic destination for discarded records.

Type: [DestinationConfig \(p. 581\)](#) object

[EventSourceArn \(p. 451\)](#)

The Amazon Resource Name (ARN) of the event source.

Type: String

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+:(a-z{2}(-gov)?-[a-zA-Z]+\d{1})?:(\d{12})?:(.*?)

[FunctionArn \(p. 451\)](#)

The ARN of the Lambda function.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:lambda:[a-zA-Z]{2}(-gov)?-[a-zA-Z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$\\$LATEST|[a-zA-Z0-9-_]+))?

[LastModified \(p. 451\)](#)

The date that the event source mapping was last updated, or its state changed, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 451\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[MaximumBatchingWindowInSeconds \(p. 451\)](#)

(Streams) The maximum amount of time to gather records before invoking the function, in seconds.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 300.

[MaximumRecordAgeInSeconds \(p. 451\)](#)

(Streams) The maximum age of a record that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 604800.

[MaximumRetryAttempts \(p. 451\)](#)

(Streams) The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000.

[ParallelizationFactor \(p. 451\)](#)

(Streams) The number of batches to process from each shard concurrently.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

[State \(p. 451\)](#)

The state of the event source mapping. It can be one of the following: `Creating`, `Enabling`, `Enabled`, `Disabling`, `Disabled`, `Updating`, or `Deleting`.

Type: String

[StateTransitionReason \(p. 451\)](#)

Indicates whether the last change to the event source mapping was made by a user, or by the Lambda service.

Type: String

[UUID \(p. 451\)](#)

The identifier of the event source mapping.

Type: String

Errors

`InvalidParameterValueException`

One of the parameters in the request is invalid.

HTTP Status Code: 400

`ResourceNotFoundException`

The resource specified in the request does not exist.

HTTP Status Code: 404

`ServiceException`

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

`TooManyRequestsException`

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)

- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetFunction

Returns information about the function or function version, with a link to download the deployment package that's valid for 10 minutes. If you specify a function version, only details that are specific to that version are returned.

Request Syntax

```
GET /2015-03-31/functions/FunctionName?Qualifier=Qualifier HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 455)

The name of the Lambda function, version, or alias.

Name formats

- Function name - my-function (name-only), my-function:v1 (with alias).
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

Qualifier (p. 455)

Specify a version or alias to get details about a published version of the function.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "Code": {
        "Location": "string",
        "RepositoryType": "string"
    },
    "Concurrency": {
        "ReservedConcurrentExecutions": number
    }
}
```

```
        },
        "Configuration": {
            "CodeSha256": "string",
            "CodeSize": number,
            "DeadLetterConfig": {
                "TargetArn": "string"
            },
            "Description": "string",
            "Environment": {
                "Error": {
                    "ErrorCode": "string",
                    "Message": "string"
                },
                "Variables": {
                    "string" : "string"
                }
            },
            "FunctionArn": "string",
            "FunctionName": "string",
            "Handler": "string",
            "KMSKeyArn": "string",
            "LastModified": "string",
            "LastUpdateStatus": "string",
            "LastUpdateStatusReason": "string",
            "LastUpdateStatusReasonCode": "string",
            "Layers": [
                {
                    "Arn": "string",
                    "CodeSize": number
                }
            ],
            "MasterArn": "string",
            "MemorySize": number,
            "RevisionId": "string",
            "Role": "string",
            "Runtime": "string",
            "State": "string",
            "StateReason": "string",
            "StateReasonCode": "string",
            "Timeout": number,
            "TracingConfig": {
                "Mode": "string"
            },
            "Version": "string",
            "VpcConfig": {
                "SecurityGroupIds": [ "string" ],
                "SubnetIds": [ "string" ],
                "VpcId": "string"
            }
        },
        "Tags": {
            "string" : "string"
        }
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Code \(p. 455\)](#)

The deployment package of the function or version.

Type: [FunctionCodeLocation \(p. 589\)](#) object
[Concurrency \(p. 455\)](#)

The function's reserved concurrency.

Type: [Concurrency \(p. 579\)](#) object
[Configuration \(p. 455\)](#)

The configuration of the function or version.

Type: [FunctionConfiguration \(p. 590\)](#) object
[Tags \(p. 455\)](#)

The function's tags.

Type: String to string map

Errors

`InvalidParameterValueException`

One of the parameters in the request is invalid.

HTTP Status Code: 400

`ResourceNotFoundException`

The resource specified in the request does not exist.

HTTP Status Code: 404

`ServiceException`

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

`TooManyRequestsException`

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetFunctionConcurrency

Returns details about the reserved concurrency configuration for a function. To set a concurrency limit for a function, use [PutFunctionConcurrency \(p. 525\)](#).

Request Syntax

```
GET /2019-09-30/functions/FunctionName/concurrency HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 458\)](#)

The name of the Lambda function.

Name formats

- Function name - my-function.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "ReservedConcurrentExecutions": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[ReservedConcurrentExecutions \(p. 458\)](#)

The number of simultaneous executions that are reserved for the function.

Type: Integer

Valid Range: Minimum value of 0.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetFunctionConfiguration

Returns the version-specific settings of a Lambda function or version. The output includes only options that can vary between versions of a function. To modify these settings, use [UpdateFunctionConfiguration \(p. 560\)](#).

To get all of a function's details, including function-level settings, use [GetFunction \(p. 455\)](#).

Request Syntax

```
GET /2015-03-31/functions/FunctionName/configuration?Qualifier=Qualifier HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 460\)](#)

The name of the Lambda function, version, or alias.

Name formats

- Function name - my-function (name-only), my-function:v1 (with alias).
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

[Qualifier \(p. 460\)](#)

Specify a version or alias to get details about a published version of the function.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Request Body

The request does not have a request body.

Response Syntax

```
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"
    }
},
"FunctionArn": "string",
"FunctionName": "string",
"Handler": "string",
"KMSKeyArn": "string",
"LastModified": "string",
"LastUpdateStatus": "string",
"LastUpdateStatusReason": "string",
"LastUpdateStatusReasonCode": "string",
"Layers": [
    {
        "Arn": "string",
        "CodeSize": number
    }
],
"MasterArn": "string",
"MemorySize": number,
"RevisionId": "string",
"Role": "string",
"Runtime": "string",
"State": "string",
"StateReason": "string",
"StateReasonCode": "string",
"Timeout": number,
"TracingConfig": {
    "Mode": "string"
},
"Version": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[CodeSha256](#) (p. 460)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize](#) (p. 460)

The size of the function's deployment package, in bytes.

Type: Long

[DeadLetterConfig](#) (p. 460)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 580\)](#) object

[Description \(p. 460\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 460\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 584\)](#) object

[FunctionArn \(p. 460\)](#)

The function's Amazon Resource Name (ARN).

Type: String

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 \(p. 460\)](#)

The name of the function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 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 \(p. 460\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: [^\s]+

[KMSKeyArn \(p. 460\)](#)

The KMS key that's used to encrypt the function's environment variables. This key is only returned if you've configured a customer managed CMK.

Type: String

Pattern: (arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-_\.]+\.:*)|()

[LastModified \(p. 460\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[LastUpdateStatus \(p. 460\)](#)

The status of the last update that was performed on the function. This is first set to `Successful` after function creation completes.

Type: String

Valid Values: Successful | Failed | InProgress

[LastUpdateStatusReason \(p. 460\)](#)

The reason for the last update that was performed on the function.

Type: String

[LastUpdateStatusReasonCode \(p. 460\)](#)

The reason code for the last update that was performed on the function.

Type: String

Valid Values: EniLimitExceeded | InsufficientRolePermissions | InvalidConfiguration | InternalError | SubnetOutOfIPAddresses | InvalidSubnet | InvalidSecurityGroup

[Layers \(p. 460\)](#)

The function's [layers](#).

Type: Array of [Layer \(p. 597\)](#) objects

[MasterArn \(p. 460\)](#)

For Lambda@Edge functions, the ARN of the master function.

Type: String

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 \(p. 460\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 460\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 460\)](#)

The function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@/_-]+

[Runtime \(p. 460\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

[State \(p. 460\)](#)

The current state of the function. When the state is `Inactive`, you can reactivate the function by invoking it.

Type: String

Valid Values: `Pending` | `Active` | `Inactive` | `Failed`

[StateReason \(p. 460\)](#)

The reason for the function's current state.

Type: String

[StateReasonCode \(p. 460\)](#)

The reason code for the function's current state. When the code is `Creating`, you can't invoke or modify the function.

Type: String

Valid Values: `Idle` | `Creating` | `Restoring` | `EniLimitExceeded` |
`InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` |
`SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup`

[Timeout \(p. 460\)](#)

The amount of time that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 460\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 608\)](#) object

[Version \(p. 460\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[VpcConfig \(p. 460\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 610\)](#) object

Errors

InvalidArgumentException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetFunctionEventInvokeConfig

Retrieves the configuration for asynchronous invocation for a function, version, or alias.

To configure options for asynchronous invocation, use [PutFunctionEventInvokeConfig \(p. 528\)](#).

Request Syntax

```
GET /2019-09-25/functions/FunctionName/event-invoke-config?Qualifier=Qualifier HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 466\)](#)

The name of the Lambda function, version, or alias.

Name formats

- Function name - my-function (name-only), my-function:v1 (with alias).
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

[Qualifier \(p. 466\)](#)

A version number or alias name.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "DestinationConfig": {
    "OnFailure": {
      "Destination": "string"
    },
    "OnSuccess": {
      "Destination": "string"
    }
}
```

```
    },
    "FunctionArn": "string",
    "LastModified": number,
    "MaximumEventAgeInSeconds": number,
    "MaximumRetryAttempts": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[DestinationConfig \(p. 466\)](#)

A destination for events after they have been sent to a function for processing.

Destinations

- Function - The Amazon Resource Name (ARN) of a Lambda function.
- Queue - The ARN of an SQS queue.
- Topic - The ARN of an SNS topic.
- Event Bus - The ARN of an Amazon EventBridge event bus.

Type: [DestinationConfig \(p. 581\)](#) object

[FunctionArn \(p. 466\)](#)

The Amazon Resource Name (ARN) of the function.

Type: String

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 \(p. 466\)](#)

The date and time that the configuration was last updated, in Unix time seconds.

Type: Timestamp

[MaximumEventAgeInSeconds \(p. 466\)](#)

The maximum age of a request that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 21600.

[MaximumRetryAttempts \(p. 466\)](#)

The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 2.

Errors

InvalidOperationException

One of the parameters in the request is invalid.

HTTP Status Code: 400
ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetLayerVersion

Returns information about a version of an [AWS Lambda layer](#), with a link to download the layer archive that's valid for 10 minutes.

Request Syntax

```
GET /2018-10-31/layers/LayerName/versions/VersionNumber HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[LayerName](#) (p. 469)

The name or Amazon Resource Name (ARN) of the layer.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_+])|([a-zA-Z0-9-_]+)

[VersionNumber](#) (p. 469)

The version number.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "CompatibleRuntimes": [ "string" ],
    "Content": {
        "CodeSha256": "string",
        "CodeSize": number,
        "Location": "string"
    },
    "CreatedDate": "string",
    "Description": "string",
    "LayerArn": "string",
    "LayerVersionArn": "string",
    "LicenseInfo": "string",
    "Version": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[CompatibleRuntimes \(p. 469\)](#)

The layer's compatible runtimes.

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

[Content \(p. 469\)](#)

Details about the layer version.

Type: [LayerVersionContentOutput \(p. 600\)](#) object

[CreatedDate \(p. 469\)](#)

The date that the layer version was created, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Description \(p. 469\)](#)

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[LayerArn \(p. 469\)](#)

The ARN of the layer.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+

[LayerVersionArn \(p. 469\)](#)

The ARN of the layer version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

[LicenseInfo \(p. 469\)](#)

The layer's software license.

Type: String

Length Constraints: Maximum length of 512.

[Version \(p. 469\)](#)

The version number.

Type: Long

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetLayerVersionByArn

Returns information about a version of an [AWS Lambda layer](#), with a link to download the layer archive that's valid for 10 minutes.

Request Syntax

```
GET /2018-10-31/layers?find=LayerVersion&Arn=Arn HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[Arn \(p. 472\)](#)

The ARN of the layer version.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "CompatibleRuntimes": [ "string" ],
    "Content": {
        "CodeSha256": "string",
        "CodeSize": number,
        "Location": "string"
    },
    "CreatedDate": "string",
    "Description": "string",
    "LayerArn": "string",
    "LayerVersionArn": "string",
    "LicenseInfo": "string",
    "Version": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[CompatibleRuntimes \(p. 472\)](#)

The layer's compatible runtimes.

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

[Content \(p. 472\)](#)

Details about the layer version.

Type: [LayerVersionContentOutput \(p. 600\)](#) object

[CreatedDate \(p. 472\)](#)

The date that the layer version was created, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Description \(p. 472\)](#)

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[LayerArn \(p. 472\)](#)

The ARN of the layer.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+

[LayerVersionArn \(p. 472\)](#)

The ARN of the layer version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

[LicenseInfo \(p. 472\)](#)

The layer's software license.

Type: String

Length Constraints: Maximum length of 512.

[Version \(p. 472\)](#)

The version number.

Type: Long

Errors

InvalidArgumentException

One of the parameters in the request is invalid.

HTTP Status Code: 400
ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetLayerVersionPolicy

Returns the permission policy for a version of an AWS Lambda layer. For more information, see [AddLayerVersionPermission \(p. 404\)](#).

Request Syntax

```
GET /2018-10-31/layers/LayerName/versions/VersionNumber/policy HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[LayerName \(p. 475\)](#)

The name or Amazon Resource Name (ARN) of the layer.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\:\d{12}:layer:[a-zA-Z0-9-_+])|([a-zA-Z0-9-_]+)

[VersionNumber \(p. 475\)](#)

The version number.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "Policy": "string",
    "RevisionId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Policy \(p. 475\)](#)

The policy document.

Type: String

[RevisionId \(p. 475\)](#)

A unique identifier for the current revision of the policy.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetPolicy

Returns the resource-based IAM policy for a function, version, or alias.

Request Syntax

```
GET /2015-03-31/functions/FunctionName/policy?Qualifier=Qualifier HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 477)

The name of the Lambda function, version, or alias.

Name formats

- Function name - my-function (name-only), my-function:v1 (with alias).
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

Qualifier (p. 477)

Specify a version or alias to get the policy for that resource.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$_.-]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "Policy": "string",
    "RevisionId": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Policy \(p. 477\)](#)

The resource-based policy.

Type: String

[RevisionId \(p. 477\)](#)

A unique identifier for the current revision of the policy.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

[TooManyRequestsException](#)

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

GetProvisionedConcurrencyConfig

Retrieves the provisioned concurrency configuration for a function's alias or version.

Request Syntax

```
GET /2019-09-30/functions/FunctionName/provisioned-concurrency?Qualifier=Qualifier HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 479)

The name of the Lambda function.

Name formats

- Function name - my-function.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

Qualifier (p. 479)

The version number or alias name.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "AllocatedProvisionedConcurrentExecutions": number,
    "AvailableProvisionedConcurrentExecutions": number,
    "LastModified": "string",
    "RequestedProvisionedConcurrentExecutions": number,
    "Status": "string",
    "StatusReason": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[AllocatedProvisionedConcurrentExecutions \(p. 479\)](#)

The amount of provisioned concurrency allocated.

Type: Integer

Valid Range: Minimum value of 0.

[AvailableProvisionedConcurrentExecutions \(p. 479\)](#)

The amount of provisioned concurrency available.

Type: Integer

Valid Range: Minimum value of 0.

[LastModified \(p. 479\)](#)

The date and time that a user last updated the configuration, in [ISO 8601 format](#).

Type: String

[RequestedProvisionedConcurrentExecutions \(p. 479\)](#)

The amount of provisioned concurrency requested.

Type: Integer

Valid Range: Minimum value of 1.

[Status \(p. 479\)](#)

The status of the allocation process.

Type: String

Valid Values: IN_PROGRESS | READY | FAILED

[StatusReason \(p. 479\)](#)

For failed allocations, the reason that provisioned concurrency could not be allocated.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ProvisionedConcurrencyConfigNotFoundException](#)

The specified configuration does not exist.

HTTP Status Code: 404

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

Invoke

Invokes a Lambda function. You can invoke a function synchronously (and wait for the response), or asynchronously. To invoke a function asynchronously, set `InvocationType` to `Event`.

For [synchronous invocation](#), details about the function response, including errors, are included in the response body and headers. For either invocation type, you can find more information in the [execution log](#) and [trace](#).

When an error occurs, your function may be invoked multiple times. Retry behavior varies by error type, client, event source, and invocation type. For example, if you invoke a function asynchronously and it returns an error, Lambda executes the function up to two more times. For more information, see [Retry Behavior](#).

For [asynchronous invocation](#), Lambda adds events to a queue before sending them to your function. If your function does not have enough capacity to keep up with the queue, events may be lost. Occasionally, your function may receive the same event multiple times, even if no error occurs. To retain events that were not processed, configure your function with a [dead-letter queue](#).

The status code in the API response doesn't reflect function errors. Error codes are reserved for errors that prevent your function from executing, such as permissions errors, [limit errors](#), or issues with your function's code and configuration. For example, Lambda returns `TooManyRequestsException` if executing the function would cause you to exceed a concurrency limit at either the account level (`ConcurrentInvocationLimitExceeded`) or function level (`ReservedFunctionConcurrentInvocationLimitExceeded`).

For functions with a long timeout, your client might be disconnected during synchronous invocation while it waits for a response. Configure your HTTP client, SDK, firewall, proxy, or operating system to allow for long connections with timeout or keep-alive settings.

This operation requires permission for the `lambda:InvokeFunction` action.

Request Syntax

```
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
```

URI Request Parameters

The request requires the following URI parameters.

[ClientContext](#) (p. 482)

Up to 3583 bytes of base64-encoded data about the invoking client to pass to the function in the `context` object.

[FunctionName](#) (p. 482)

The name of the Lambda function, version, or alias.

Name formats

- Function name - `my-function` (name-only), `my-function:v1` (with alias).
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.

- Partial ARN - 123456789012:function:my-function.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 170.

Pattern: (arn:(aws[a-zA-Z-]*)?:lambda:)?([a-z]{2}(-gov)?-[a-zA-Z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\\$LATEST|[a-zA-Z0-9-_]+))?

[InvocationType \(p. 482\)](#)

Choose from the following options.

- RequestResponse (default) - Invoke the function synchronously. Keep the connection open until the function returns a response or times out. The API response includes the function response and additional data.
- Event - Invoke the function asynchronously. Send events that fail multiple times to the function's dead-letter queue (if it's configured). The API response only includes a status code.
- DryRun - Validate parameter values and verify that the user or role has permission to invoke the function.

Valid Values: Event | RequestResponse | DryRun

[LogType \(p. 482\)](#)

Set to Tail to include the execution log in the response.

Valid Values: None | Tail

[Qualifier \(p. 482\)](#)

Specify a version or alias to invoke a published version of the function.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Request Body

The request accepts the following binary data.

[Payload \(p. 482\)](#)

The JSON that you want to provide to your Lambda function as input.

Response Syntax

```
HTTP/1.1 $statusCode
X-Amz-Function-Error: $FunctionError
X-Amz-Log-Result: $LogResult
X-Amz-Executed-Version: $ExecutedVersion

$Payload
```

Response Elements

If the action is successful, the service sends back the following HTTP response.

[StatusCode \(p. 483\)](#)

The HTTP status code is in the 200 range for a successful request. For the RequestResponse invocation type, this status code is 200. For the Event invocation type, this status code is 202. For the DryRun invocation type, the status code is 204.

The response returns the following HTTP headers.

[ExecutedVersion \(p. 483\)](#)

The version of the function that executed. When you invoke a function with an alias, this indicates which version the alias resolved to.

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[FunctionError \(p. 483\)](#)

If present, indicates that an error occurred during function execution. Details about the error are included in the response payload.

[LogResult \(p. 483\)](#)

The last 4 KB of the execution log, which is base64 encoded.

The response returns the following as the HTTP body.

[Payload \(p. 483\)](#)

The response from the function, or an error object.

Errors

EC2AccessDeniedException

Need additional permissions to configure VPC settings.

HTTP Status Code: 502

EC2ThrottledException

AWS Lambda was throttled by Amazon EC2 during Lambda function initialization using the execution role provided for the Lambda function.

HTTP Status Code: 502

EC2UnexpectedException

AWS Lambda received an unexpected EC2 client exception while setting up for the Lambda function.

HTTP Status Code: 502

ENILimitReachedException

AWS Lambda was not able to create an elastic network interface in the VPC, specified as part of Lambda function configuration, because the limit for network interfaces has been reached.

HTTP Status Code: 502

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

InvalidRequestContentException

The request body could not be parsed as JSON.

HTTP Status Code: 400

InvalidRuntimeException

The runtime or runtime version specified is not supported.

HTTP Status Code: 502

InvalidSecurityGroupIDException

The Security Group ID provided in the Lambda function VPC configuration is invalid.

HTTP Status Code: 502

InvalidSubnetIDException

The Subnet ID provided in the Lambda function VPC configuration is invalid.

HTTP Status Code: 502

InvalidZipFileException

AWS Lambda could not unzip the deployment package.

HTTP Status Code: 502

KMSAccessDeniedException

Lambda was unable to decrypt the environment variables because KMS access was denied. Check the Lambda function's KMS permissions.

HTTP Status Code: 502

KMSDisabledException

Lambda was unable to decrypt the environment variables because the KMS key used is disabled. Check the Lambda function's KMS key settings.

HTTP Status Code: 502

KMSInternalServerError

Lambda was unable to decrypt the environment variables because the KMS key used is in an invalid state for Decrypt. Check the function's KMS key settings.

HTTP Status Code: 502

KMSNotFoundException

Lambda was unable to decrypt the environment variables because the KMS key was not found. Check the function's KMS key settings.

HTTP Status Code: 502

RequestTooLargeException

The request payload exceeded the `Invoke` request body JSON input limit. For more information, see [Limits](#).

HTTP Status Code: 413

ResourceConflictException

The resource already exists, or another operation is in progress.

- HTTP Status Code: 409
`ResourceNotFoundException`

The resource specified in the request does not exist.
- HTTP Status Code: 404
`ResourceNotReadyException`

The function is inactive and its VPC connection is no longer available. Wait for the VPC connection to reestablish and try again.
- HTTP Status Code: 502
`ServiceException`

The AWS Lambda service encountered an internal error.
- HTTP Status Code: 500
`SubnetIPAddressLimitReachedException`

AWS Lambda was not able to set up VPC access for the Lambda function because one or more configured subnets has no available IP addresses.
- HTTP Status Code: 502
`TooManyRequestsException`

The request throughput limit was exceeded.
- HTTP Status Code: 429
`UnsupportedMediaTypeException`

The content type of the `Invoke` request body is not JSON.
- HTTP Status Code: 415

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

InvokeAsync

This action has been deprecated.

Important

For asynchronous function invocation, use [Invoke \(p. 482\)](#).

Invokes a function asynchronously.

Request Syntax

```
POST /2014-11-13/functions/FunctionName/invoke-async/ HTTP/1.1
```

```
InvokeArgs
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 487)

The name of the Lambda function.

Name formats

- Function name - my-function.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

Request Body

The request accepts the following binary data.

InvokeArgs (p. 487)

The JSON that you want to provide to your Lambda function as input.

Response Syntax

```
HTTP/1.1 Status
```

Response Elements

If the action is successful, the service sends back the following HTTP response.

[Status \(p. 487\)](#)

The status code.

Errors

InvalidRequestContentException

The request body could not be parsed as JSON.

HTTP Status Code: 400

InvalidRuntimeException

The runtime or runtime version specified is not supported.

HTTP Status Code: 502

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListAliases

Returns a list of [aliases](#) for a Lambda function.

Request Syntax

```
GET /2015-03-31/functions/FunctionName/aliases?  
FunctionVersion=FunctionVersion&Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 489)

The name of the Lambda function.

Name formats

- Function name - `MyFunction`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- Partial ARN - `123456789012:function:MyFunction`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?`

[FunctionVersion](#) (p. 489)

Specify a function version to only list aliases that invoke that version.

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[Marker](#) (p. 489)

Specify the pagination token that's returned by a previous request to retrieve the next page of results.

[MaxItems](#) (p. 489)

Limit the number of aliases returned.

Valid Range: Minimum value of 1. Maximum value of 10000.

Request Body

The request does not have a request body.

Response Syntax

```
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"  
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Aliases \(p. 489\)](#)

A list of aliases.

Type: Array of [AliasConfiguration \(p. 576\)](#) objects

[NextMarker \(p. 489\)](#)

The pagination token that's included if more results are available.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

[TooManyRequestsException](#)

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListEventSourceMappings

Lists event source mappings. Specify an `EventSourceArn` to only show event source mappings for a single event source.

Request Syntax

```
GET /2015-03-31/event-source-mappings/?  
EventSourceArn=EventSourceArn&FunctionName=FunctionName&Marker=Marker&MaxItems=MaxItems  
HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[EventSourceArn \(p. 492\)](#)

The Amazon Resource Name (ARN) of the event source.

- Amazon Kinesis - The ARN of the data stream or a stream consumer.
- Amazon DynamoDB Streams - The ARN of the stream.
- Amazon Simple Queue Service - The ARN of the queue.

Pattern: `arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9-]+)([a-z]{2}(-gov)?-[a-z]+-\d{1})?:(\d{12})?:(.*?)`

[FunctionName \(p. 492\)](#)

The name of the Lambda function.

Name formats

- Function name - `MyFunction`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- Version or Alias ARN - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction:PROD`.
- Partial ARN - `123456789012:function:MyFunction`.

The length constraint applies only to the full ARN. If you specify only the function name, it's limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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 \(p. 492\)](#)

A pagination token returned by a previous call.

[MaxItems \(p. 492\)](#)

The maximum number of event source mappings to return.

Valid Range: Minimum value of 1. Maximum value of 10000.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "EventSourceMappings": [
        {
            "BatchSize": number,
            "BisectBatchOnFunctionError": boolean,
            "DestinationConfig": {
                "OnFailure": {
                    "Destination": "string"
                },
                "OnSuccess": {
                    "Destination": "string"
                }
            },
            "EventSourceArn": "string",
            "FunctionArn": "string",
            "LastModified": number,
            "LastProcessingResult": "string",
            "MaximumBatchingWindowInSeconds": number,
            "MaximumRecordAgeInSeconds": number,
            "MaximumRetryAttempts": number,
            "ParallelizationFactor": number,
            "State": "string",
            "StateTransitionReason": "string",
            "UUID": "string"
        }
    ],
    "NextMarker": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[EventSourceMappings \(p. 493\)](#)

A list of event source mappings.

Type: Array of [EventSourceMappingConfiguration \(p. 585\)](#) objects

[NextMarker \(p. 493\)](#)

A pagination token that's returned when the response doesn't contain all event source mappings.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource specified in the request does not exist.

HTTP Status Code: 404
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListFunctionEventInvokeConfigs

Retrieves a list of configurations for asynchronous invocation for a function.

To configure options for asynchronous invocation, use [PutFunctionEventInvokeConfig \(p. 528\)](#).

Request Syntax

```
GET /2019-09-25/functions/FunctionName/event-invoke-config/list?  
Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 495\)](#)

The name of the Lambda function.

Name formats

- Function name - my-function.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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 \(p. 495\)](#)

Specify the pagination token that's returned by a previous request to retrieve the next page of results.

[MaxItems \(p. 495\)](#)

The maximum number of configurations to return.

Valid Range: Minimum value of 1. Maximum value of 50.

Request Body

The request does not have a request body.

Response Syntax

```
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"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[FunctionEventInvokeConfigs \(p. 495\)](#)

A list of configurations.

Type: Array of [FunctionEventInvokeConfig \(p. 595\)](#) objects

[NextMarker \(p. 495\)](#)

The pagination token that's included if more results are available.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

[TooManyRequestsException](#)

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListFunctions

Returns a list of Lambda functions, with the version-specific configuration of each. Lambda returns up to 50 functions per call.

Set `FunctionVersion` to `ALL` to include all published versions of each function in addition to the unpublished version. To get more information about a function or version, use [GetFunction \(p. 455\)](#).

Request Syntax

```
GET /2015-03-31/functions/?  
FunctionVersion=FunctionVersion&Marker=Marker&MasterRegion=MasterRegion&MaxItems=MaxItems  
HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionVersion \(p. 498\)](#)

Set to `ALL` to include entries for all published versions of each function.

Valid Values: `ALL`

[Marker \(p. 498\)](#)

Specify the pagination token that's returned by a previous request to retrieve the next page of results.

[MasterRegion \(p. 498\)](#)

For Lambda@Edge functions, the AWS Region of the master function. For example, `us-east-1` filters the list of functions to only include Lambda@Edge functions replicated from a master function in US East (N. Virginia). If specified, you must set `FunctionVersion` to `ALL`.

Pattern: `ALL | [a-z]{2}(-gov)?-[a-z]+-\d{1}`

[MaxItems \(p. 498\)](#)

The maximum number of functions to return.

Valid Range: Minimum value of 1. Maximum value of 10000.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200  
Content-type: application/json  
  
{  
    "Functions": [  
        {  
            "CodeSha256": "string",  
            "CodeSize": number,  
            "DeadLetterConfig": {  
                "TargetArn": "string"  
            },  
            "Description": "string",  
            "FunctionArn": "string",  
            "FunctionName": "string",  
            "Handler": "string",  
            "LastModified": "string",  
            "MemorySize": number,  
            "Runtime": "string",  
            "Timeout": number  
        }  
    ]  
}
```

```
"Description": "string",
"Environment": {
    "Error": {
        "ErrorCode": "string",
        "Message": "string"
    },
    "Variables": {
        "string" : "string"
    }
},
"FunctionArn": "string",
"FunctionName": "string",
"Handler": "string",
"KMSKeyArn": "string",
"LastModified": "string",
"LastUpdateStatus": "string",
"LastUpdateStatusReason": "string",
"LastUpdateStatusReasonCode": "string",
"Layers": [
    {
        "Arn": "string",
        "CodeSize": number
    }
],
"MasterArn": "string",
"MemorySize": number,
"RevisionId": "string",
"Role": "string",
"Runtime": "string",
"State": "string",
"StateReason": "string",
"StateReasonCode": "string",
"Timeout": number,
"TracingConfig": {
    "Mode": "string"
},
"Version": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
},
"NextMarker": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Functions \(p. 498\)](#)

A list of Lambda functions.

Type: Array of [FunctionConfiguration \(p. 590\)](#) objects

[NextMarker \(p. 498\)](#)

The pagination token that's included if more results are available.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListLayers

Lists AWS Lambda layers and shows information about the latest version of each. Specify a runtime identifier to list only layers that indicate that they're compatible with that runtime.

Request Syntax

```
GET /2018-10-31/layers?CompatibleRuntime=CompatibleRuntime&Marker=Marker&MaxItems=MaxItems
HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

CompatibleRuntime (p. 501)

A runtime identifier. For example, go1.x.

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

Marker (p. 501)

A pagination token returned by a previous call.

MaxItems (p. 501)

The maximum number of layers to return.

Valid Range: Minimum value of 1. Maximum value of 50.

Request Body

The request does not have a request body.

Response Syntax

```
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"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Layers \(p. 501\)](#)

A list of function layers.

Type: Array of [LayersListItem \(p. 598\)](#) objects

[NextMarker \(p. 501\)](#)

A pagination token returned when the response doesn't contain all layers.

Type: String

Errors

`InvalidParameterValueException`

One of the parameters in the request is invalid.

HTTP Status Code: 400

`ServiceException`

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

`TooManyRequestsException`

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListLayerVersions

Lists the versions of an [AWS Lambda layer](#). Versions that have been deleted aren't listed. Specify a [runtime identifier](#) to list only versions that indicate that they're compatible with that runtime.

Request Syntax

```
GET /2018-10-31/layers/LayerName/versions?  
CompatibleRuntime=CompatibleRuntime&Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

CompatibleRuntime (p. 503)

A runtime identifier. For example, go1.x.

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

LayerName (p. 503)

The name or Amazon Resource Name (ARN) of the layer.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\:\d{12}:layer:[a-zA-Z0-9-_+])|[a-zA-Z0-9-_+]

Marker (p. 503)

A pagination token returned by a previous call.

MaxItems (p. 503)

The maximum number of versions to return.

Valid Range: Minimum value of 1. Maximum value of 50.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200  
Content-type: application/json  
  
{  
    "LayerVersions": [  
        {  
            "CompatibleRuntimes": [ "string" ],  
            "CreatedDate": "string",  
            "Description": "string",  
            "LayerVersionArn": "string",  
            "LicenseInfo": "string",  
            "Version": number  
        }  
    ]  
}
```

```
        },
    ],
    "NextMarker": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[LayerVersions \(p. 503\)](#)

A list of versions.

Type: Array of [LayerVersionsListItem \(p. 601\)](#) objects

[NextMarker \(p. 503\)](#)

A pagination token returned when the response doesn't contain all versions.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

[TooManyRequestsException](#)

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)

- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListProvisionedConcurrencyConfigs

Retrieves a list of provisioned concurrency configurations for a function.

Request Syntax

```
GET /2019-09-30/functions/FunctionName/provisioned-concurrency?  
List=ALL&Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 506)

The name of the Lambda function.

Name formats

- Function name - `my-function`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- Partial ARN - `123456789012:function:my-function`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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. 506)

Specify the pagination token that's returned by a previous request to retrieve the next page of results.

[MaxItems](#) (p. 506)

Specify a number to limit the number of configurations returned.

Valid Range: Minimum value of 1. Maximum value of 50.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200  
Content-type: application/json  
  
{  
    "NextMarker": "string",  
    "ProvisionedConcurrencyConfigs": [  
        {  
            "AllocatedProvisionedConcurrentExecutions": number,  
            "AvailableProvisionedConcurrentExecutions": number,  
            "FunctionArn": "string",  
            "ProvisionedConcurrencyConfig": {  
                "AllocatedConcurrentExecutions": number,  
                "AvailableConcurrentExecutions": number,  
                "FunctionArn": "string",  
                "LastModified": "2020-01-01T12:00:00Z",  
                "ProvisionedConcurrencyType": "PROVISIONED",  
                "ProvisionedConcurrentExecutions": number,  
                "ProvisioningStatus": "PENDING",  
                "ProvisioningType": "ON_DEMAND",  
                "Status": "PENDING",  
                "StatusReason": "PENDING",  
                "StatusReasonCode": "PENDING",  
                "StatusReasonMessage": "PENDING",  
                "StatusReasonType": "PENDING",  
                "StatusType": "PENDING",  
                "Version": "2020-01-01T12:00:00Z"  
            }  
        }  
    ]  
}
```

```
    "LastModified": "string",
    "RequestedProvisionedConcurrentExecutions": number,
    "Status": "string",
    "StatusReason": "string"
}
]
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[NextMarker \(p. 506\)](#)

The pagination token that's included if more results are available.

Type: String

[ProvisionedConcurrencyConfigs \(p. 506\)](#)

A list of provisioned concurrency configurations.

Type: Array of [ProvisionedConcurrencyConfigListItem \(p. 605\)](#) objects

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)

- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

ListTags

Returns a function's [tags](#). You can also view tags with [GetFunction \(p. 455\)](#).

Request Syntax

```
GET /2017-03-31/tags/ARN HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[ARN \(p. 509\)](#)

The function's Amazon Resource Name (ARN).

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-_+]))?

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "Tags": [
    {
      "string" : "string"
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[Tags \(p. 509\)](#)

The function's tags.

Type: String to string map

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

ListVersionsByFunction

Returns a list of [versions](#), with the version-specific configuration of each. Lambda returns up to 50 versions per call.

Request Syntax

```
GET /2015-03-31/functions/FunctionName/versions?Marker=Marker&MaxItems=MaxItems HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 511)

The name of the Lambda function.

Name formats

- Function name - MyFunction.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- Partial ARN - 123456789012:function:MyFunction.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

[Marker](#) (p. 511)

Specify the pagination token that's returned by a previous request to retrieve the next page of results.

[MaxItems](#) (p. 511)

The maximum number of versions to return.

Valid Range: Minimum value of 1. Maximum value of 10000.

Request Body

The request does not have a request body.

Response Syntax

```
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"
            }
        },
        "FunctionArn": "string",
        "FunctionName": "string",
        "Handler": "string",
        "KMSKeyArn": "string",
        "LastModified": "string",
        "LastUpdateStatus": "string",
        "LastUpdateStatusReason": "string",
        "LastUpdateStatusReasonCode": "string",
        "Layers": [
            {
                "Arn": "string",
                "CodeSize": number
            }
        ],
        "MasterArn": "string",
        "MemorySize": number,
        "RevisionId": "string",
        "Role": "string",
        "Runtime": "string",
        "State": "string",
        "StateReason": "string",
        "StateReasonCode": "string",
        "Timeout": number,
        "TracingConfig": {
            "Mode": "string"
        },
        "Version": "string",
        "VpcConfig": {
            "SecurityGroupIds": [ "string" ],
            "SubnetIds": [ "string" ],
            "VpcId": "string"
        }
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[NextMarker \(p. 511\)](#)

The pagination token that's included if more results are available.

Type: String

[Versions \(p. 511\)](#)

A list of Lambda function versions.

Type: Array of [FunctionConfiguration \(p. 590\)](#) objects

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

PublishLayerVersion

Creates an [AWS Lambda layer](#) from a ZIP archive. Each time you call `PublishLayerVersion` with the same layer name, a new version is created.

Add layers to your function with [CreateFunction \(p. 421\)](#) or [UpdateFunctionConfiguration \(p. 560\)](#).

Request Syntax

```
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"
}
```

URI Request Parameters

The request requires the following URI parameters.

[LayerName \(p. 514\)](#)

The name or Amazon Resource Name (ARN) of the layer.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (`arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\:\d{12}:layer:[a-zA-Z0-9-_+]|[a-zA-Z0-9-_+]`)

Request Body

The request accepts the following data in JSON format.

[CompatibleRuntimes \(p. 514\)](#)

A list of compatible [function runtimes](#). Used for filtering with [ListLayers \(p. 501\)](#) and [ListLayerVersions \(p. 503\)](#).

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: `nodejs10.x` | `nodejs12.x` | `java8` | `java11` | `python2.7` | `python3.6` | `python3.7` | `python3.8` | `dotnetcore2.1` | `dotnetcore3.1` | `go1.x` | `ruby2.5` | `ruby2.7` | `provided`

Required: No

[Content \(p. 514\)](#)

The function layer archive.

Type: [LayerVersionContentInput \(p. 599\)](#) object

Required: Yes

[Description \(p. 514\)](#)

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[LicenseInfo \(p. 514\)](#)

The layer's software license. It can be any of the following:

- An [SPDX license identifier](#). For example, `MIT`.
- The URL of a license hosted on the internet. For example, <https://opensource.org/licenses/MIT>.
- The full text of the license.

Type: String

Length Constraints: Maximum length of 512.

Required: No

Response Syntax

```
HTTP/1.1 201
Content-type: application/json

{
    "CompatibleRuntimes": [ "string" ],
    "Content": {
        "CodeSha256": "string",
        "CodeSize": number,
        "Location": "string"
    },
    "CreatedDate": "string",
    "Description": "string",
    "LayerArn": "string",
    "LayerVersionArn": "string",
    "LicenseInfo": "string",
    "Version": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

[CompatibleRuntimes \(p. 515\)](#)

The layer's compatible runtimes.

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

[Content \(p. 515\)](#)

Details about the layer version.

Type: [LayerVersionContentOutput \(p. 600\)](#) object

[CreatedDate \(p. 515\)](#)

The date that the layer version was created, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[Description \(p. 515\)](#)

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[LayerArn \(p. 515\)](#)

The ARN of the layer.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+

[LayerVersionArn \(p. 515\)](#)

The ARN of the layer version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

[LicenseInfo \(p. 515\)](#)

The layer's software license.

Type: String

Length Constraints: Maximum length of 512.

[Version \(p. 515\)](#)

The version number.

Type: Long

Errors

CodeStorageExceededException

You have exceeded your maximum total code size per account. [Learn more](#)

HTTP Status Code: 400
InvalidParameterValueException
One of the parameters in the request is invalid.
HTTP Status Code: 400
ResourceNotFoundException
The resource specified in the request does not exist.
HTTP Status Code: 404
ServiceException
The AWS Lambda service encountered an internal error.
HTTP Status Code: 500
TooManyRequestsException
The request throughput limit was exceeded.
HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

PublishVersion

Creates a [version](#) from the current code and configuration of a function. Use versions to create a snapshot of your function code and configuration that doesn't change.

AWS Lambda doesn't publish a version if the function's configuration and code haven't changed since the last version. Use [UpdateFunctionCode \(p. 553\)](#) or [UpdateFunctionConfiguration \(p. 560\)](#) to update the function before publishing a version.

Clients can invoke versions directly or with an alias. To create an alias, use [CreateAlias \(p. 411\)](#).

Request Syntax

```
POST /2015-03-31/functions/FunctionName/versions HTTP/1.1
Content-type: application/json

{
  "CodeSha256": "string",
  "Description": "string",
  "RevisionId": "string"
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 518\)](#)

The name of the Lambda function.

Name formats

- Function name - MyFunction.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- Partial ARN - 123456789012:function:MyFunction.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

Request Body

The request accepts the following data in JSON format.

[CodeSha256 \(p. 518\)](#)

Only publish a version if the hash value matches the value that's specified. Use this option to avoid publishing a version if the function code has changed since you last updated it. You can get the hash for the version that you uploaded from the output of [UpdateFunctionCode \(p. 553\)](#).

Type: String

Required: No

Description (p. 518)

A description for the version to override the description in the function configuration.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

RevisionId (p. 518)

Only update the function if the revision ID matches the ID that's specified. Use this option to avoid publishing a version if the function configuration has changed since you last updated it.

Type: String

Required: No

Response Syntax

```
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"
        }
    },
    "FunctionArn": "string",
    "FunctionName": "string",
    "Handler": "string",
    "KMSKeyArn": "string",
    "LastModified": "string",
    "LastUpdateStatus": "string",
    "LastUpdateStatusReason": "string",
    "LastUpdateStatusReasonCode": "string",
    "Layers": [
        {
            "Arn": "string",
            "CodeSize": number
        }
    ],
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "string",
    "State": "string",
    "StateReason": "string",
    "StateReasonCode": "string",
    "Timeout": number,
```

```
"TracingConfig": {  
    "Mode": "string"  
},  
"Version": "string",  
"VpcConfig": {  
    "SecurityGroupIds": [ "string" ],  
    "SubnetIds": [ "string" ],  
    "VpcId": "string"  
}  
}
```

Response Elements

If the action is successful, the service sends back an HTTP 201 response.

The following data is returned in JSON format by the service.

[CodeSha256 \(p. 519\)](#)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize \(p. 519\)](#)

The size of the function's deployment package, in bytes.

Type: Long

[DeadLetterConfig \(p. 519\)](#)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 580\)](#) object

[Description \(p. 519\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 519\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 584\)](#) object

[FunctionArn \(p. 519\)](#)

The function's Amazon Resource Name (ARN).

Type: String

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 \(p. 519\)](#)

The name of the function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 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 \(p. 519\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`
[KMSKeyArn \(p. 519\)](#)

The KMS key that's used to encrypt the function's environment variables. This key is only returned if you've configured a customer managed CMK.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:\.*|()`
[LastModified \(p. 519\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[LastUpdateStatus \(p. 519\)](#)

The status of the last update that was performed on the function. This is first set to `Successful` after function creation completes.

Type: String

Valid Values: `Successful` | `Failed` | `InProgress`

[LastUpdateStatusReason \(p. 519\)](#)

The reason for the last update that was performed on the function.

Type: String

[LastUpdateStatusReasonCode \(p. 519\)](#)

The reason code for the last update that was performed on the function.

Type: String

Valid Values: `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup`

[Layers \(p. 519\)](#)

The function's [layers](#).

Type: Array of [Layer \(p. 597\)](#) objects

[MasterArn \(p. 519\)](#)

For Lambda@Edge functions, the ARN of the master function.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:lambda:[a-zA-Z]{2}(-gov)?-[a-zA-Z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$\$LATEST|[a-zA-Z0-9-_]+))?`

[MemorySize \(p. 519\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 519\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 519\)](#)

The function's execution role.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/[a-zA-Z_0-9+=,.@\\-/]+`

[Runtime \(p. 519\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: `nodejs10.x` | `nodejs12.x` | `java8` | `java11` | `python2.7` | `python3.6` | `python3.7` | `python3.8` | `dotnetcore2.1` | `dotnetcore3.1` | `go1.x` | `ruby2.5` | `ruby2.7` | `provided`

[State \(p. 519\)](#)

The current state of the function. When the state is `Inactive`, you can reactivate the function by invoking it.

Type: String

Valid Values: `Pending` | `Active` | `Inactive` | `Failed`

[StateReason \(p. 519\)](#)

The reason for the function's current state.

Type: String

[StateReasonCode \(p. 519\)](#)

The reason code for the function's current state. When the code is `Creating`, you can't invoke or modify the function.

Type: String

Valid Values: `Idle` | `Creating` | `Restoring` | `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup`

[Timeout \(p. 519\)](#)

The amount of time that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 519\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 608\)](#) object

[Version \(p. 519\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 519\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 610\)](#) object

Errors

CodeStorageExceededException

You have exceeded your maximum total code size per account. [Learn more](#)

HTTP Status Code: 400

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

PreconditionFailedException

The RevisionId provided does not match the latest RevisionId for the Lambda function or alias. Call the GetFunction or the GetAlias API to retrieve the latest RevisionId for your resource.

HTTP Status Code: 412

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

PutFunctionConcurrency

Sets the maximum number of simultaneous executions for a function, and reserves capacity for that concurrency level.

Concurrency settings apply to the function as a whole, including all published versions and the unpublished version. Reserving concurrency both ensures that your function has capacity to process the specified number of events simultaneously, and prevents it from scaling beyond that level. Use [GetFunction \(p. 455\)](#) to see the current setting for a function.

Use [GetAccountSettings \(p. 446\)](#) to see your Regional concurrency limit. You can reserve concurrency for as many functions as you like, as long as you leave at least 100 simultaneous executions unreserved for functions that aren't configured with a per-function limit. For more information, see [Managing Concurrency](#).

Request Syntax

```
PUT /2017-10-31/functions/FunctionName/concurrency HTTP/1.1
Content-type: application/json

{
    "ReservedConcurrentExecutions": number
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 525\)](#)

The name of the Lambda function.

Name formats

- Function name - my-function.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

Request Body

The request accepts the following data in JSON format.

[ReservedConcurrentExecutions \(p. 525\)](#)

The number of simultaneous executions to reserve for the function.

Type: Integer

Valid Range: Minimum value of 0.

Required: Yes

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "ReservedConcurrentExecutions": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[ReservedConcurrentExecutions \(p. 526\)](#)

The number of concurrent executions that are reserved for this function. For more information, see [Managing Concurrency](#).

Type: Integer

Valid Range: Minimum value of 0.

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceConflictException](#)

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

[ResourceNotFoundException](#)

The resource specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

[TooManyRequestsException](#)

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

PutFunctionEventInvokeConfig

Configures options for [asynchronous invocation](#) on a function, version, or alias. If a configuration already exists for a function, version, or alias, this operation overwrites it. If you exclude any settings, they are removed. To set one option without affecting existing settings for other options, use [PutFunctionEventInvokeConfig \(p. 528\)](#).

By default, Lambda retries an asynchronous invocation twice if the function returns an error. It retains events in a queue for up to six hours. When an event fails all processing attempts or stays in the asynchronous invocation queue for too long, Lambda discards it. To retain discarded events, configure a dead-letter queue with [UpdateFunctionConfiguration \(p. 560\)](#).

To send an invocation record to a queue, topic, function, or event bus, specify a [destination](#). You can configure separate destinations for successful invocations (on-success) and events that fail all processing attempts (on-failure). You can configure destinations in addition to or instead of a dead-letter queue.

Request Syntax

```
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
}
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 528)

The name of the Lambda function, version, or alias.

Name formats

- Function name - my-function (name-only), my-function:v1 (with alias).
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_+]))?

Qualifier (p. 528)

A version number or alias name.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (| [a-zA-Z0-9\$_-]+)

Request Body

The request accepts the following data in JSON format.

[DestinationConfig \(p. 528\)](#)

A destination for events after they have been sent to a function for processing.

Destinations

- Function - The Amazon Resource Name (ARN) of a Lambda function.
- Queue - The ARN of an SQS queue.
- Topic - The ARN of an SNS topic.
- Event Bus - The ARN of an Amazon EventBridge event bus.

Type: [DestinationConfig \(p. 581\)](#) object

Required: No

[MaximumEventAgeInSeconds \(p. 528\)](#)

The maximum age of a request that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 21600.

Required: No

[MaximumRetryAttempts \(p. 528\)](#)

The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 2.

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "FunctionArn": "string",
    "LastModified": number,
```

```
"MaximumEventAgeInSeconds": number,  
"MaximumRetryAttempts": number  
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[DestinationConfig \(p. 529\)](#)

A destination for events after they have been sent to a function for processing.

Destinations

- Function - The Amazon Resource Name (ARN) of a Lambda function.
- Queue - The ARN of an SQS queue.
- Topic - The ARN of an SNS topic.
- Event Bus - The ARN of an Amazon EventBridge event bus.

Type: [DestinationConfig \(p. 581\)](#) object

[FunctionArn \(p. 529\)](#)

The Amazon Resource Name (ARN) of the function.

Type: String

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 \(p. 529\)](#)

The date and time that the configuration was last updated, in Unix time seconds.

Type: Timestamp

[MaximumEventAgeInSeconds \(p. 529\)](#)

The maximum age of a request that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 21600.

[MaximumRetryAttempts \(p. 529\)](#)

The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 2.

Errors

InvalidArgumentException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

PutProvisionedConcurrencyConfig

Adds a provisioned concurrency configuration to a function's alias or version.

Request Syntax

```
PUT /2019-09-30/functions/FunctionName/provisioned-concurrency?Qualifier=Qualifier HTTP/1.1
Content-type: application/json

{
    "ProvisionedConcurrentExecutions": number
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 532\)](#)

The name of the Lambda function.

Name formats

- Function name - my-function.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

[Qualifier \(p. 532\)](#)

The version number or alias name.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-_]+)

Request Body

The request accepts the following data in JSON format.

[ProvisionedConcurrentExecutions \(p. 532\)](#)

The amount of provisioned concurrency to allocate for the version or alias.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
    "AllocatedProvisionedConcurrentExecutions": number,
    "AvailableProvisionedConcurrentExecutions": number,
    "LastModified": "string",
    "RequestedProvisionedConcurrentExecutions": number,
    "Status": "string",
    "StatusReason": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

[AllocatedProvisionedConcurrentExecutions \(p. 533\)](#)

The amount of provisioned concurrency allocated.

Type: Integer

Valid Range: Minimum value of 0.

[AvailableProvisionedConcurrentExecutions \(p. 533\)](#)

The amount of provisioned concurrency available.

Type: Integer

Valid Range: Minimum value of 0.

[LastModified \(p. 533\)](#)

The date and time that a user last updated the configuration, in [ISO 8601](#) format.

Type: String

[RequestedProvisionedConcurrentExecutions \(p. 533\)](#)

The amount of provisioned concurrency requested.

Type: Integer

Valid Range: Minimum value of 1.

[Status \(p. 533\)](#)

The status of the allocation process.

Type: String

Valid Values: IN_PROGRESS | READY | FAILED

[StatusReason \(p. 533\)](#)

For failed allocations, the reason that provisioned concurrency could not be allocated.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

RemoveLayerVersionPermission

Removes a statement from the permissions policy for a version of an AWS Lambda layer. For more information, see [AddLayerVersionPermission \(p. 404\)](#).

Request Syntax

```
DELETE /2018-10-31/layers/LayerName/versions/VersionNumber/policy/StatementId?  
RevisionId=RevisionId HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[LayerName \(p. 535\)](#)

The name or Amazon Resource Name (ARN) of the layer.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_+])|([a-zA-Z0-9-_]+)

[RevisionId \(p. 535\)](#)

Only update the policy if the revision ID matches the ID specified. Use this option to avoid modifying a policy that has changed since you last read it.

[StatementId \(p. 535\)](#)

The identifier that was specified when the statement was added.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ([a-zA-Z0-9-_]+)

[VersionNumber \(p. 535\)](#)

The version number.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

PreconditionFailedException

The RevisionId provided does not match the latest RevisionId for the Lambda function or alias. Call the GetFunction or the GetAlias API to retrieve the latest RevisionId for your resource.

HTTP Status Code: 412

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

RemovePermission

Revokes function-use permission from an AWS service or another account. You can get the ID of the statement from the output of [GetPolicy \(p. 477\)](#).

Request Syntax

```
DELETE /2015-03-31/functions/FunctionName/policy/StatementId?  
Qualifier=Qualifier&RevisionId=RevisionId HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 537\)](#)

The name of the Lambda function, version, or alias.

Name formats

- Function name - `my-function` (name-only), `my-function:v1` (with alias).
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- Partial ARN - `123456789012:function:my-function`.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

[Qualifier \(p. 537\)](#)

Specify a version or alias to remove permissions from a published version of the function.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-_]+)

[RevisionId \(p. 537\)](#)

Only update the policy if the revision ID matches the ID that's specified. Use this option to avoid modifying a policy that has changed since you last read it.

[StatementId \(p. 537\)](#)

Statement ID of the permission to remove.

Length Constraints: Minimum length of 1. Maximum length of 100.

Pattern: ([a-zA-Z0-9-_]+)

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

PreconditionFailedException

The RevisionId provided does not match the latest RevisionId for the Lambda function or alias. Call the GetFunction or the GetAlias API to retrieve the latest RevisionId for your resource.

HTTP Status Code: 412

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

TagResource

Adds [tags](#) to a function.

Request Syntax

```
POST /2017-03-31/tags/ARN HTTP/1.1
Content-type: application/json

{
  "Tags": {
    "string" : "string"
  }
}
```

URI Request Parameters

The request requires the following URI parameters.

[ARN \(p. 539\)](#)

The function's Amazon Resource Name (ARN).

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-_]+))?

Request Body

The request accepts the following data in JSON format.

[Tags \(p. 539\)](#)

A list of tags to apply to the function.

Type: String to string map

Required: Yes

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

UntagResource

Removes [tags](#) from a function.

Request Syntax

```
DELETE /2017-03-31/tags/ARN?tagKeys=TagKeys HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[ARN](#) (p. 541)

The function's Amazon Resource Name (ARN).

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-_]+))?

[TagKeys](#) (p. 541)

A list of tag keys to remove from the function.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceConflictException](#)

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

[ResourceNotFoundException](#)

The resource specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
`TooManyRequestsException`

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

UpdateAlias

Updates the configuration of a Lambda function [alias](#).

Request Syntax

```
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
    }
  }
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 543)

The name of the Lambda function.

Name formats

- Function name - `MyFunction`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:MyFunction`.
- Partial ARN - `123456789012:function:MyFunction`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?`

[Name](#) (p. 543)

The name of the alias.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(?!^[\d-]+$)([a-zA-Z0-9-_]+)`

Request Body

The request accepts the following data in JSON format.

[Description](#) (p. 543)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[FunctionVersion \(p. 543\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

Required: No

[RevisionId \(p. 543\)](#)

Only update the alias if the revision ID matches the ID that's specified. Use this option to avoid modifying an alias that has changed since you last read it.

Type: String

Required: No

[RoutingConfig \(p. 543\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 578\)](#) object

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
    "AliasArn": "string",
    "Description": "string",
    "FunctionVersion": "string",
    "Name": "string",
    "RevisionId": "string",
    "RoutingConfig": {
        "AdditionalVersionWeights": {
            "string" : number
        }
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[AliasArn \(p. 544\)](#)

The Amazon Resource Name (ARN) of the alias.

Type: String

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 \(p. 544\)](#)

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 544\)](#)

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[Name \(p. 544\)](#)

The name of the alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\$)([a-zA-Z0-9-_]+)

[RevisionId \(p. 544\)](#)

A unique identifier that changes when you update the alias.

Type: String

[RoutingConfig \(p. 544\)](#)

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 578\)](#) object

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[PreconditionFailedException](#)

The RevisionId provided does not match the latest RevisionId for the Lambda function or alias. Call the [GetFunction](#) or the [GetAlias](#) API to retrieve the latest RevisionId for your resource.

HTTP Status Code: 412

[ResourceConflictException](#)

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

UpdateEventSourceMapping

Updates an event source mapping. You can change the function that AWS Lambda invokes, or pause invocation and resume later from the same location.

The following error handling options are only available for stream sources (DynamoDB and Kinesis):

- `BisectBatchOnFunctionError` - If the function returns an error, split the batch in two and retry.
- `DestinationConfig` - Send discarded records to an Amazon SQS queue or Amazon SNS topic.
- `MaximumRecordAgeInSeconds` - Discard records older than the specified age.
- `MaximumRetryAttempts` - Discard records after the specified number of retries.
- `ParallelizationFactor` - Process multiple batches from each shard concurrently.

Request Syntax

```
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",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number
}
```

URI Request Parameters

The request requires the following URI parameters.

UUID (p. 547)

The identifier of the event source mapping.

Request Body

The request accepts the following data in JSON format.

BatchSize (p. 547)

The maximum number of items to retrieve in a single batch.

- Amazon Kinesis - Default 100. Max 10,000.
- Amazon DynamoDB Streams - Default 100. Max 1,000.
- Amazon Simple Queue Service - Default 10. Max 10.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

Required: No

[BisectBatchOnFunctionError \(p. 547\)](#)

(Streams) If the function returns an error, split the batch in two and retry.

Type: Boolean

Required: No

[DestinationConfig \(p. 547\)](#)

(Streams) An Amazon SQS queue or Amazon SNS topic destination for discarded records.

Type: [DestinationConfig \(p. 581\)](#) object

Required: No

[Enabled \(p. 547\)](#)

Disables the event source mapping to pause polling and invocation.

Type: Boolean

Required: No

[FunctionName \(p. 547\)](#)

The name of the Lambda function.

Name formats

- Function name - MyFunction.
- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction.
- Version or Alias ARN - arn:aws:lambda:us-west-2:123456789012:function:MyFunction:PROD.
- Partial ARN - 123456789012:function:MyFunction.

The length constraint applies only to the full ARN. If you specify only the function name, it's limited to 64 characters in length.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

Required: No

[MaximumBatchingWindowInSeconds \(p. 547\)](#)

(Streams) The maximum amount of time to gather records before invoking the function, in seconds.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 300.

Required: No

[MaximumRecordAgeInSeconds \(p. 547\)](#)

(Streams) The maximum age of a record that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 604800.

Required: No

[MaximumRetryAttempts \(p. 547\)](#)

(Streams) The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000.

Required: No

[ParallelizationFactor \(p. 547\)](#)

(Streams) The number of batches to process from each shard concurrently.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

Required: No

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
    "BatchSize": number,
    "BisectBatchOnFunctionError": boolean,
    "DestinationConfig": {
        "OnFailure": {
            "Destination": "string"
        },
        "OnSuccess": {
            "Destination": "string"
        }
    },
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "MaximumBatchingWindowInSeconds": number,
    "MaximumRecordAgeInSeconds": number,
    "MaximumRetryAttempts": number,
    "ParallelizationFactor": number,
    "State": "string",
    "StateTransitionReason": "string",
    "UUID": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 202 response.

The following data is returned in JSON format by the service.

[BatchSize \(p. 549\)](#)

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[BisectBatchOnFunctionError \(p. 549\)](#)

(Streams) If the function returns an error, split the batch in two and retry.

Type: Boolean

[DestinationConfig \(p. 549\)](#)

(Streams) An Amazon SQS queue or Amazon SNS topic destination for discarded records.

Type: [DestinationConfig \(p. 581\)](#) object

[EventSourceArn \(p. 549\)](#)

The Amazon Resource Name (ARN) of the event source.

Type: String

Pattern: arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+:([a-z]{2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.*)

[FunctionArn \(p. 549\)](#)

The ARN of the Lambda function.

Type: String

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 \(p. 549\)](#)

The date that the event source mapping was last updated, or its state changed, in Unix time seconds.

Type: Timestamp

[LastProcessingResult \(p. 549\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[MaximumBatchingWindowInSeconds \(p. 549\)](#)

(Streams) The maximum amount of time to gather records before invoking the function, in seconds.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 300.

[MaximumRecordAgeInSeconds \(p. 549\)](#)

(Streams) The maximum age of a record that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 604800.

[MaximumRetryAttempts \(p. 549\)](#)

(Streams) The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000.

[ParallelizationFactor \(p. 549\)](#)

(Streams) The number of batches to process from each shard concurrently.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

[State \(p. 549\)](#)

The state of the event source mapping. It can be one of the following: Creating, Enabling, Enabled, Disabling, Disabled, Updating, or Deleting.

Type: String

[StateTransitionReason \(p. 549\)](#)

Indicates whether the last change to the event source mapping was made by a user, or by the Lambda service.

Type: String

[UUID \(p. 549\)](#)

The identifier of the event source mapping.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid.

HTTP Status Code: 400

[ResourceConflictException](#)

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

[ResourceInUseException](#)

The operation conflicts with the resource's availability. For example, you attempted to update an EventSource Mapping in CREATING, or tried to delete a EventSource mapping currently in the UPDATING state.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource specified in the request does not exist.

HTTP Status Code: 404

[ServiceException](#)

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
`TooManyRequestsException`

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

UpdateFunctionCode

Updates a Lambda function's code.

The function's code is locked when you publish a version. You can't modify the code of a published version, only the unpublished version.

Request Syntax

```
PUT /2015-03-31/functions/FunctionName/code HTTP/1.1
Content-type: application/json

{
  "DryRun": boolean,
  "Publish": boolean,
  "RevisionId": "string",
  "S3Bucket": "string",
  "S3Key": "string",
  "S3ObjectVersion": "string",
  "ZipFile": blob
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 553\)](#)

The name of the Lambda function.

Name formats

- Function name - `my-function`.
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- Partial ARN - `123456789012:function:my-function`.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?`

Request Body

The request accepts the following data in JSON format.

[DryRun \(p. 553\)](#)

Set to true to validate the request parameters and access permissions without modifying the function code.

Type: Boolean

Required: No

[Publish \(p. 553\)](#)

Set to true to publish a new version of the function after updating the code. This has the same effect as calling [PublishVersion \(p. 518\)](#) separately.

Type: Boolean

Required: No

[RevisionId \(p. 553\)](#)

Only update the function if the revision ID matches the ID that's specified. Use this option to avoid modifying a function that has changed since you last read it.

Type: String

Required: No

[S3Bucket \(p. 553\)](#)

An Amazon S3 bucket in the same AWS Region as your function. The bucket can be in a different AWS account.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 63.

Pattern: ^[0-9A-Za-z\.\-_]*(\?<!\.).\$

Required: No

[S3Key \(p. 553\)](#)

The Amazon S3 key of the deployment package.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

[S3ObjectVersion \(p. 553\)](#)

For versioned objects, the version of the deployment package object to use.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

[ZipFile \(p. 553\)](#)

The base64-encoded contents of the deployment package. AWS SDK and AWS CLI clients handle the encoding for you.

Type: Base64-encoded binary data object

Required: No

Response Syntax

```
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"
        }
    },
    "FunctionArn": "string",
    "FunctionName": "string",
    "Handler": "string",
    "KMSKeyArn": "string",
    "LastModified": "string",
    "LastUpdateStatus": "string",
    "LastUpdateStatusReason": "string",
    "LastUpdateStatusReasonCode": "string",
    "Layers": [
        {
            "Arn": "string",
            "CodeSize": number
        }
    ],
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "string",
    "State": "string",
    "StateReason": "string",
    "StateReasonCode": "string",
    "Timeout": number,
    "TracingConfig": {
        "Mode": "string"
    },
    "Version": "string",
    "VpcConfig": {
        "SecurityGroupIds": [ "string" ],
        "SubnetIds": [ "string" ],
        "VpcId": "string"
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[CodeSha256 \(p. 554\)](#)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize \(p. 554\)](#)

The size of the function's deployment package, in bytes.

Type: Long

[DeadLetterConfig \(p. 554\)](#)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 580\)](#) object

[Description \(p. 554\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 554\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 584\)](#) object

[FunctionArn \(p. 554\)](#)

The function's Amazon Resource Name (ARN).

Type: String

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 \(p. 554\)](#)

The name of the function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 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 \(p. 554\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 554\)](#)

The KMS key that's used to encrypt the function's environment variables. This key is only returned if you've configured a customer managed CMK.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+\.*\.)|()`

[LastModified \(p. 554\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[LastUpdateStatus \(p. 554\)](#)

The status of the last update that was performed on the function. This is first set to `Successful` after function creation completes.

Type: String

Valid Values: `Successful` | `Failed` | `InProgress`

[LastUpdateStatusReason \(p. 554\)](#)

The reason for the last update that was performed on the function.

Type: String

[LastUpdateStatusReasonCode \(p. 554\)](#)

The reason code for the last update that was performed on the function.

Type: String

Valid Values: `EniLimitExceeded` | `InsufficientRolePermissions` |
`InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` |
`InvalidSubnet` | `InvalidSecurityGroup`

[Layers \(p. 554\)](#)

The function's [layers](#).

Type: Array of [Layer \(p. 597\)](#) objects

[MasterArn \(p. 554\)](#)

For Lambda@Edge functions, the ARN of the master function.

Type: String

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 \(p. 554\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 554\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 554\)](#)

The function's execution role.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@/-/_]+`

[Runtime \(p. 554\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

[State \(p. 554\)](#)

The current state of the function. When the state is `Inactive`, you can reactivate the function by invoking it.

Type: String

Valid Values: Pending | Active | Inactive | Failed

[StateReason \(p. 554\)](#)

The reason for the function's current state.

Type: String

[StateReasonCode \(p. 554\)](#)

The reason code for the function's current state. When the code is `Creating`, you can't invoke or modify the function.

Type: String

Valid Values: Idle | Creating | Restoring | EniLimitExceeded | InsufficientRolePermissions | InvalidConfiguration | InternalError | SubnetOutOfIPAddresses | InvalidSubnet | InvalidSecurityGroup

[Timeout \(p. 554\)](#)

The amount of time that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 554\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 608\)](#) object

[Version \(p. 554\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 554\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 610\)](#) object

Errors

CodeStorageExceededException

You have exceeded your maximum total code size per account. [Learn more](#)

HTTP Status Code: 400
InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400
PreconditionFailedException

The RevisionId provided does not match the latest RevisionId for the Lambda function or alias. Call the GetFunction or the GetAlias API to retrieve the latest RevisionId for your resource.

HTTP Status Code: 412
ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409
ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404
ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500
TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

UpdateFunctionConfiguration

Modify the version-specific settings of a Lambda function.

When you update a function, Lambda provisions an instance of the function and its supporting resources. If your function connects to a VPC, this process can take a minute. During this time, you can't modify the function, but you can still invoke it. The `LastUpdateStatus`, `LastUpdateStatusReason`, and `LastUpdateStatusReasonCode` fields in the response from [GetFunctionConfiguration \(p. 460\)](#) indicate when the update is complete and the function is processing events with the new configuration. For more information, see [Function States](#).

These settings can vary between versions of a function and are locked when you publish a version. You can't modify the configuration of a published version, only the unpublished version.

To configure function concurrency, use [PutFunctionConcurrency \(p. 525\)](#). To grant invoke permissions to an account or AWS service, use [AddPermission \(p. 407\)](#).

Request Syntax

```
PUT /2015-03-31/functions/FunctionName/configuration HTTP/1.1
Content-type: application/json

{
  "DeadLetterConfig": {
    "TargetArn": "string"
  },
  "Description": "string",
  "Environment": {
    "Variables": {
      "string": "string"
    }
  },
  "Handler": "string",
  "KMSKeyArn": "string",
  "Layers": [ "string" ],
  "MemorySize": number,
  "RevisionId": "string",
  "Role": "string",
  "Runtime": "string",
  "Timeout": number,
  "TracingConfig": {
    "Mode": "string"
  },
  "VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ]
  }
}
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 560)

The name of the Lambda function.

Name formats

- Function name - `my-function`.

- Function ARN - arn:aws:lambda:us-west-2:123456789012:function:my-function.
- Partial ARN - 123456789012:function:my-function.

The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?

Request Body

The request accepts the following data in JSON format.

[DeadLetterConfig \(p. 560\)](#)

A dead letter queue configuration that specifies the queue or topic where Lambda sends asynchronous events when they fail processing. For more information, see [Dead Letter Queues](#).

Type: [DeadLetterConfig \(p. 580\)](#) object

Required: No

[Description \(p. 560\)](#)

A description of the function.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[Environment \(p. 560\)](#)

Environment variables that are accessible from function code during execution.

Type: [Environment \(p. 582\)](#) object

Required: No

[Handler \(p. 560\)](#)

The name of the method within your code that Lambda calls to execute your function. The format includes the file name. It can also include namespaces and other qualifiers, depending on the runtime. For more information, see [Programming Model](#).

Type: String

Length Constraints: Maximum length of 128.

Pattern: [^\s]+

Required: No

[KMSKeyArn \(p. 560\)](#)

The ARN of the AWS Key Management Service (AWS KMS) key that's used to encrypt your function's environment variables. If it's not provided, AWS Lambda uses a default service key.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:[.]*|()|)`

Required: No

[Layers \(p. 560\)](#)

A list of [function layers](#) to add to the function's execution environment. Specify each layer by its ARN, including the version.

Type: Array of strings

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: `arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+`

Required: No

[MemorySize \(p. 560\)](#)

The amount of memory that your function has access to. Increasing the function's memory also increases its CPU allocation. The default value is 128 MB. The value must be a multiple of 64 MB.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

Required: No

[RevisionId \(p. 560\)](#)

Only update the function if the revision ID matches the ID that's specified. Use this option to avoid modifying a function that has changed since you last read it.

Type: String

Required: No

[Role \(p. 560\)](#)

The Amazon Resource Name (ARN) of the function's execution role.

Type: String

Pattern: `arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\\-/]+`

Required: No

[Runtime \(p. 560\)](#)

The identifier of the function's [runtime](#).

Type: String

Valid Values: `nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided`

Required: No

[Timeout \(p. 560\)](#)

The amount of time that Lambda allows a function to run before stopping it. The default is 3 seconds. The maximum allowed value is 900 seconds.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

[TracingConfig \(p. 560\)](#)

Set Mode to Active to sample and trace a subset of incoming requests with AWS X-Ray.

Type: [TracingConfig \(p. 607\)](#) object

Required: No

[VpcConfig \(p. 560\)](#)

For network connectivity to AWS resources in a VPC, specify a list of security groups and subnets in the VPC. When you connect a function to a VPC, it can only access resources and the internet through that VPC. For more information, see [VPC Settings](#).

Type: [VpcConfig \(p. 609\)](#) object

Required: No

Response Syntax

```
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"
        }
    },
    "FunctionArn": "string",
    "FunctionName": "string",
    "Handler": "string",
    "KMSKeyArn": "string",
    "LastModified": "string",
    "LastUpdateStatus": "string",
    "LastUpdateStatusReason": "string",
    "LastUpdateStatusReasonCode": "string",
    "Layers": [
        {
            "Arn": "string",
            "CodeSize": number
        }
    ],
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "string",
    "State": "string",
    "StateReason": "string",
    "Timeout": "string"
}
```

```
"StateReasonCode": "string",
"Timeout": number,
"TracingConfig": {
    "Mode": "string"
},
"Version": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "SubnetIds": [ "string" ],
    "VpcId": "string"
}
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[CodeSha256 \(p. 563\)](#)

The SHA256 hash of the function's deployment package.

Type: String

[CodeSize \(p. 563\)](#)

The size of the function's deployment package, in bytes.

Type: Long

[DeadLetterConfig \(p. 563\)](#)

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 580\)](#) object

[Description \(p. 563\)](#)

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 563\)](#)

The function's environment variables.

Type: [EnvironmentResponse \(p. 584\)](#) object

[FunctionArn \(p. 563\)](#)

The function's Amazon Resource Name (ARN).

Type: String

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 \(p. 563\)](#)

The name of the function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 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 \(p. 563\)](#)

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 563\)](#)

The KMS key that's used to encrypt the function's environment variables. This key is only returned if you've configured a customer managed CMK.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+\.:*)|()`

[LastModified \(p. 563\)](#)

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

[LastUpdateStatus \(p. 563\)](#)

The status of the last update that was performed on the function. This is first set to `Successful` after function creation completes.

Type: String

Valid Values: `Successful` | `Failed` | `InProgress`

[LastUpdateStatusReason \(p. 563\)](#)

The reason for the last update that was performed on the function.

Type: String

[LastUpdateStatusReasonCode \(p. 563\)](#)

The reason code for the last update that was performed on the function.

Type: String

Valid Values: `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup`

[Layers \(p. 563\)](#)

The function's `layers`.

Type: Array of [Layer \(p. 597\)](#) objects

[MasterArn \(p. 563\)](#)

For Lambda@Edge functions, the ARN of the master function.

Type: String

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 \(p. 563\)](#)

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 563\)](#)

The latest updated revision of the function or alias.

Type: String

[Role \(p. 563\)](#)

The function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/[a-zA-Z_0-9+=,.@\\-/]+

[Runtime \(p. 563\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

[State \(p. 563\)](#)

The current state of the function. When the state is Inactive, you can reactivate the function by invoking it.

Type: String

Valid Values: Pending | Active | Inactive | Failed

[StateReason \(p. 563\)](#)

The reason for the function's current state.

Type: String

[StateReasonCode \(p. 563\)](#)

The reason code for the function's current state. When the code is Creating, you can't invoke or modify the function.

Type: String

Valid Values: Idle | Creating | Restoring | EniLimitExceeded | InsufficientRolePermissions | InvalidConfiguration | InternalError | SubnetOutOfRangeAddresses | InvalidSubnet | InvalidSecurityGroup

[Timeout \(p. 563\)](#)

The amount of time that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 563\)](#)

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 608\)](#) object

[Version \(p. 563\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 563\)](#)

The function's networking configuration.

Type: [VpcConfigResponse \(p. 610\)](#) object

Errors

InvalidParameterValueException

One of the parameters in the request is invalid.

HTTP Status Code: 400

PreconditionFailedException

The RevisionId provided does not match the latest RevisionId for the Lambda function or alias. Call the `GetFunction` or the `GetAlias` API to retrieve the latest RevisionId for your resource.

HTTP Status Code: 412

ResourceConflictException

The resource already exists, or another operation is in progress.

HTTP Status Code: 409

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

UpdateFunctionEventInvokeConfig

Updates the configuration for asynchronous invocation for a function, version, or alias.

To configure options for asynchronous invocation, use [PutFunctionEventInvokeConfig \(p. 528\)](#).

Request Syntax

```
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
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 569\)](#)

The name of the Lambda function, version, or alias.

Name formats

- Function name - `my-function` (name-only), `my-function:v1` (with alias).
- Function ARN - `arn:aws:lambda:us-west-2:123456789012:function:my-function`.
- Partial ARN - `123456789012:function:my-function`.

You can append a version number or alias to any of the formats. The length constraint applies only to the full ARN. If you specify only the function name, it is limited to 64 characters in length.

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?`

[Qualifier \(p. 569\)](#)

A version number or alias name.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(|[a-zA-Z0-9$-_]+)`

Request Body

The request accepts the following data in JSON format.

[DestinationConfig \(p. 569\)](#)

A destination for events after they have been sent to a function for processing.

Destinations

- Function - The Amazon Resource Name (ARN) of a Lambda function.
- Queue - The ARN of an SQS queue.
- Topic - The ARN of an SNS topic.
- Event Bus - The ARN of an Amazon EventBridge event bus.

Type: [DestinationConfig \(p. 581\)](#) object

Required: No

[MaximumEventAgeInSeconds \(p. 569\)](#)

The maximum age of a request that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 21600.

Required: No

[MaximumRetryAttempts \(p. 569\)](#)

The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 2.

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "DestinationConfig": {
    "OnFailure": {
      "Destination": "string"
    },
    "OnSuccess": {
      "Destination": "string"
    }
  },
  "FunctionArn": "string",
  "LastModified": number,
  "MaximumEventAgeInSeconds": number,
  "MaximumRetryAttempts": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

[DestinationConfig \(p. 570\)](#)

A destination for events after they have been sent to a function for processing.

Destinations

- Function - The Amazon Resource Name (ARN) of a Lambda function.
- Queue - The ARN of an SQS queue.
- Topic - The ARN of an SNS topic.
- Event Bus - The ARN of an Amazon EventBridge event bus.

Type: [DestinationConfig \(p. 581\)](#) object

[FunctionArn \(p. 570\)](#)

The Amazon Resource Name (ARN) of the function.

Type: String

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 \(p. 570\)](#)

The date and time that the configuration was last updated, in Unix time seconds.

Type: Timestamp

[MaximumEventAgeInSeconds \(p. 570\)](#)

The maximum age of a request that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 21600.

[MaximumRetryAttempts \(p. 570\)](#)

The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 2.

Errors

InvalidArgumentException

One of the parameters in the request is invalid.

HTTP Status Code: 400

ResourceNotFoundException

The resource specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

The request throughput limit was exceeded.

HTTP Status Code: 429

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V3](#)

Data Types

The following data types are supported:

- [AccountLimit \(p. 574\)](#)
- [AccountUsage \(p. 575\)](#)
- [AliasConfiguration \(p. 576\)](#)
- [AliasRoutingConfiguration \(p. 578\)](#)
- [Concurrency \(p. 579\)](#)
- [DeadLetterConfig \(p. 580\)](#)
- [DestinationConfig \(p. 581\)](#)
- [Environment \(p. 582\)](#)
- [EnvironmentError \(p. 583\)](#)
- [EnvironmentResponse \(p. 584\)](#)
- [EventSourceMappingConfiguration \(p. 585\)](#)
- [FunctionCode \(p. 588\)](#)
- [FunctionCodeLocation \(p. 589\)](#)
- [FunctionConfiguration \(p. 590\)](#)
- [FunctionEventInvokeConfig \(p. 595\)](#)
- [Layer \(p. 597\)](#)
- [LayersListItem \(p. 598\)](#)
- [LayerVersionContentInput \(p. 599\)](#)
- [LayerVersionContentOutput \(p. 600\)](#)
- [LayerVersionsListItem \(p. 601\)](#)
- [OnFailure \(p. 603\)](#)
- [OnSuccess \(p. 604\)](#)
- [ProvisionedConcurrencyConfigListItem \(p. 605\)](#)

- [TracingConfig \(p. 607\)](#)
- [TracingConfigResponse \(p. 608\)](#)
- [VpcConfig \(p. 609\)](#)
- [VpcConfigResponse \(p. 610\)](#)

AccountLimit

Limits that are related to concurrency and storage. All file and storage sizes are in bytes.

Contents

CodeSizeUnzipped

The maximum size of a function's deployment package and layers when they're extracted.

Type: Long

Required: No

CodeSizeZipped

The maximum size of a deployment package when it's uploaded directly to AWS Lambda. Use Amazon S3 for larger files.

Type: Long

Required: No

ConcurrentExecutions

The maximum number of simultaneous function executions.

Type: Integer

Required: No

TotalCodeSize

The amount of storage space that you can use for all deployment packages and layer archives.

Type: Long

Required: No

UnreservedConcurrentExecutions

The maximum number of simultaneous function executions, minus the capacity that's reserved for individual functions with [PutFunctionConcurrency \(p. 525\)](#).

Type: Integer

Valid Range: Minimum value of 0.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

AccountUsage

The number of functions and amount of storage in use.

Contents

FunctionCount

The number of Lambda functions.

Type: Long

Required: No

TotalCodeSize

The amount of storage space, in bytes, that's being used by deployment packages and layer archives.

Type: Long

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

AliasConfiguration

Provides configuration information about a Lambda function [alias](#).

Contents

AliasArn

The Amazon Resource Name (ARN) of the alias.

Type: String

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-_]+))?

Required: No

Description

A description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

FunctionVersion

The function version that the alias invokes.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$\\$LATEST|[0-9]+)

Required: No

Name

The name of the alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\\$)([a-zA-Z0-9-_]+)

Required: No

RevisionId

A unique identifier that changes when you update the alias.

Type: String

Required: No

RoutingConfig

The [routing configuration](#) of the alias.

Type: [AliasRoutingConfiguration \(p. 578\)](#) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

AliasRoutingConfiguration

The [traffic-shifting](#) configuration of a Lambda function alias.

Contents

AdditionalVersionWeights

The name of the second alias, and the percentage of traffic that's routed to it.

Type: String to double map

Key Length Constraints: Minimum length of 1. Maximum length of 1024.

Key Pattern: [0–9]⁺

Valid Range: Minimum value of 0.0. Maximum value of 1.0.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

Concurrency

Contents

ReservedConcurrentExecutions

The number of concurrent executions that are reserved for this function. For more information, see [Managing Concurrency](#).

Type: Integer

Valid Range: Minimum value of 0.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

DeadLetterConfig

The [dead-letter queue](#) for failed asynchronous invocations.

Contents

TargetArn

The Amazon Resource Name (ARN) of an Amazon SQS queue or Amazon SNS topic.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:[.*])|()`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

DestinationConfig

A configuration object that specifies the destination of an event after Lambda processes it.

Contents

OnFailure

The destination configuration for failed invocations.

Type: [OnFailure \(p. 603\)](#) object

Required: No

OnSuccess

The destination configuration for successful invocations.

Type: [OnSuccess \(p. 604\)](#) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

Environment

A function's environment variable settings.

Contents

Variables

Environment variable key-value pairs.

Type: String to string map

Key Pattern: [a-zA-Z]([a-zA-Z0-9_])+

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

EnvironmentError

Error messages for environment variables that couldn't be applied.

Contents

ErrorCode

The error code.

Type: String

Required: No

Message

The error message.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

EnvironmentResponse

The results of an operation to update or read environment variables. If the operation is successful, the response contains the environment variables. If it failed, the response contains details about the error.

Contents

Error

Error messages for environment variables that couldn't be applied.

Type: [EnvironmentError \(p. 583\)](#) object

Required: No

Variables

Environment variable key-value pairs.

Type: String to string map

Key Pattern: [a-zA-Z]([a-zA-Z0-9_])+

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

EventSourceMappingConfiguration

A mapping between an AWS resource and an AWS Lambda function. See [CreateEventSourceMapping \(p. 415\)](#) for details.

Contents

BatchSize

The maximum number of items to retrieve in a single batch.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

Required: No

BisectBatchOnFunctionError

(Streams) If the function returns an error, split the batch in two and retry.

Type: Boolean

Required: No

DestinationConfig

(Streams) An Amazon SQS queue or Amazon SNS topic destination for discarded records.

Type: [DestinationConfig \(p. 581\)](#) object

Required: No

EventSourceArn

The Amazon Resource Name (ARN) of the event source.

Type: String

Pattern: `arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-])+:([a-z]{2}(-gov)?-[a-z]+\d{1})?:(\d{12})?:(.*)?`

Required: No

FunctionArn

The ARN of the Lambda function.

Type: String

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-_]+))?`

Required: No

LastModified

The date that the event source mapping was last updated, or its state changed, in Unix time seconds.

Type: Timestamp

Required: No

LastProcessingResult

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

Required: No

MaximumBatchingWindowInSeconds

(Streams) The maximum amount of time to gather records before invoking the function, in seconds.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 300.

Required: No

MaximumRecordAgeInSeconds

(Streams) The maximum age of a record that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 604800.

Required: No

MaximumRetryAttempts

(Streams) The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 10000.

Required: No

ParallelizationFactor

(Streams) The number of batches to process from each shard concurrently.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10.

Required: No

State

The state of the event source mapping. It can be one of the following: Creating, Enabling, Enabled, Disabling, Disabled, Updating, or Deleting.

Type: String

Required: No

StateTransitionReason

Indicates whether the last change to the event source mapping was made by a user, or by the Lambda service.

Type: String

Required: No

UUID

The identifier of the event source mapping.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

FunctionCode

The code for the Lambda function. You can specify either an object in Amazon S3, or upload a deployment package directly.

Contents

S3Bucket

An Amazon S3 bucket in the same AWS Region as your function. The bucket can be in a different AWS account.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 63.

Pattern: ^[0-9A-Za-z\.\-_]*(\?<!\.)\$

Required: No

S3Key

The Amazon S3 key of the deployment package.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

S3ObjectVersion

For versioned objects, the version of the deployment package object to use.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

ZipFile

The base64-encoded contents of the deployment package. AWS SDK and AWS CLI clients handle the encoding for you.

Type: Base64-encoded binary data object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

FunctionCodeLocation

Details about a function's deployment package.

Contents

Location

A presigned URL that you can use to download the deployment package.

Type: String

Required: No

RepositoryType

The service that's hosting the file.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

FunctionConfiguration

Details about a function's configuration.

Contents

CodeSha256

The SHA256 hash of the function's deployment package.

Type: String

Required: No

CodeSize

The size of the function's deployment package, in bytes.

Type: Long

Required: No

DeadLetterConfig

The function's dead letter queue.

Type: [DeadLetterConfig \(p. 580\)](#) object

Required: No

Description

The function's description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

Environment

The function's environment variables.

Type: [EnvironmentResponse \(p. 584\)](#) object

Required: No

FunctionArn

The function's Amazon Resource Name (ARN).

Type: String

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-_]+))?

Required: No

FunctionName

The name of the function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 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-_]+))?`

Required: No

Handler

The function that Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

Required: No

KMSKeyArn

The KMS key that's used to encrypt the function's environment variables. This key is only returned if you've configured a customer managed CMK.

Type: String

Pattern: `(arn:(aws[a-zA-Z-]*)?:[a-zA-Z0-9-.]+:\.*|()`

Required: No

LastModified

The date and time that the function was last updated, in [ISO-8601 format](#) (YYYY-MM-DDThh:mm:ss.sTZD).

Type: String

Required: No

LastUpdateStatus

The status of the last update that was performed on the function. This is first set to `Successful` after function creation completes.

Type: String

Valid Values: `Successful` | `Failed` | `InProgress`

Required: No

LastUpdateStatusReason

The reason for the last update that was performed on the function.

Type: String

Required: No

LastUpdateStatusReasonCode

The reason code for the last update that was performed on the function.

Type: String

Valid Values: `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalServerError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup`

Required: No

Layers

The function's [layers](#).

Type: Array of [Layer \(p. 597\)](#) objects

Required: No

MasterArn

For Lambda@Edge functions, the ARN of the master function.

Type: String

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-_]+))?

Required: No

MemorySize

The memory that's allocated to the function.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

Required: No

RevisionId

The latest updated revision of the function or alias.

Type: String

Required: No

Role

The function's execution role.

Type: String

Pattern: arn:(aws[a-zA-Z-]*)?:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/.]+

Required: No

Runtime

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

Required: No

State

The current state of the function. When the state is `Inactive`, you can reactivate the function by invoking it.

Type: String

Valid Values: Pending | Active | Inactive | Failed

Required: No

StateReason

The reason for the function's current state.

Type: String

Required: No

StateReasonCode

The reason code for the function's current state. When the code is `Creating`, you can't invoke or modify the function.

Type: String

Valid Values: `Idle` | `Creating` | `Restoring` | `EniLimitExceeded` | `InsufficientRolePermissions` | `InvalidConfiguration` | `InternalError` | `SubnetOutOfIPAddresses` | `InvalidSubnet` | `InvalidSecurityGroup`

Required: No

Timeout

The amount of time that Lambda allows a function to run before stopping it.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

TracingConfig

The function's AWS X-Ray tracing configuration.

Type: [TracingConfigResponse \(p. 608\)](#) object

Required: No

Version

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

Required: No

VpcConfig

The function's networking configuration.

Type: [VpcConfigResponse \(p. 610\)](#) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

FunctionEventInvokeConfig

Contents

DestinationConfig

A destination for events after they have been sent to a function for processing.

Destinations

- Function - The Amazon Resource Name (ARN) of a Lambda function.
- Queue - The ARN of an SQS queue.
- Topic - The ARN of an SNS topic.
- Event Bus - The ARN of an Amazon EventBridge event bus.

Type: [DestinationConfig \(p. 581\)](#) object

Required: No

FunctionArn

The Amazon Resource Name (ARN) of the function.

Type: String

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-_]+))?`

Required: No

LastModified

The date and time that the configuration was last updated, in Unix time seconds.

Type: Timestamp

Required: No

MaximumEventAgeInSeconds

The maximum age of a request that Lambda sends to a function for processing.

Type: Integer

Valid Range: Minimum value of 60. Maximum value of 21600.

Required: No

MaximumRetryAttempts

The maximum number of times to retry when the function returns an error.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 2.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V3

Layer

An [AWS Lambda layer](#).

Contents

Arn

The Amazon Resource Name (ARN) of the function layer.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: `arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+`

Required: No

CodeSize

The size of the layer archive in bytes.

Type: Long

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

LayersListItem

Details about an [AWS Lambda layer](#).

Contents

LatestMatchingVersion

The newest version of the layer.

Type: [LayerVersionsListItem \(p. 601\)](#) object

Required: No

LayerArn

The Amazon Resource Name (ARN) of the function layer.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+

Required: No

LayerName

The name of the layer.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+:\d{12}:layer:[a-zA-Z0-9-_]+)|[a-zA-Z0-9-_]+

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

LayerVersionContentInput

A ZIP archive that contains the contents of an [AWS Lambda layer](#). You can specify either an Amazon S3 location, or upload a layer archive directly.

Contents

S3Bucket

The Amazon S3 bucket of the layer archive.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 63.

Pattern: ^[0-9A-Za-z\.\-_]*(\?<!\.\.)\$

Required: No

S3Key

The Amazon S3 key of the layer archive.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

S3ObjectVersion

For versioned objects, the version of the layer archive object to use.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

ZipFile

The base64-encoded contents of the layer archive. AWS SDK and AWS CLI clients handle the encoding for you.

Type: Base64-encoded binary data object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

LayerVersionContentOutput

Details about a version of an [AWS Lambda layer](#).

Contents

CodeSha256

The SHA-256 hash of the layer archive.

Type: String

Required: No

CodeSize

The size of the layer archive in bytes.

Type: Long

Required: No

Location

A link to the layer archive in Amazon S3 that is valid for 10 minutes.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

LayerVersionsListItem

Details about a version of an [AWS Lambda layer](#).

Contents

CompatibleRuntimes

The layer's compatible runtimes.

Type: Array of strings

Array Members: Maximum number of 5 items.

Valid Values: nodejs10.x | nodejs12.x | java8 | java11 | python2.7 | python3.6 | python3.7 | python3.8 | dotnetcore2.1 | dotnetcore3.1 | go1.x | ruby2.5 | ruby2.7 | provided

Required: No

CreatedDate

The date that the version was created, in ISO 8601 format. For example, 2018-11-27T15:10:45.123+0000.

Type: String

Required: No

Description

The description of the version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

LayerVersionArn

The ARN of the layer version.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: arn:[a-zA-Z0-9-]+:lambda:[a-zA-Z0-9-]+\d{12}:layer:[a-zA-Z0-9-_]+:[0-9]+

Required: No

LicensesInfo

The layer's open-source license.

Type: String

Length Constraints: Maximum length of 512.

Required: No

Version

The version number.

Type: Long

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

OnFailure

A destination for events that failed processing.

Contents

Destination

The Amazon Resource Name (ARN) of the destination resource.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 350.

Pattern: ^\$|arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-.])+:([a-z]{2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.*)

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

OnSuccess

A destination for events that were processed successfully.

Contents

Destination

The Amazon Resource Name (ARN) of the destination resource.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 350.

Pattern: ^\$|arn:(aws[a-zA-Z0-9-]*):([a-zA-Z0-9\-.])+:([a-z]{2}(-gov)?-[a-z]+\-\d{1})?:(\d{12})?:(.*)

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

ProvisionedConcurrencyConfigListItem

Details about the provisioned concurrency configuration for a function alias or version.

Contents

AllocatedProvisionedConcurrentExecutions

The amount of provisioned concurrency allocated.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

AvailableProvisionedConcurrentExecutions

The amount of provisioned concurrency available.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

FunctionArn

The Amazon Resource Name (ARN) of the alias or version.

Type: String

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-_+]))?`

Required: No

LastModified

The date and time that a user last updated the configuration, in [ISO 8601 format](#).

Type: String

Required: No

RequestedProvisionedConcurrentExecutions

The amount of provisioned concurrency requested.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

Status

The status of the allocation process.

Type: String

Valid Values: `IN_PROGRESS` | `READY` | `FAILED`

Required: No

StatusReason

For failed allocations, the reason that provisioned concurrency could not be allocated.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

TracingConfig

The function's AWS X-Ray tracing configuration. To sample and record incoming requests, set `Mode` to `Active`.

Contents

Mode

The tracing mode.

Type: String

Valid Values: `Active` | `PassThrough`

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

TracingConfigResponse

The function's AWS X-Ray tracing configuration.

Contents

Mode

The tracing mode.

Type: String

Valid Values: Active | PassThrough

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

VpcConfig

The VPC security groups and subnets that are attached to a Lambda function. For more information, see [VPC Settings](#).

Contents

SecurityGroupIds

A list of VPC security groups IDs.

Type: Array of strings

Array Members: Maximum number of 5 items.

Required: No

SubnetIds

A list of VPC subnet IDs.

Type: Array of strings

Array Members: Maximum number of 16 items.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

VpcConfigResponse

The VPC security groups and subnets that are attached to a Lambda function.

Contents

SecurityGroupIds

A list of VPC security groups IDs.

Type: Array of strings

Array Members: Maximum number of 5 items.

Required: No

SubnetIds

A list of VPC subnet IDs.

Type: Array of strings

Array Members: Maximum number of 16 items.

Required: No

VpcId

The ID of the VPC.

Type: String

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for Ruby V3](#)

在使用 SDK 时出现证书错误

由于 AWS 开发工具包使用的是来自计算机的 CA 证书，因此更改 AWS 服务器上的证书可能会导致您在尝试使用开发工具包时无法连接。您可以通过使计算机的 CA 证书和操作系统保持最新来防止出现这些故障。如果您在公司环境中遇到这个问题而且未管理您自己的计算机，则可能需要请求管理员来协助处理更新过程。以下列表显示了最低的操作系统和 Java 版本：

- 已安装 2005 年 1 月版或更高版本更新的 Microsoft Windows 版本在其信任列表中至少包含一个必需 CA。
- 带 Java for Mac OS X 10.4 版本 5 的 Mac OS X 10.4 (2007 年 2 月版)、Mac OS X 10.5 (2007 年 10 月版) 及更高版本在其信任列表中至少包含一个必需 CA。
- Red Hat Enterprise Linux 5 (2007 年 3 月版)、6 和 7 以及 CentOS 5、6 和 7 在其默认信任 CA 列表中至少包含一个必需 CA。

- Java 1.4.2_12 (2006 年 5 月版)、5 Update 2 (2005 年 3 月版) 以及所有更高版本，包括 Java 6 (2006 年 12 月版)、7 和 8 在其默认信任 CA 列表中至少包含一个必需 CA。

在访问 AWS Lambda 管理控制台或 AWS Lambda API 终端节点时，无论是通过浏览器还是以编程方式，您都需要确保您的客户端计算机支持任何以下 CA：

- Amazon Root CA 1
- Starfield Services Root Certificate Authority – G2
- Starfield Class 2 Certification Authority

可以从 [Amazon Trust Services](#) 获得来自前两个颁发机构的根证书，而使您的计算机保持最新是更直接的解决方案。要了解 ACM 提供的证书的更多信息，请参阅 [AWS Certificate Manager 常见问题](#)。

AWS 词汇表

有关最新 AWS 术语，请参阅 AWS General Reference 中的 [AWS 词汇表](#)。