
AWS Lambda

开发人员指南



AWS Lambda: 开发人员指南

Copyright © 2018 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 Lambda ?	1
您是 AWS Lambda 的新用户吗 ?	1
入门	3
基于 Lambda 的应用程序的构建基块	3
创建并测试基于 Lambda 的应用程序的工具	3
开始前的准备工作	3
下一步	3
设置 AWS 账户	4
设置 AWS 账户并创建管理员用户	4
设置 AWS CLI	6
下一步	6
安装 SAM Local	6
安装 Docker	6
安装 SAM Local	7
创建简单的 Lambda 函数和探索控制台	7
入门准备工作	7
创建简单的 Lambda 函数	8
Lambda 函数	15
构建 Lambda 函数	15
为您的 Lambda 函数编写代码	15
部署代码和创建 Lambda 函数	16
监控和故障排除	17
基于 AWS Lambda 的应用程序示例	17
相关主题	17
编程模型	18
创建部署程序包	77
使用 SAM Local 在本地测试您的无服务应用程序 (公开测试版)	96
使用 AWS Lambda 控制台编辑器创建函数	103
配置 Lambda 函数	127
从 Lambda 函数访问资源	128
访问 AWS 服务	128
访问非 AWS 服务	128
访问私有服务或资源	128
VPC 支持	129
AWS Lambda 执行模型	139
调用 Lambda 函数	140
示例 1	140
示例 2	141
调用类型	142
事件源映射	142
了解重试行为	146
了解扩展行为	147
并发执行请求速率	147
扩展	147
支持的事件源	148
Amazon S3	149
Amazon DynamoDB	149
Amazon Kinesis Data Streams	150
Amazon Simple Notification Service	150
Amazon Simple Email Service	150
Amazon Cognito	151
AWS CloudFormation	151
Amazon CloudWatch Logs	151

Amazon CloudWatch 事件	151
AWS CodeCommit	152
计划的事件 (由 Amazon CloudWatch 事件提供支持)	152
AWS Config	152
Amazon Alexa	152
Amazon Lex	153
Amazon API Gateway	153
AWS IoT 按钮	153
Amazon CloudFront	153
Amazon Kinesis Data Firehose	154
其他事件源：根据需要调用 Lambda 函数	154
示例事件数据	154
使用案例	165
Amazon S3	165
教程	166
Kinesis	181
教程	182
Amazon DynamoDB	191
教程	192
AWS CloudTrail	203
教程	204
Amazon SNS	217
教程	217
Amazon API Gateway	222
将 AWS Lambda 与 Amazon API Gateway 结合使用 (按需并通过 HTTPS)	223
步骤 3：使用 Lambda 和 API 网关 创建简单的微服务	235
移动后端 (Android)	237
教程	239
计划的事件	249
教程	249
自定义用户应用程序	254
教程	255
AWS Lambda@Edge	261
如何为 Lambda@Edge 创建和使用 Lambda 函数	262
如何删除 Lambda 函数的副本	262
设置 Lambda@Edge 的 IAM 权限和角色	262
创建 Lambda@Edge 函数	262
为 Lambda@Edge 函数添加触发器 (AWS Lambda 控制台)	262
为 Lambda@Edge 编写函数	263
为 Lambda@Edge 编辑 Lambda 函数	263
测试和调试	263
Lambda@Edge 限制	263
部署基于 Lambda 的应用程序	264
版本控制和别名	264
版本控制	265
Aliases	269
版本控制、别名和资源策略	276
管理版本控制	278
使用别名的流量转移	280
使用 AWS 无服务器应用程序模型 (AWS SAM)	281
使用 AWS SAM 的无服务器资源	282
下一步	286
创建您自己的无服务器应用程序	286
基于 Lambda 应用程序的自动化部署	288
下一步	288
为您的无服务器应用程序构建管道	288
逐步代码部署	293

对基于 Lambda 的应用程序进行监控和问题排查	296
使用 Amazon CloudWatch	296
诊断场景	296
访问 CloudWatch 指标	297
访问 CloudWatch 日志	299
指标	300
使用 AWS X-Ray	302
利用 AWS X-Ray 跟踪基于 Lambda 的应用程序	303
利用 Lambda 设置 AWS X-Ray	304
从 Lambda 函数发送跟踪分段	305
Lambda 环境中的 AWS X-Ray 守护程序	311
使用环境变量与 AWS X-Ray 通信	312
在 AWS X-Ray 控制台中跟踪 Lambda：示例	312
管理基于 Lambda 的应用程序	314
标记 Lambda 函数	314
为账单标记 Lambda 函数	314
对 Lambda 函数应用标签	314
筛选标记的 Lambda 函数	316
标签限制	317
使用 AWS CloudTrail 进行 API 日志记录	317
CloudTrail 中的 AWS Lambda 信息	317
了解 AWS Lambda 日志文件条目	318
使用 CloudTrail 跟踪函数调用	319
身份验证和访问控制	319
身份验证	320
访问控制	320
访问管理概述	321
使用基于身份的策略 (IAM 策略)	324
使用基于资源的策略 (Lambda 函数策略)	336
权限模型	339
Lambda API 权限参考	341
策略模板	343
管理并发	350
账户级别并发执行数限制	350
函数级别并发执行数限制	351
限制行为	352
监控您的并发使用情况	353
高级主题	354
环境变量	354
设置	354
环境变量的命名规则	356
环境变量和函数版本控制	357
环境变量加密	357
使用环境变量创建 Lambda 函数	358
使用环境变量创建 Lambda 函数以存储敏感信息	360
死信队列	361
最佳实践	362
函数代码	362
函数配置	363
警报与指标	363
流事件调用	363
Async 调用	364
Lambda VPC	364
运行时支持策略	365
执行环境	366
适用于 Lambda 函数的环境变量	366
限制	369

AWS Lambda 限制	369
AWS Lambda 限制错误	370
API 参考	371
在使用 SDK 时出现证书错误	371
Actions	371
AddPermission	373
CreateAlias	378
CreateEventSourceMapping	382
CreateFunction	387
DeleteAlias	395
DeleteEventSourceMapping	397
DeleteFunction	400
DeleteFunctionConcurrency	403
GetAccountSettings	405
GetAlias	407
GetEventSourceMapping	410
GetFunction	413
GetFunctionConfiguration	417
GetPolicy	422
Invoke	425
InvokeAsync	430
ListAliases	433
ListEventSourceMappings	436
ListFunctions	439
ListTags	442
ListVersionsByFunction	444
PublishVersion	447
PutFunctionConcurrency	453
RemovePermission	455
TagResource	458
UntagResource	460
UpdateAlias	462
UpdateEventSourceMapping	466
UpdateFunctionCode	470
UpdateFunctionConfiguration	477
Data Types	484
AccountLimit	485
AccountUsage	487
AliasConfiguration	488
AliasRoutingConfiguration	490
Concurrency	491
DeadLetterConfig	492
Environment	493
EnvironmentError	494
EnvironmentResponse	495
EventSourceMappingConfiguration	496
FunctionCode	498
FunctionCodeLocation	499
FunctionConfiguration	500
TracingConfig	504
TracingConfigResponse	505
VpcConfig	506
VpcConfigResponse	507
文档历史记录	508
AWS 词汇表	513

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，而且无需执行任何管理。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（目前为 Node.js、Java、C#、Go 和 Python）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 AWS CodePipeline 和 AWS CodeBuild 自动部署这些应用程序。有关更多信息，请参阅 [部署基于 Lambda 的应用程序 \(p. 264\)](#)。

有关 AWS Lambda 执行环境的更多信息，请参阅 [Lambda 执行环境和可用库 \(p. 366\)](#)。有关 AWS Lambda 如何确定执行您的代码所需的计算资源的信息，请参阅 [配置 Lambda 函数 \(p. 127\)](#)。

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（即，Node.js、Java、Go、C# 和 Python）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，或自定义操作系统或语言运行时。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。

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. 查看指南中的“Lambda 函数”部分。要了解 Lambda 函数的编程模型和部署选项，您应该熟悉几个核心概念。本节介绍了这些概念，并提供了有关这些概念在可用于编写 Lambda 函数代码的各种语言中的工作方式的详细信息。有关更多信息，请参阅 [Lambda 函数 \(p. 15\)](#)。
3. 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。您还可以了解控制台提供的蓝图以快速创建您的 Lambda 函数。有关更多信息，请参阅 [入门 \(p. 3\)](#)。

4. 阅读本指南的“使用 AWS Lambda 部署应用程序”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。有关更多信息，请参阅 [部署基于 Lambda 的应用程序 \(p. 264\)](#)。

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 [使用案例 \(p. 165\)](#)。

以下主题提供了有关 AWS Lambda 的更多信息：

- [AWS Lambda 函数版本控制和别名 \(p. 264\)](#)
- [使用 Amazon CloudWatch \(p. 296\)](#)
- [使用 AWS Lambda 函数的最佳实践 \(p. 362\)](#)
- [AWS Lambda 限制 \(p. 369\)](#)

入门

在此部分中，我们将向您介绍基于 Lambda 的典型应用程序的基本概念以及可用于创建和测试应用程序的选项。此外，还将向您提供以下相关说明：安装必要的工具来完成本指南中附带的教程以及创建您的第一个 Lambda 函数。

基于 Lambda 的应用程序的构建基块

- Lambda 函数：这是基础，它包含您的自定义代码和任何依赖库。有关更多信息，请参阅 [Lambda 函数 \(p. 15\)](#)。
- 事件源：一个触发您的函数并执行其逻辑的 AWS 服务，例如 Amazon SNS 或自定义服务。有关更多信息，请参阅 [事件源映射 \(p. 142\)](#)。
- 下游资源：您的 Lambda 函数在被触发时所调用的 AWS 服务，例如 DynamoDB 表或 Amazon S3 存储桶。
- 日志流：虽然 Lambda 会自动监控您的函数调用并向 CloudWatch 报告指标，您也可以使用自定义日志记录语句注释您的函数代码，通过这些语句来分析您的 Lambda 函数的执行流程和性能，从而确保它正常工作。
- AWS SAM：一个用于定义 [无服务器应用程序](#)的模型。AWS SAM 受 AWS CloudFormation 的原生支持，可以为表达无服务器资源定义简化的语法。有关更多信息，请参阅 [使用 AWS 无服务器应用程序模型 \(AWS SAM\) \(p. 281\)](#)。

创建并测试基于 Lambda 的应用程序的工具

有三个重要工具可供您用来与 AWS Lambda 服务进行交互，如下所述。我们将在后面若干部分中介绍在构建基于 AWS Lambda 的应用程序时使用的工具。

- Lambda 控制台：为您提供一种方式，使您能够以图形化方式设计您的基于 Lambda 的应用程序，创作或更新您的 Lambda 函数代码，以及配置您的函数所需的事件、下游资源和 IAM 权限。它还包括[高级主题 \(p. 354\)](#)中概述的高级配置选项。
- AWS CLI：一个命令行界面，您可以通过命令行界面来利用 Lambda 的 API 操作，例如创建函数和映射事件源。有关 Lambda 的 API 操作的完整列表，请参阅 [Actions \(p. 371\)](#)。
- SAM Local：一个命令行界面，您可以用于在本地开发、测试和分析您的无服务器应用程序，然后再将其上传到 Lambda 运行时。有关更多信息，请参阅 [使用 SAM Local 在本地测试您的无服务应用程序 \(公开测试版\) \(p. 96\)](#)。

开始前的准备工作

要使用本部分结尾处提供的教程，请确保您已完成以下操作：

- [设置 AWS 账户 \(p. 4\)](#)
- [设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)
- 已完成使用 SAM Local (包括 Docker) 的步骤，此处概述了这些步骤：[安装 SAM Local \(p. 6\)](#)。

下一步

[设置 AWS 账户 \(p. 4\)](#)

设置 AWS 账户

如果您尚未执行此操作，则需要注册 AWS 账户并在该账户中创建管理员用户。您还需要设置 AWS Command Line Interface (AWS CLI)。很多教程使用了 AWS CLI。

要完成设置，请遵循以下主题中的说明：

设置 AWS 账户并创建管理员用户

注册 AWS

当您注册 Amazon Web Services (AWS) 时，您的 AWS 账户会自动注册 AWS 中的所有服务，包括 AWS Lambda。您只需为使用的服务付费。

借助 AWS Lambda，您仅需为实际使用的资源付费。有关 AWS Lambda 使用费率的更多信息，请参阅 [AWS Lambda 产品页面](#)。如果您是新的 AWS 客户，则可以开始免费使用 AWS Lambda。有关更多信息，请参阅 [AWS 免费使用套餐](#)。

如果您已有一个 AWS 账户，请跳到下一个任务。如果您还没有 AWS 账户，请使用以下步骤创建。

创建 AWS 账户

1. 打开 <https://aws.amazon.com/>，然后选择 Create an AWS Account。

Note

如果您之前已登录 AWS 管理控制台，则可能无法在浏览器中执行此操作。在此情况下，请选择 Sign in to a different account，然后选择 Create a new AWS account。

2. 按照屏幕上的说明进行操作。

作为注册流程的一部分，您会收到一个电话，需要您使用电话键盘输入一个 PIN 码。

请记住您的 AWS 账户 ID，因为进行下一个任务时需要用到。

创建 IAM 用户

AWS 中的服务（例如 AWS Lambda）要求您在访问时提供凭证，以便让服务确定您是否有权访问服务所拥有的资源。控制台要求您的密码。您可以为您的 AWS 账户创建访问密钥以访问 AWS CLI 或 API。但是，我们不建议使用您的 AWS 账户的凭证访问 AWS。相反，我们建议您使用 AWS Identity and Access Management (IAM)。创建 IAM 用户，将该用户添加到具有管理权限的 IAM 组，然后向您创建的 IAM 用户授予管理权限。您随后可以使用特殊 URL 和该 IAM 用户的凭证访问 AWS。

如果您已注册 AWS 但尚未为自己创建一个 IAM 用户，则可以使用 IAM 控制台自行创建。

本指南中的入门练习和教程假定您拥有具有管理权限的用户（adminuser）。在按照流程进行操作时，请创建名为“adminuser”的用户。

为您自己创建一个 IAM 用户并将该用户添加到管理员组

1. 使用 AWS 账户电子邮件地址和密码，以 [AWS 账户根用户](#) 身份登录到 IAM 控制台 (<https://console.aws.amazon.com/iam/>)。

Note

我们强烈建议您遵守以下使用###用户的最佳实践，妥善保存根用户凭证。只在执行少数[账户和服务管理任务](#)时才作为根用户登录。

2. 在控制台的导航窗格中，选择 Users，然后选择 Add user。

3. 对于 User name , 键入 **Administrator**。
4. 选中 AWS 管理控制台 access 旁边的复选框 , 选择 Custom password , 然后在文本框中键入新用户的密码。您可以选择 Require password reset 以强制用户在下次登录时选择新密码。
5. 选择 Next: Permissions。
6. 在 Set permissions for user 页面上 , 选择 Add user to group。
7. 选择 Create group。
8. 在 Create group 对话框中 , 键入 **Administrators**。
9. 对于 Filter , 选择 Job function。
10. 在策略列表中 , 选中 AdministratorAccess 的复选框。然后选择 Create group。
11. 返回到组列表中 , 选中您的新组所对应的复选框。如有必要 , 选择 Refresh 以便在列表中查看该组。
12. 选择 Next: Review 以查看要添加到新用户的组成员资格的列表。如果您已准备好继续 , 请选择 Create user。

您可使用此相同的流程创建更多的组和用户 , 并允许您的用户访问 AWS 账户资源。要了解有关使用策略限制用户对特定 AWS 资源的权限的信息 , 请参阅[访问管理和示例策略](#)。

以新 IAM 用户身份登录

1. 注销 AWS 管理控制台。
2. 使用下面的 URL 格式登录控制台 :

```
https://aws_account_number.signin.aws.amazon.com/console/
```

aws_account_number 是您的 AWS 账户 ID (无连字符) 。例如 , 如果您的 AWS 账户 ID 是 1234-5678-9012 , 则您的 AWS 账号为 123456789012 。有关如何查找账号的信息 , 请参阅 IAM 用户指南 中的 [AWS 账户 ID 及其别名](#)。

3. 输入您刚创建的 IAM 用户名和密码。登录后 , 导航栏将显示 *your_user_name* @ *your_aws_account_id* 。

如果您不希望您的登录页面 URL 包含 AWS 账户 ID , 可以创建账户别名。

创建或删除账户别名

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/> 。
2. 在导航窗格上 , 选择 Dashboard。
3. 查找 IAM 用户登录链接。
4. 要创建别名 , 请单击 Customize , 输入要用作别名的名称 , 然后选择 Yes, Create。
5. 要删除别名 , 请选择 Customize , 然后选择 Yes, Delete。登录 URL 会恢复使用 AWS 账户 ID。

要在创建账户别名后登录 , 请使用以下 URL :

```
https://your_account_alias.signin.aws.amazon.com/console/
```

要为您的账户验证 IAM 用户的登录链接 , 请打开 IAM 控制台并在控制面板的 IAM users sign-in link: 下进行检查。

有关 IAM 的更多信息 , 请参阅下文 :

- [AWS Identity and Access Management \(IAM\)](#)
- [入门](#)
- [IAM 用户指南](#)

下一步

[设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)

设置 AWS Command Line Interface (AWS CLI)

本指南中的所有练习均假定您使用账户中的管理员用户凭证 (adminuser) 来执行这些操作。有关在您的 AWS 账户中创建管理员用户的说明，请参阅[设置 AWS 账户并创建管理员用户 \(p. 4\)](#)，然后遵循这些步骤来下载和配置 AWS Command Line Interface (AWS CLI)。

设置 AWS CLI

1. 下载并配置 AWS CLI。有关说明，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](#)
2. 在 AWS CLI 配置文件中为管理员用户添加一个命名配置文件。在执行 AWS CLI 命令时，您将使用此配置文件。

```
[profile adminuser]
aws_access_key_id = adminuser access key ID
aws_secret_access_key = adminuser secret access key
region = aws-region
```

有关可用 AWS 区域的列表，请参阅 Amazon Web Services 一般参考 中的[区域和终端节点](#)。

3. 在命令提示符处输入以下命令来验证设置。
 - 尝试 help 命令来验证您的计算机上是否安装了 AWS CLI：

```
aws help
```

- 尝试一个 Lambda 命令来验证用户是否可以访问 AWS Lambda。此命令将列出账户中的 Lambda 函数（如果有）。AWS CLI 使用 adminuser 凭证来验证请求。

```
aws lambda list-functions --profile adminuser
```

下一步

[安装 SAM Local \(p. 6\)](#)

安装 SAM Local

SAM Local 是一种工具，它还可用来实现更快地迭代开发 Lambda 函数代码，在[使用 SAM Local 在本地测试您的无服务应用程序 \(公开测试版\) \(p. 96\)](#)中对此进行了介绍。要使用 SAM Local，您首先需要安装 Docker。

安装 Docker

Docker 是开源软件容器平台，允许您构建、管理和测试在 Linux、Mac 或 Windows 上运行的应用程序。有关更多信息和下载说明，请参阅[Docker](#)。

安装 Docker 后，SAM Local 会自动提供自定义 Docker 镜像，名为 docker-lambda。此镜像由 AWS 合作伙伴专门设计，用于模拟真正的 AWS Lambda 执行环境。此环境中包括已安装的软件、库、安全权限、环境变量以及[Lambda 执行环境和可用库 \(p. 366\)](#)中介绍的其他功能。

您可以使用 docker-lambda 在本地调用您的 Lambda 函数。在此环境中，您的无服务应用程序就像在 AWS Lambda 运行时中那样执行和表现，而您不必重新部署运行时。它们在此环境中的执行和性能反应了超时和内存使用等情况。

Important

由于这是模拟环境，不能保证本地测试结果与实际 AWS 运行时完全相同。

有关更多信息，请参阅 [GitHub 上的 Docker Lambda](#)。(如果您没有 Github 账户，则可以免费创建一个，然后访问 Docker Lambda。)

安装 SAM Local

您可以在 Linux、Mac 和 Windows 环境中运行 SAM Local。SAM Local 最简单的安装方法是使用 [NPM](#)，

```
npm install -g aws-sam-local
```

然后验证安装是否成功。

```
sam --version
```

如果 NPM 不适用，您可以下载最新二进制文件，立即开始使用 SAM Local。您可以在 [SAM CLI GitHub 存储库](#) 中的“Releases”部分找到二进制文件。

下一步

[创建简单的 Lambda 函数和探索控制台 \(p. 7\)](#)

创建简单的 Lambda 函数和探索控制台

在本入门练习中，您首先将使用 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。

在执行这些步骤时，您还将熟悉 AWS Lambda 控制台，包括：

- 探索蓝图。每个蓝图均提供了示例代码和示例配置，让您只需单击几下即可创建 Lambda 函数。
- 查看和更新您的 Lambda 函数的配置信息。
- 手动调用 Lambda 函数并在 Execution results 部分中探索结果。
- 在控制台中监控 CloudWatch 指标。

入门准备工作

首先，您需要注册 AWS 账户并在账户中创建管理员用户。有关说明，请参阅[设置 AWS 账户 \(p. 4\)](#)。

下一步

[创建简单的 Lambda 函数 \(p. 8\)](#)

创建简单的 Lambda 函数

按照此部分中的步骤创建简单的 Lambda 函数。

创建 Lambda 函数

1. 登录 AWS 管理控制台并打开 AWS Lambda 控制台。
2. 请注意，AWS Lambda 在推出时在 How it works 标签下方提供了一个简单的 Hello World 函数，并且包含一个 Run 选项，使您可以通过一般方式调用该函数。此教程介绍了其他一些选项，您在创建、测试和更新您的 Lambda 函数以及 Lambda 控制台提供的其他功能时可以使用这些选项；此外提供了指向每个选项的链接，可让您深入地探索每个选项。

选择 Get Started 部分下方的 Create a function 以继续。

The screenshot shows the AWS Lambda 'Get Started' page. At the top, it says 'COMPUTE' and features a large heading 'AWS Lambda' with the tagline 'lets you run code without thinking about servers.' Below this, a sub-headline reads 'You pay only for the compute time you consume — there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service, all with zero administration.' A code editor window titled 'How it works' contains the following Node.js code:

```
1 exports.handler = (event, context, callback) => {
2     // Succeed with the string "Hello world!"
3     callback(null, 'Hello world!');
4 }
```

Below the code editor, a section titled 'Just write the code' contains the text: 'Above is a simple Node.js Lambda function. Try changing callback values and run the function before proceeding.' At the bottom, there's a 'Related services' section with cards for S3 and API Gateway.

Related services

S3
Amazon Simple Storage Service (Amazon S3) makes it simple and practical to collect, store, and analyze data - regardless of format - all at massive scale.

[See more related services](#)

API Gateway
Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.

Note

仅当您尚未创建任何 Lambda 函数时，控制台才显示 Get Started 页面。如果您已创建函数，则会看到 Lambda > Functions 页面。在该列表页面上，选择创建函数以转到创建函数页面。

3. 在创建函数页面上，会向您提供两个选项：

- 从头开始创作
 - 蓝图
- a. 如果您想查看蓝图，请选择蓝图按钮，这将显示可用的蓝图。您还可以使用 Filter 来搜索特定蓝图。例如：
- 在 Filter 中输入 **s3** 以仅获取处理 Amazon S3 事件的可用蓝图的列表。
 - 在 Filter 中输入 **dynamodb** 以获取处理 Amazon DynamoDB 事件的可用蓝图的列表。
- b. 对于此入门练习，请选择 **Author from scratch** 按钮。
4. 在 Author from scratch 中，执行以下操作：
- 在文件名*中，指定您的 Lambda 函数的名称。
 - 在 Runtime* 中，保留默认值 `Node.js 6.10`。
 - 在 Role* 中，保留默认值 `Choose an existing role`。
 - 在现有角色*中，选择 `lambda_basic_execution`。
 - 选择创建函数。
5. 在您的新 **function-name** 页面上，请注意以下几点：

Congratulations! Your Lambda function "MyFunction" has been successfully created. You can now change its code and configuration. Click on the "Test" button to input a test event when you are ready to test your function.

Configuration Monitoring

Add triggers

Click on a trigger from the list below to add it to your function.

- API Gateway
- AWS IoT
- Alexa Skills Kit
- Alexa Smart Home
- CloudFront
- CloudWatch Events

MyFunction

Amazon CloudWatch Logs

Amazon SNS

Resources the function's role has access to will be shown here

在 Add triggers 面板中，您可以选择性地通过选择列出的服务选项之一，来选择某个自动触发您的 Lambda 函数的服务。

- a. 根据所选择的服务，系统会提示您为该服务提供相关信息。例如，如果选择 DynamoDB，则您需要提供以下信息：
- DynamoDB 表的名称
 - 批处理大小
 - 起始位置
- b. 在此示例中，不要配置触发器。
- 在 Function code 中，请注意提供了使用 Node.js 编写的代码。它会返回简单的“Hello Lambda”问候语。

- Handler 显示 `lambda_function.lambda_handler` 值。它是 [`filename.handler-function`](#)。控制台将示例代码保存在 `lambda_function.py` 文件中，而在该代码中，`lambda_handler` 是调用 Lambda 函数时将事件作为参数接收的函数名称。有关更多信息，请参阅 [Lambda 函数处理程序 \(Python\) \(p. 53\)](#)。
 - 请注意嵌入式 IDE (集成开发环境)。要了解更多信息，请参阅[“使用 AWS Lambda 控制台编辑器创建函数 \(p. 103\)”](#)。
6. 此页面上的其他配置选项包括：
- Environment variables – 用于 Lambda 函数，使您可以动态地将设置传递到函数代码和库，而无需更改代码。有关更多信息，请参阅 [环境变量 \(p. 354\)](#)。
 - Tags – 您附加到 AWS 资源的键值对，以便更好地组织资源。有关更多信息，请参阅 [标记 Lambda 函数 \(p. 314\)](#)。
 - Execution role – 允许您通过使用定义的角色和策略或创建新的角色和策略来管理函数的安全性。有关更多信息，请参阅 [AWS Lambda 的身份验证和访问控制 \(p. 319\)](#)。
 - Basic settings – 允许您指明 Lambda 函数的内存分配和超时限制。有关更多信息，请参阅 [AWS Lambda 限制 \(p. 369\)](#)。
 - Network – 允许您选择函数要访问的 VPC。有关更多信息，请参阅 [配置 Lambda 函数以访问 Amazon VPC 中的资源 \(p. 129\)](#)。
 - Debugging and error handling – 允许您选择 [死信队列 \(p. 361\)](#) 资源来分析失败的函数调用重试。此外，它还允许您启用活动跟踪。有关更多信息，请参阅 [使用 AWS X-Ray \(p. 302\)](#)。
 - Concurrency – 允许您为此函数分配特定的并发执行限制。有关更多信息，请参阅 [函数级别并发执行数限制 \(p. 351\)](#)。
 - Auditing and compliance – 记录函数调用以用于运营和风险审核、治理和合规性目的。有关更多信息，请参阅 [配合使用 AWS Lambda 和 AWS CloudTrail \(p. 203\)](#)。

手动调用 Lambda 函数并验证结果、日志和指标

按照以下步骤，使用控制台中提供的示例事件数据调用 Lambda 函数。

- 在 [yourfunction](#) 页面上，选择 Test。
- 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。选择 Save and test。

- 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。
- 成功执行后，在控制台中查看结果。

The screenshot shows the AWS Lambda execution results page. At the top, it says "Execution result: succeeded (logs)". Below that is a "Details" section with a link to "Logs". A note says "The area below shows the result returned by your function execution. Learn more about returning results from your function." The logs show the output "Hello from Lambda". The "Summary" section provides performance metrics: Code SHA-256 (EK2s5r/qvcnS/tlz7w9OWqkT6XkRzQ6RtC2JFG3K8o=), Duration (9.04 ms), Resources configured (128 MB), Request ID (5e2630d3-d476-11e7-8121-ff2df5001d1f), Billed duration (100 ms), Max memory used (19 MB). The "Log output" section shows CloudWatch log entries: START RequestId: 5e2630d3-d476-11e7-8121-ff2df5001d1f Version: \$LATEST, END RequestId: 5e2630d3-d476-11e7-8121-ff2df5001d1f, REPORT RequestId: 5e2630d3-d476-11e7-8121-ff2df5001d1f Duration: 9.04 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 19 MB.

请注意以下几点：

- 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。

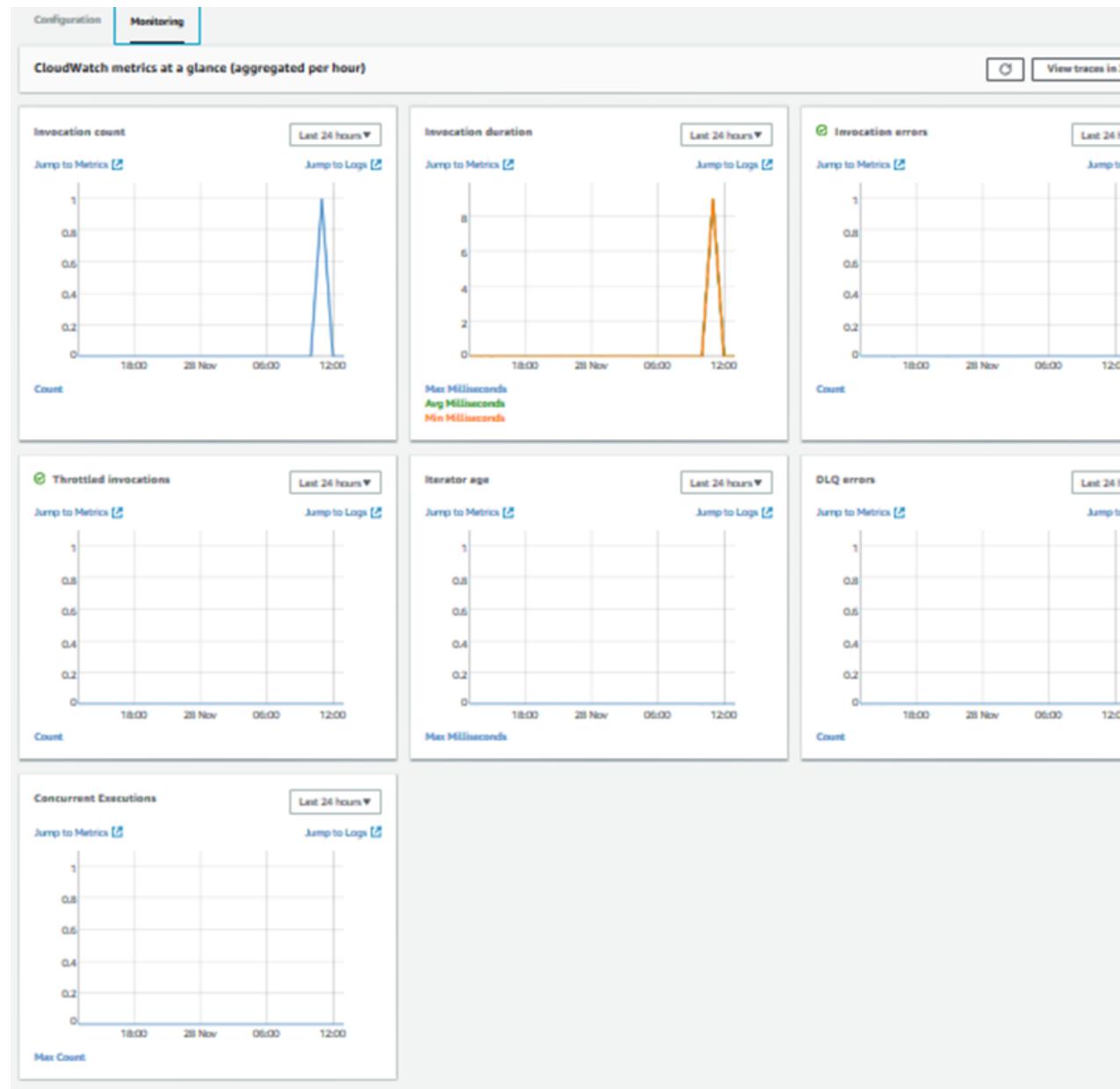
Note

控制台在调用可导致 AWS Lambda 立即返回响应的 Lambda 函数时始终使用 RequestResponse 调用类型（同步调用）。有关更多信息，请参阅 [调用类型 \(p. 142\)](#)。

- 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。

- 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。
- 选择 Monitoring 选项卡以查看 Lambda 函数的 CloudWatch 指标。此页面显示 CloudWatch 指标。



请注意以下几点：

- X 轴显示自当前时间起的过去 24 小时。
- 调用计数显示此时间间隔内的调用数。
- 调用持续时间显示您的 Lambda 函数运行所需的时长。它显示了执行的最短、最长和平均时间。
- 调用错误显示您的 Lambda 函数失败的次数。您可以比较函数执行的次数与函数失败的次数（如果有）。
- 受限的调用指标显示 AWS Lambda 是否限制了您的 Lambda 函数调用。有关更多信息，请参阅 [AWS Lambda 限制 \(p. 369\)](#)。
- 并发执行指标显示了您的 Lambda 函数的并发调用数。有关更多信息，请参阅 [管理并发 \(p. 350\)](#)。
- 为了方便起见，AWS Lambda 控制台为您显示了这些 CloudWatch 指标。您可以通过单击任一指标来在 Amazon CloudWatch 控制台中查看这些指标。

有关这些指标及其含义的更多信息，请参阅[AWS Lambda CloudWatch 指标 \(p. 301\)](#)。

Lambda 函数

如果您是首次使用 AWS Lambda，您可能会问：AWS Lambda 如何执行我的代码？AWS Lambda 如何知道运行我的 Lambda 代码所需的内存量和 CPU 要求？以下部分概述了 Lambda 函数的工作方式。

在后续部分中，我们将介绍如何调用您创建的函数以及如何部署和监控这些函数。我们还建议阅读[使用 AWS Lambda 函数的最佳实践 \(p. 362\)](#)中的函数代码和函数配置部分。

开始时，我们将向您介绍有关构建 Lambda 函数的基础知识方面的主题：[构建 Lambda 函数 \(p. 15\)](#)。

构建 Lambda 函数

您以一个或多个 Lambda 函数的形式将应用程序代码上传到 AWS Lambda（一种计算服务），此服务可代表您运行代码。AWS Lambda 负责预配置和管理服务器以便在调用时运行代码。

通常，基于 AWS Lambda 的应用程序的生命周期包括编写代码、将代码部署到 AWS Lambda，然后是监控和故障排除。以下是生命周期的每个阶段出现的常见问题：

- 为 Lambda 函数编写代码 - 支持哪些语言？是否有我需要遵循的编程模型？如何打包我的代码和依赖项以便上传到 AWS Lambda？有哪些可用工具？
- 上传代码并创建 Lambda 函数 - 如何将我的代码包上传到 AWS Lambda？如何告知 AWS Lambda 从何处开始执行我的代码？如何指定像内存和超时这样的计算要求？
- 监控和故障排除 - 对于我的生产中的 Lambda 函数，有哪些可用的指标？如果出现任何故障，如何获取日志或解决问题？

下列部分提供了介绍性信息，而最后的“示例”部分提供了工作示例以帮助您了解。

为您的 Lambda 函数编写代码

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[Lambda 执行环境和可用库 \(p. 366\)](#)。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• 有关更多信息，请参阅 部署代码和创建 Lambda 函数 (p. 16)。

语言	用于编写代码的工具和选项
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）您自己的编写环境有关更多信息，请参阅部署代码和创建 Lambda 函数 (p. 16)。
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 内核（请参阅.NET 内核安装指南）您自己的编写环境有关更多信息，请参阅部署代码和创建 Lambda 函数 (p. 16)。
Python	<ul style="list-style-type: none">AWS Lambda 控制台您自己的编写环境有关更多信息，请参阅部署代码和创建 Lambda 函数 (p. 16)。
转到	<ul style="list-style-type: none">您自己的编写环境有关更多信息，请参阅部署代码和创建 Lambda 函数 (p. 16)。

此外，无论您选择哪种语言，都有一个用于编写 Lambda 函数代码的模式。例如，如何编写 Lambda 函数的处理程序方法（即，AWS Lambda 在开始执行代码时首先调用的方法）、如何将事件传递给处理程序、可在代码中使用哪些语句来在 CloudWatch Logs 中生成日志、如何与 AWS Lambda 运行时交互并获取信息（例如，超时前的剩余时间）以及如何处理异常。[编程模型 \(p. 18\)](#)部分提供了每种受支持的语言的信息。

Note

在您熟悉 AWS Lambda 后，请参阅[使用案例 \(p. 165\)](#)，其中提供的分步说明可帮助您探究端到端体验。

部署代码和创建 Lambda 函数

要创建 Lambda 函数，您首先将代码和依赖项打包到部署程序包中。然后，将部署程序包上传到 AWS Lambda 以创建 Lambda 函数。

主题

- [创建部署程序包 - 组织代码和依赖项 \(p. 16\)](#)
- [上传部署程序包 - 创建 Lambda 函数 \(p. 16\)](#)
- [测试 Lambda 函数 \(p. 17\)](#)

创建部署程序包 - 组织代码和依赖项

您必须首先通过特定方式来组织您的代码和依赖项并创建部署程序包。有关创建部署程序包的说明因您选择用来编写代码的语言而异。例如，您可以使用构建插件（例如，Jenkins（针对 Node.js 和 Python）和 Maven（针对 Java））来创建部署程序包。有关更多信息，请参阅[创建部署程序包 \(p. 77\)](#)。

在使用控制台创建 Lambda 函数时，控制台会为您创建部署程序包，然后上传该部署程序包以创建 Lambda 函数。

上传部署程序包 - 创建 Lambda 函数

AWS Lambda 提供了[CreateFunction \(p. 387\)](#)操作，您可使用此操作创建 Lambda 函数。可使用 AWS Lambda 控制台、AWS CLI 和 AWS 开发工具包创建 Lambda 函数。在内部，所有这些接口都调用[CreateFunction](#)操作。

除了提供您的部署程序包之外，您还可以在创建 Lambda 函数时提供配置信息，包括 Lambda 函数的计算要求、Lambda 函数中的处理程序方法的名称以及运行时，具体取决于您选择用来编写代码的语言。有关更多信息，请参阅 [Lambda 函数 \(p. 15\)](#)。

测试 Lambda 函数

如果您的 Lambda 函数旨在处理特定类型的事件，您可以使用示例事件数据通过下列方法之一来测试 Lambda 函数：

- 在控制台中测试 Lambda 函数。
- 使用 AWS CLI 测试 Lambda 函数。您可以使用 `Invoke` 方法调用您的 Lambda 函数并传入示例事件数据。
- 使用 [使用 SAM Local 在本地测试您的无服务应用程序 \(公开测试版\) \(p. 96\)](#) 在本地测试您的 Lambda 函数。

控制台提供了示例事件数据。[事件源发布的示例事件 \(p. 154\)](#)主题中还提供了相同的数据，您可以在 AWS CLI 中使用此数据来调用 Lambda 函数。

监控和故障排除

在您的 Lambda 函数处于生产中后，AWS Lambda 将代表您自动监控函数，并通过 Amazon CloudWatch 报告指标。有关更多信息，请参阅 [访问 AWS Lambda 的 Amazon CloudWatch 指标 \(p. 297\)](#)。

为帮助您排除函数中的故障，Lambda 会记录您的函数处理的所有请求，并自动将代码生成的日志存储在 Amazon CloudWatch Logs 中。有关更多信息，请参阅 [访问 AWS Lambda 的 Amazon CloudWatch 日志 \(p. 299\)](#)。

基于 AWS Lambda 的应用程序示例

本指南提供了多个示例以及分步说明。如果您是首次使用 AWS Lambda，建议您尝试以下练习：

- [入门 \(p. 3\)](#) - 入门练习提供了基于控制台的体验。并为您的首选运行时提供了示例代码。您还可以使用[代码编辑器](#)在控制台中编写代码，并将代码上传到 AWS Lambda，然后使用控制台中提供的示例事件数据来测试代码。
- [使用案例 \(p. 165\)](#) - 如果您无法使用控制台编写代码，则必须创建您自己的部署程序包，并使用 AWS CLI（或开发工具包）创建您的 Lambda 函数。有关更多信息，请参阅 [为您的 Lambda 函数编写代码 \(p. 15\)](#)。“使用案例”部分中的大多数示例都使用 AWS CLI。如果您是首次使用 AWS Lambda，建议您尝试下列练习之一。

相关主题

下列主题提供了其他信息。

[编程模型 \(p. 18\)](#)

[创建部署程序包 \(p. 77\)](#)

[AWS Lambda 函数版本控制和别名 \(p. 264\)](#)

[使用 Amazon CloudWatch \(p. 296\)](#)

编程模型

您将使用 AWS Lambda 支持的语言之一为 Lambda 函数编写代码。无论您选择哪种语言，都可以采用一个通用模式，即为包含以下核心概念的 Lambda 函数编写代码：

- 处理程序 - 处理程序是 AWS Lambda 调用的用于开始执行 Lambda 函数的函数。您在创建 Lambda 函数时标识处理程序。在调用 Lambda 函数时，AWS Lambda 将通过调用处理程序函数来开始执行您的代码。AWS Lambda 将任何事件数据作为第一个参数传递给处理程序。您的处理程序应处理此传入事件数据，并且可调用您的代码中的任何其他函数/方法。
- context 对象以及它在运行时与 Lambda 交互的方式 - AWS Lambda 还将 context 对象作为第二个参数传递给处理程序函数。通过此 context 对象，您的代码可与 AWS Lambda 交互。例如，您的代码可在 AWS Lambda 终止 Lambda 函数前发现剩余的执行时间。

此外，对于 Node.js 等语言，有一个使用回调的异步平台。AWS Lambda 提供了有关此 context 对象的其他方法。您将使用这些 context 对象方法指示 AWS Lambda 终止 Lambda 函数并将值返回到调用方（可选）。

- 日志记录 - 您的 Lambda 函数可包含日志记录语句。AWS Lambda 将这些日志写入 CloudWatch Logs。特定语言语句将生成日志条目，具体取决于您编写 Lambda 函数代码所用的语言。
- 异常 - 您的 Lambda 函数需要将函数执行的结果传递给 AWS Lambda。根据您编写 Lambda 函数代码所用的语言，成功结束请求或向 AWS Lambda 通知执行期间出现了错误有几种不同的方式。如果您同步调用该函数，则 AWS Lambda 会将结果转发回客户端。

Note

必须以无状态风格编写 Lambda 函数代码，且与底层的计算基础设施不存在关联。您的代码应该要求将本地文件系统访问权限、子流程和类似项目限制为请求的生命周期。持久状态应存储在 Amazon S3、Amazon DynamoDB 或其他云存储服务中。要求函数无状态可使 AWS Lambda 根据需要启动合适数量的函数副本，以根据传入事件和请求的速率调整规模。对于不同的请求，这些函数可能不会始终运行在同一个计算实例上，且 AWS Lambda 可能会多次使用 Lambda 函数的给定实例。有关更多信息，请参阅 [编程模型 \(Node.js\) \(p. 18\)](#)。

以下语言特定主题提供了详细信息：

- [编程模型 \(Node.js\) \(p. 18\)](#)
- [使用 Java 编写 Lambda 函数的编程模型 \(p. 33\)](#)
- [使用 C# 编写 Lambda 函数的编程模型 \(p. 68\)](#)
- [用于使用 Python 编写 Lambda 函数的编程模型 \(p. 53\)](#)
- [使用 Go 编写 Lambda 函数的编程模型 \(p. 60\)](#)

编程模型 (Node.js)

AWS Lambda 当前支持以下 Node.js 运行时：

- Node.js 运行时 v6.10 (runtime = nodejs6.10)
- Node.js 运行时 v4.3 (runtime = nodejs4.3)
- Node.js 运行时 v0.10.42 (runtime = nodejs)

Important

Node v0.10.42 目前已淘汰。有关更多信息，请参阅 [运行时支持策略 \(p. 365\)](#)。您必须尽快将现有函数迁移到 AWS Lambda 上可用的较新 Node.js 运行时版本 (nodejs4.3 或 nodejs6.10)。注意，对于包含以 Node v0.10.42 运行时编写的函数的每个区域，您都必须遵循此过程。有关 v0.10.42 运行时中编程模型差异的信息，请参阅 [使用较早的 Node.js 运行时 v0.10.42 \(p. 28\)](#)。

当您创建 Lambda 函数时，请指定要使用的运行时。有关更多信息，请参阅 [CreateFunction \(p. 387\)](#) 的 `runtime` 参数。

以下各节说明了在使用 Node.js 编写 Lambda 函数代码时 [常见的编程模式和核心概念](#) 的适用情况。除非另有说明，否则以下部分中描述的编程模型适用于所有版本。

主题

- [Lambda 函数处理程序 \(Node.js\) \(p. 19\)](#)
- [Context 对象 \(Node.js\) \(p. 21\)](#)
- [日志记录 \(Node.js\) \(p. 24\)](#)
- [函数错误 \(Node.js\) \(p. 26\)](#)
- [使用较早的 Node.js 运行时 v0.10.42 \(p. 28\)](#)

Lambda 函数处理程序 (Node.js)

在创建 Lambda 函数时，需要指定一个处理程序（您代码中的一个函数），以供 AWS Lambda 在服务执行代码时调用。在使用 Node.js 创建处理程序函数时，请使用以下一般语法。

```
exports.myHandler = function(event, context) {  
    ...  
}
```

回调参数是可选的，取决于您是否希望将信息返回到调用方。

```
exports.myHandler = function(event, context, callback) {  
    ...  
  
    // Use callback() and return information to the caller.  
}
```

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。
- `context` - AWS Lambda 使用此参数为您的处理程序提供正在执行的 Lambda 函数的运行时信息。有关更多信息，请参阅 [Context 对象 \(Node.js\) \(p. 21\)](#)。
- `callback` - 您可以使用可选的回调将信息返回给调用方，否则返回值为 `null`。有关更多信息，请参阅 [使用回调参数 \(p. 20\)](#)。

Note

只有 Node.js 运行时 v6.10 和 v4.3 支持回调。如果您使用运行时 v0.10.42，则需要使用 `context` 方法 (`done`、`succeed` 和 `fail`) 正确终止 Lambda 函数。有关信息，请参阅 [使用较早的 Node.js 运行时 v0.10.42 \(p. 28\)](#)。

- `myHandler` - 这是 AWS Lambda 调用的函数的名称。您导出此项，使其对 AWS Lambda 可见。假设您将此代码保存为 `helloworld.js`。然后，`helloworld.myHandler` 是处理程序。有关更多信息，请参阅 [CreateFunction \(p. 387\)](#)。

- 如果使用了 RequestResponse 调用类型（同步执行），则 AWS Lambda 会将 Node.js 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，已序列化为 JSON）。例如，AWS Lambda 控制台使用 RequestResponse 调用类型，因此当您使用该控制台测试调用函数时，该控制台将显示返回值。

如果处理程序未返回任何内容，AWS Lambda 将返回空值。

- 如果使用了 Event 调用类型（异步执行），则会放弃该值。

示例

请考虑以下 Node.js 示例代码。

```
exports.myHandler = function(event, context, callback) {
    console.log("value1 = " + event.key1);
    console.log("value2 = " + event.key2);
    callback(null, "some success message");
    // or
    // callback("some error type");
}
```

此示例有一个函数，该函数同时也是处理程序。在该函数中，`console.log()` 语句会将一些传入的事件数据记录到 CloudWatch Logs。在调用回调时，Lambda 函数仅在 Node.js 事件循环为空（Node.js 事件循环与作为参数传递的事件不同）时退出。

Note

如果您使用运行时 v0.10.42，则需要使用 `context` 方法（`done`、`succeed` 和 `fail`）正确终止 Lambda 函数。有关更多信息，请参阅 [使用较早的 Node.js 运行时 v0.10.42 \(p. 28\)](#)。

上传此代码并将其作为 Lambda 函数（控制台）进行测试

- 在控制台中，使用以下信息创建 Lambda 函数：

- 使用 hello-world 蓝图。
- 建议将 nodejs6.10 指定为 runtime，不过您也可以选择 nodejs4.3。所提供的代码示例对于这两个版本都适用。

有关使用控制台创建 Lambda 函数的说明，请参阅 [创建简单的 Lambda 函数 \(p. 8\)](#)。

- 将模板代码替换为本部分中提供的代码，然后创建函数。
- 使用 Lambda 控制台中提供的名为 Hello World 的 Sample event template 测试 Lambda 函数。

使用回调参数

Node.js 运行时 v4.3 和 v6.10 支持可选的 `callback` 参数。您可以使用它显式将信息返回到调用方。一般语法为：

```
callback(Error error, Object result);
```

其中：

- `error` - 是可选参数，可用于提供失败的 Lambda 函数执行的结果。Lambda 函数成功时，您可以传递 `null` 作为第一个参数。
- `result` - 是可选参数，可用于提供成功的函数执行的结果。提供的 `result` 必须兼容 `JSON.stringify`。如果提供了错误，则忽略此参数。

Note

使用 `callback` 参数是可选的。如果您不使用可选的 `callback` 参数，则行为与您不使用任何参数调用 `callback()` 的情况相同。您可以在代码中指定 `callback` 以将信息返回到调用方。

如果您在代码中不使用 `callback`，AWS Lambda 将隐式调用它，返回值是 `null`。

在调用回调（显式或隐式）时，AWS Lambda 继续执行 Lambda 函数调用，直至 Node.js 事件循环为空。

以下为示例回调：

```
callback();      // Indicates success but no information returned to the caller.  
callback(null); // Indicates success but no information returned to the caller.  
callback(null, "success"); // Indicates success with information returned to the caller.  
callback(error); // Indicates error with error information returned to the caller.
```

AWS Lambda 将任何非 `null` 的 `error` 参数值视为已处理的异常。

请注意以下几点：

- 无论在调用 Lambda 函数时指定了什么调用类型（请参阅 [Invoke \(p. 425\)](#)），回调方法自动将 `error` 的非 `null` 的字符串表示形式记录到与 Lambda 函数关联的 Amazon CloudWatch Logs 流中。
- 如果同步调用了 Lambda 函数（使用 `RequestResponse` 调用类型），则回调返回响应正文，如下所示：
 - 如果 `error` 为 `null`，则将响应正文设置为 `result` 的字符串表示形式。
 - 如果 `error` 不为 `null`，则 `error` 值将填充到响应正文中。

Note

在调用 `callback(error, null)`（和 `callback(error)`）时，Lambda 将记录错误对象的前 256 KB。对于较大的错误对象，AWS Lambda 截断日志并在错误对象旁边显示文本 `Truncated by Lambda`。

Context 对象 (Node.js)

在执行 Lambda 函数时，它可以与 AWS Lambda 进行交互以获取有用的运行时信息，例如：

- AWS Lambda 终止您的 Lambda 函数之前的剩余时间量（超时是 Lambda 函数配置属性之一）。
- 与正在执行的 Lambda 函数关联的 CloudWatch 日志组和日志流。
- 返回到调用了 Lambda 函数的客户端的 AWS 请求 ID。可以使用此请求 ID 向 AWS Support 进行任何跟进查询。
- 如果通过 AWS 移动软件开发工具包调用 Lambda 函数，则可了解有关调用 Lambda 函数的移动应用程序的更多信息。

AWS Lambda 通过 `context` 对象提供此信息，服务将此对象作为第二个参数传递给 Lambda 函数处理程序。有关更多信息，请参阅 [Lambda 函数处理程序 \(Node.js\) \(p. 19\)](#)。

以下部分提供了使用 `context` 对象的示例 Lambda 函数，然后列出了所有可用的方法和属性。

示例

请考虑以下 Node.js 示例。处理程序通过 `context` 参数接收运行时信息。

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
    //console.log('Received event:', JSON.stringify(event, null, 2));
    console.log('value1 = ', event.key1);
```

```
console.log('value2 =', event.key2);
console.log('value3 =', event.key3);
console.log('remaining time =', context.getRemainingTimeInMillis());
console.log('functionName =', context.functionName);
console.log('AWSrequestID =', context.awsRequestId);
console.log('logGroupName =', context.logGroupName);
console.log('logStreamName =', context.logStreamName);
console.log('clientContext =', context.clientContext);
if (typeof context.identity !== 'undefined') {
    console.log('Cognito
    identity ID =', context.identity.cognitoIdentityId);
}
callback(null, event.key1); // Echo back the first key value
// or
// callback("some error type");
};
```

在本示例中，处理程序将 Lambda 函数的一些运行时信息记录到 CloudWatch。如果使用 Lambda 控制台调用该函数，控制台将在 Log output 部分中显示日志。您可使用此代码创建 Lambda 函数并使用控制台测试它。

在 AWS Lambda 控制台中测试此代码

1. 在控制台中，使用 hello-world 蓝图创建 Lambda 函数。在 runtime 中，选择 nodejs6.10。有关如何执行此操作的说明，请参阅 [创建简单的 Lambda 函数 \(p. 8\)](#)。
2. 测试此函数，然后也可更新代码以获取更多上下文信息。

Context 对象方法 (Node.js)

context 对象提供了以下方法。

context.getRemainingTimeInMillis()

返回当前正在执行的 Lambda 函数的大约剩余执行时间（在发生超时前）。超时是 Lambda 函数配置之一。当达到超时时间时，AWS Lambda 会终止您的 Lambda 函数。

您可利用该方法在函数执行期间检查剩余时间并在运行时采取适当的纠正措施。

一般语法为：

```
context.getRemainingTimeInMillis();
```

Context 对象属性 (Node.js)

context 对象提供您可以更新的以下属性：

callbackWaitsForEmptyEventLoop

默认值为 true。此属性仅在修改回调的默认行为时有用。默认情况下，回调将等待直至 Node.js 运行时事件循环为空，然后才冻结进程并将结果返回给调用方。您可以将此属性设置为 false，以请求 AWS Lambda 在调用 callback 后立即冻结进程，即使事件循环中存在事件也是如此。AWS Lambda 将冻结进程、任何状态数据以及 Node.js 事件循环中的事件（下次调用 Lambda 函数时，如果 AWS Lambda 选择使用冻结的进程，则处理事件循环中的任何剩余事件）。有关回调的更多信息，请参阅 [使用回调参数 \(p. 20\)](#)。

此外，context 对象提供了您可用来获取运行时信息的以下属性：

functionName

正在执行的 Lambda 函数的名称。

functionVersion

正在执行的 Lambda 函数版本。如果别名用于调用函数，`function_version` 将为别名指向的版本。
`invokedFunctionArn`

ARN 用于调用此函数。它可以是函数 ARN 或别名 ARN。非限定的 ARN 执行 `$LATEST` 版本，别名执行它指向的函数版本。

`memoryLimitInMB`

为 Lambda 函数配置的内存限制（以 MB 为单位）。您在创建 Lambda 函数时设置内存限制，并且随后可更改此限制。

`awsRequestId`

与请求关联的 AWS 请求 ID。这是返回到调用了 `invoke` 方法的客户端的 ID。

Note

如果 AWS Lambda 重试调用（例如，在处理 Kinesis 记录的 Lambda 函数引发异常的情况下）时，请求 ID 保持不变。

`logGroupName`

CloudWatch 日志组的名称，可从该日志组中查找由 Lambda 函数写入的日志。

`logStreamName`

CloudWatch 日志组的名称，可从该日志组中查找由 Lambda 函数写入的日志。每次调用 Lambda 函数时，日志流可能会更改，也可能不更改。

如果 Lambda 函数无法创建日志流，则该值为空。当向 Lambda 函数授予必要权限的执行角色未包括针对 CloudWatch 操作的权限时，可能会发生这种情况。

`identity`

有关 Amazon Cognito 身份提供商的信息（通过 AWS 移动软件开发工具包调用时）。它可以为空。

- `identity.cognitoIdentityId`
- `identity.cognitoIdentityPoolId`

有关特定移动平台准确值的更多信息，请参阅 AWS Mobile SDK for iOS Developer Guide 中的 [身份上下文](#) 以及“适用于 Android 的 AWS 移动软件开发工具包 Developer Guide”中的 [身份上下文](#)。

`clientContext`

通过 AWS 移动软件开发工具包进行调用时的客户端应用程序和设备的相关信息。它可以为空。利用 `clientContext`，您可获取以下信息：

- `clientContext.client.installation_id`
- `clientContext.client.app_title`
- `clientContext.client.app_version_name`
- `clientContext.client.app_version_code`
- `clientContext.client.app_package_name`
- `clientContext.Custom`

由移动客户端应用程序设置的自定义值。

- `clientContext.env.platform_version`
- `clientContext.env.platform`
- `clientContext.env.make`
- `clientContext.env.model`
- `clientContext.env.locale`

有关特定移动平台准确值的更多信息，请参阅 AWS Mobile SDK for iOS Developer Guide中的[客户端上下文](#)和适用于 Android 的 AWS 移动软件开发工具包 Developer Guide中的[客户端上下文](#)。

日志记录 (Node.js)

您的 Lambda 函数可包含日志记录语句。AWS Lambda 将这些日志写入 CloudWatch。如果您使用 Lambda 控制台调用 Lambda 函数，控制台将显示相同的日志。

以下 Node.js 语句生成了日志条目：

- `console.log()`
- `console.error()`
- `console.warn()`
- `console.info()`

例如，请考虑以下 Node.js 代码示例。

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
    //console.log('Received event:', JSON.stringify(event, null, 2));
    console.log('value1 =', event.key1);
    console.log('value2 =', event.key2);
    console.log('value3 =', event.key3);
    callback(null, event.key1); // Echo back the first key value
};
```

屏幕截图显示了 Lambda 控制台中的示例 Log output 部分，您还可在 CloudWatch 中查找这些日志。有关更多信息，请参阅[访问 AWS Lambda 的 Amazon CloudWatch 日志 \(p. 299\)](#)。

The screenshot shows the AWS Lambda execution results page. At the top, it says "Execution result: succeeded (logs)". Below that is a "Details" section with a note: "The area below shows the result returned by your function execution. Learn more about returning results from your function." A box contains the string "value1".

Summary

- Code SHA-256: yhxWtSs9l4Kb+WPTJ+VLnUoIyzuctEOuevh8ZIlg=
- Request ID: 9c454e0d-6d78-11e7-b93a-439d5c9ba65d
- Duration: 472.47 ms
- Billed duration: 500 ms
- Resources configured: 128 MB

Log output

The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. [Click here](#) to view the CloudWatch log group.

```
START RequestId: 9c454e0d-6d78-11e7-b93a-439d5c9ba65d Version: $LATEST
2017-07-20T18:24:06.610Z 9c454e0d-6d78-11e7-b93a-439d5c9ba65d value1 = value1
2017-07-20T18:24:06.629Z 9c454e0d-6d78-11e7-b93a-439d5c9ba65d value2 = value2
2017-07-20T18:24:06.629Z 9c454e0d-6d78-11e7-b93a-439d5c9ba65d value3 = value3
END RequestId: 9c454e0d-6d78-11e7-b93a-439d5c9ba65d
REPORT RequestId: 9c454e0d-6d78-11e7-b93a-439d5c9ba65d Duration: 472.47 ms Billed Duration: 500 ms Memory Size: 128 MB Max Memory Used: 18 MB
```

控制台在调用该函数时使用 `RequestResponse` 调用类型（同步调用），因此它会获得控制台显示的来自 AWS Lambda 的返回值 (`value1`)。

在 AWS Lambda 控制台中测试上述 Node.js 代码

- 在控制台中，使用 `hello-world` 蓝图创建 Lambda 函数。确保选择 Node.js 作为 runtime。有关如何执行此操作的说明，请参阅 [创建简单的 Lambda 函数 \(p. 8\)](#)。
- 使用 Lambda 控制台中提供的名为 Hello World 的 Sample event template 测试 Lambda 函数。您也可以更新该代码并尝试本部分中讨论的其他日志记录方法和属性。

如需分步指导，请参阅 [入门 \(p. 3\)](#)。

查找日志

可以通过以下方式找到您的 Lambda 函数写入的日志：

- 在 AWS Lambda 控制台中 - AWS Lambda 控制台中的 Log output 部分显示日志。
- 在响应标头中，当您以编程方式调用 Lambda 函数时 - 如果以编程方式调用 Lambda 函数，则可添加 `LogType` 参数以检索已写入 CloudWatch 日志的最后 4 KB 日志数据。AWS Lambda 在响应的 `x-amz-log-results` 头中返回此日志信息。有关更多信息，请参阅[调用](#)。

如果您使用 AWS CLI 调用该函数，则可指定带有值 `Tail` 的 `--log-type parameter` 来检索相同信息。

- 在 CloudWatch 日志中 - 要在 CloudWatch 中查找您的日志，您需要知道日志组名称和日志流名称。您可通过在代码中添加 `context.logGroupName` 和 `context.logStreamName` 方法来获取这类信息。当您运行 Lambda 函数时，在控制台或 CLI 中生成的日志将为您显示日志组名称和日志流名称。

函数错误 (Node.js)

如果您的 Lambda 函数通知 AWS Lambda 它未能正确执行，Lambda 会尝试将错误对象转换为字符串。考虑以下示例：

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
    // This example code only throws error.
    var error = new Error("something is wrong");
    callback(error);

};
```

当您调用此 Lambda 函数时，它将向 AWS Lambda 通知函数执行已完成但返回错误，并将错误信息传递到 AWS Lambda。AWS Lambda 会将错误信息返回给客户端：

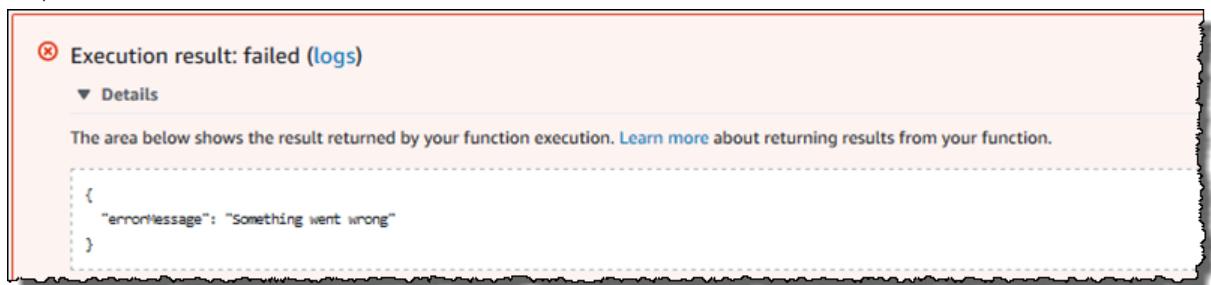
```
{
  "errorMessage": "something is wrong",
  "errorType": "Error",
  "stackTrace": [
    "exports.handler (/var/task/index.js:10:17)"
  ]
}
```

请注意，堆栈跟踪将以堆栈跟踪元素的 stackTrace JSON 数组形式返回。

取回错误信息的方式取决于在函数调用时客户端指定的调用类型：

- 如果某个客户端指定 RequestResponse 调用类型（即同步执行），该函数会将结果返回到执行调用的客户端。

例如，控制台始终使用 RequestResponse 调用类型，因此控制台将在 Execution result 部分中显示错误，如下所示：



相同信息也将发送到 CloudWatch，并且 Log output 部分将显示相同的日志。

Summary	Log output
Code SHA-256 U4b2T1IAJ6Jhw7VXNt W02s3RXctox6Ph3Jw3 7xQfO6g=	The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. Click here to view the CloudWatch log group.
Request ID 093bf2a1-6cba-11e7-b9ad-850729ae85a8	START RequestId: 093bf2a1-6cba-11e7-b9ad-850729ae85a8 Version: \$LATEST 2017-07-19T19:39:53.469Z 093bf2a1-6cba-11e7-b9ad-850729ae85a8 value3 = undefined 2017-07-19T19:39:53.469Z 093bf2a1-6cba-11e7-b9ad-850729ae85a8 value4 = undefined 2017-07-19T19:39:53.488Z 093bf2a1-6cba-11e7-b9ad-850729ae85a8 value5 = undefined 2017-07-19T19:39:53.489Z 093bf2a1-6cba-11e7-b9ad-850729ae85a8 {"errorMessage": "Something went wrong"} END RequestId: 093bf2a1-6cba-11e7-b9ad-850729ae85a8 REPORT RequestId: 093bf2a1-6cba-11e7-b9ad-850729ae85a8 Duration: 41.24 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 17 MB
Duration 41.24 ms	
Billed duration 100 ms	

- 如果某个客户端指定了 Event 调用类型（即异步执行），AWS Lambda 不会返回任何信息。相反，它将错误信息记录到 CloudWatch 日志。您还可 在 CloudWatch 指标中查看错误指标。

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 [了解重试行为 \(p. 146\)](#)。

测试上述 Node.js 代码（控制台）

- 在控制台中，使用 hello-world 蓝图创建 Lambda 函数。在 runtime 中，选择 Node.js；在 Role 中，选择 Basic execution role。有关如何执行此操作的说明，请参阅 [创建简单的 Lambda 函数 \(p. 8\)](#)。
- 将模板代码替换为此部分中提供的代码。
- 使用 Lambda 控制台中提供的名为 Hello World 的 Sample event template 测试 Lambda 函数。

函数错误处理

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，并直接在 AWS Step Functions 状态机中进行处理（重试或捕获）。有关更多信息，请参阅 [使用状态机处理错误情况](#)。

请将 CreateAccount 状态看作使用 Lambda 函数将客户的详细信息写入数据库的任务。

- 如果任务成功，会创建账户并发送欢迎电子邮件。
- 如果用户尝试创建的账户用户名已存在，Lambda 函数将出错，状态机会建议其他用户名，并重试账户创建过程。

以下代码示例演示了如何执行此操作。请注意，Node.js 中的自定义错误必须扩展错误原型。

```
exports.handler = function(event, context, callback) {
    function AccountAlreadyExistsError(message) {
        this.name = "AccountAlreadyExistsError";
        this.message = message;
    }
    AccountAlreadyExistsError.prototype = new Error();

    const error = new AccountAlreadyExistsError("Account is in use!");
    callback(error);
};
```

您可以配置 Step Functions，使用 Catch 规则捕获错误：

```
{  
    "StartAt": "CreateAccount",  
    "States": {  
        "CreateAccount": {  
            "Type": "Task",  
            "Resource": "arn:aws:lambda:us-east-1:123456789012:function:CreateAccount",  
            "Next": "SendWelcomeEmail",  
            "Catch": [  
                {  
                    "ErrorEquals": ["AccountAlreadyExistsError"],  
                    "Next": "SuggestAccountName"  
                }  
            ],  
            ...  
        }  
    }  
}
```

AWS Step Functions 可于运行时捕获错误，按照 Next 转换中指定的方式，[转换至 SuggestAccountName 状态](#)。

Note

Error 对象的名称属性必须匹配 ErrorEquals 值。

自定义错误处理机制使创建[无服务器](#)应用程序变得更加容易。此功能与 Lambda 编程模型 (p. 18) 支持的所有语言相集成，您可以任选编辑语言设计您的应用程序，并进行混合搭配。

要进一步了解如何使用 AWS Step Functions 和 AWS Lambda 创建您自己的无服务器应用程序，请参阅 [AWS Step Functions](#)。

使用较早的 Node.js 运行时 v0.10.42

截至 2017 年 5 月，AWS Lambda 支持 Node.js 6.10 和 Node.js 4.3。有关在创建 Lambda 函数时指定此运行时的信息，请参阅 [CreateFunction \(p. 387\)](#) 的 --runtime 参数。

Node v0.10.42 目前已淘汰。有关更多信息，请参阅 [运行时支持策略 \(p. 365\)](#)。您必须尽快将现有函数迁移到 AWS Lambda 上可用的较新 Node.js 运行时版本 (nodejs4.3 或 nodejs6.10)。注意，对于包含以 Node v0.10.42 运行时编写的函数的每个区域，您都必须遵循此过程。以下部分重点介绍 AWS Lambda 的运行时支持策略、运行时 v0.10.42 的特有行为，以及如何将现有函数尽快迁移到较新的版本 (nodejs4.3 或 nodejs6.10)。不迁移也不延期会导致以 Node v0.10.42 运行时编写的函数的任意调用返回无效参数值错误。注意，对于包含以 Node v0.10.42 运行时编写的函数的每个区域，您都必须遵循此过程。下一节介绍了运行时 v0.10.42 特有的行为，以及如何将现有函数迁移到较新的版本。

主题

- 将 Lambda 函数代码转换为较新的运行时 (p. 28)
- Node.js 运行时 v0.10.42 中的 context 方法 (p. 30)

将 Lambda 函数代码转换为较新的运行时

Node v0.10.42 目前已淘汰。有关更多信息，请参阅 [运行时支持策略 \(p. 365\)](#)。您必须尽快将现有函数迁移到 AWS Lambda 上可用的较新 Node.js 运行时版本 (nodejs4.3 或 nodejs6.10)。不迁移也不延期会导致以 Node v0.10.42 运行时编写的函数的任意调用返回无效参数错误。注意，对于包含以 Node v0.10.42 运行时编写的函数的每个区域，您都必须遵循此过程。

以下部分说明了如何将现有的 Lambda 函数代码迁移到较新的运行时：

1. 查看全部现有 Lambda 函数，并规划迁移。您可以通过如下方式获取函数列表，以及它们的版本和别名：

要列出使用蓝图的 Lambda 函数的更多信息，请参阅[使用运行时更新蓝图列出 Lambda 函数并更新到较新的运行时 \(p. 30\)](#)。

使用控制台列出 Lambda 函数：

- a. 登录 AWS 管理控制台并打开 Lambda 控制台。
- b. 选择 Runtime 列。该区域的所有 Lambda 函数将按其运行时值进行排序。
- c. 打开运行时值为 Node.js 的每个 Lambda 函数，然后选择 Configuration 选项卡。
- d. 选择 Qualifiers 下拉列表。
- e. 选择每个版本并查看其运行时。
- f. 选择每个别名，查看它指向的版本。
- g. 根据需要针对每个区域重复上述步骤。

2. 对于每个函数：

- a. 首先以手动更新方式或以 UPDATE 模式运行 nodejs-upgrade-functions 蓝图来更新运行时 (有关更多信息，请参阅[使用运行时更新蓝图列出 Lambda 函数并更新到较新的运行时 \(p. 30\)](#))。我们强烈建议您更新所用的所有上下文方法，并用回调方法进行替换。有关更多信息，请参阅[Node.js 运行时 v0.10.42 中的 context 方法 \(p. 30\)](#)。
- b. 测试并验证 Lambda 函数是否能通过对其行为的内部验证。如果无法通过验证，您可能需要更新 Lambda 代码以在新运行时中使用：
 - 有关 Node.js v6.10 中的更改列表，请参阅 GitHub 上 [v5 与 v6 之间的主要变更](#)。
 - 有关在 Node.js v4.3 中进行的更改的列表，请参阅 GitHub 上 [v0.10 与 v4 之间的 API 更改](#)。
- c. 一旦您的函数已成功调用，转换即完成。

3. 检查现有函数的版本和别名。您可以使用[使用 Lambda 控制台列出 Lambda 函数并更新到较新的运行时 \(p. 30\)](#)或[使用运行时更新蓝图列出 Lambda 函数并更新到较新的运行时 \(p. 30\)](#)获取每个函数的版本列表。对于每个版本，请执行以下操作：

- a. 将代码复制到 \$ LATEST。
- b. 从步骤 2 开始重复上述过程。
- c. 新版本完成后，重新发布代码。
- d. 将当前指向旧版本的别名更新为指向新发布的版本。
- e. 删除旧版本。

使用 CLI 列出 Lambda 函数并更新运行时

您可以使用 [ListFunctions \(p. 439\)](#) 命令返回全部 Lambda 函数的列表，并返回使用 v0.10 运行时创建的函数。以下代码示例演示了如何执行此操作：

```
#!/bin/bash

for REGION in $(aws ec2 describe-regions --output text --query 'Regions[].[RegionName]' | egrep -v 'ca-central-1|sa-east-1' | sort); do
    echo "...checking $REGION"
    echo " nodejs0.10 functions: "
    for i in $(aws lambda list-functions --output json --query 'Functions[*].[FunctionName, Runtime]' --region $REGION | grep -v nodejs4.3 | grep -v nodejs6.10 | grep -B1 nodejs | grep , | sort); do
        echo " -> $i"
    done
done

echo "This script only accounts for the \$LATEST versions of functions. You may need to take a closer look if you are using versioning."
```

对于返回的每个使用 v0.10 运行时创建的 Lambda 函数，请使用 [UpdateFunctionConfiguration \(p. 477\)](#) 命令将 `--runtime` 值设置为 `nodejs4.3` 或 `nodejs6.10`。

使用 Lambda 控制台列出 Lambda 函数并更新到较新的运行时

- 登录 AWS 管理控制台并打开 Lambda 控制台。
- 选择 Runtime 选项卡。该区域的所有 Lambda 函数将按其运行时值进行排序。
- 打开运行时值为 `node.js` 的每个 Lambda 函数，然后选择 Configuration 选项卡。
- 将 Runtime 值设置为 Node.js 4.3 或 Node.js 6.10。
- 根据需要针对每个区域重复此过程。

使用运行时更新蓝图列出 Lambda 函数并更新到较新的运行时

- 登录 AWS 管理控制台并打开 Lambda 控制台。
- 选择 Create a Lambda Function。
- 选择 `nodejs-upgrade-functions` 蓝图，使用它创建一个函数。
- 注意，该函数有以下可用的环境变量：
 - `MODE` = List 或 Backup 或 Upgrade
 - `TARGET_RUNTIME` = `nodejs4.3` 或 `nodejs6.10`
 - `EXCLUDED` = 要从处理中排除的函数名称的逗号分隔列表（不要在列表中包含空格）
- 要获取函数和版本列表，请从控制台调用函数，不要对变量值进行任何更改。
- 要在升级之前备份函数，请将 `MODE` 的值更改为 `Backup`，然后从控制台调用该函数。强烈建议在升级函数前执行该操作。
- 要更新函数的运行时值，请将 `MODE` 的值更改为 `Upgrade`，然后从控制台调用该函数。
- 根据需要针对每个区域重复此过程。
- 请注意：
 - 蓝图将您现有的 Node.js v1.0 函数保存为一个版本，并根据所选版本将 `$LATEST` 更新为 `nodejs4.3` 或 `nodejs6.10`。没有其他版本的函数可以升级。您可以使用该版本信息使所有现有应用程序指向该版本。
 - 蓝图不修改别名。指向该函数的任何别名都必须重新映射到新版本。有关更多信息，请参阅 [AWS Lambda 函数版本控制和别名 \(p. 264\)](#)。

Node.js 运行时 v0.10.42 中的 context 方法

Node.js 运行时 v0.10.42 不支持 Lambda 函数的回调参数，而运行时 v4.3 和 v6.10 支持。在使用运行时 v0.10.42 时，您使用以下 `context` 对象方法来正确终止 Lambda 函数。`Context` 对象支持 `done()`、`succeed()` 和 `fail()` 方法，您可用来终止 Lambda 函数。运行时 v4.3 和 v6.10 中也存在这些方法，用于提供向后兼容性。有关转换代码以使用运行时 v4.3 或 v6.10 的信息，请参阅 [将 Lambda 函数代码转换为较新的运行时 \(p. 28\)](#)。

`context.succeed()`

表示 Lambda 函数执行和所有回调已成功完成。一般语法如下：

```
context.succeed(Object result);
```

其中：

`result` - 是可选参数，可用于提供函数执行的结果。

提供的 `result` 必须兼容 `JSON.stringify`。如果 AWS Lambda 无法转化字符串或遇到其他错误，则会引发 `Unhandled` 异常，并将 `X-Amz-Function-Error` 响应头设为 `Unhandled`。

您可以在不使用任何参数的情况下调用此方法 (`succeed()`)，也可以传递 `null` 值 (`succeed(null)`)。

此方法的行为取决于 Lambda 函数调用中指定的调用类型。有关调用类型的更多信息，请参阅 [Invoke \(p. 425\)](#)。

- 如果使用 Event 调用类型（异步调用）调用 Lambda 函数，该方法将返回 HTTP status 202, request accepted 响应。
- 如果使用 RequestResponse 调用类型（同步调用）调用 Lambda 函数，该方法将返回 HTTP 状态 200 (OK)，并将响应正文设为 result 的字符串表示。

[context.fail\(\)](#)

表示 Lambda 函数执行和所有回调失败，生成了一个已处理的异常。一般语法显示如下：

```
context.fail(Error error);
```

其中：

`error` - 是可选参数，可用于提供 Lambda 函数执行的结果。

如果 `error` 值非 null，该方法会将响应正文设为 `error` 的字符串表示形式，并将对应日志写入 CloudWatch。如果 AWS Lambda 无法转化字符串或遇到其他错误，则会出现一个未处理的错误，并会将 `X-Amz-Function-Error` 头设为 `Unhandled`。

Note

对于来自 `context.fail(error)` 和 `context.done(error, null)` 的错误，Lambda 记录错误对象的前 256 KB 内容。对于较大的错误对象，AWS Lambda 截断错误并在错误对象旁边显示文本：`Truncated by Lambda`。

您可以在不使用任何参数的情况下调用此方法 (`fail()`)，也可以传递 null 值 (`fail(null)`)。

[context.done\(\)](#)

导致 Lambda 函数执行终止。

Note

该方法是对 `succeed()` 和 `fail()` 方法的补充，它允许使用“错误优先”的回调设计模式。它不提供多余的功能。

一般语法为：

```
context.done(Error error, Object result);
```

其中：

- `error` - 是可选参数，可用于提供失败的 Lambda 函数执行的结果。
- `result` - 是可选参数，可用于提供成功的函数执行的结果。提供的 `result` 必须兼容 `JSON.stringify`。如果提供了错误，则忽略此参数。

您可以在不使用任何参数的情况下调用此方法 (`done()`)，也可以传递 null (`done(null)`)。

AWS Lambda 将任何非 null 的 `error` 参数值视为已处理的异常。

函数行为取决于在 Lambda 调用时指定的调用类型。有关调用类型的更多信息，请参阅 [Invoke \(p. 425\)](#)。

- 无论调用类型如何，该方法都会自动将 `error` 的非 null 值的字符串表示记录到与 Lambda 函数关联的 Amazon CloudWatch Logs 流。

- 如果使用了 RequestResponse (同步) 调用类型调用 Lambda 函数，该方法将返回如下响应正文：
 - 如果 error 为 null，则将响应正文设为 result 的 JSON 表示形式。这类似于 context.succeed()。
 - 如果 error 不为 null 或该函数是使用 error 类型的单一参数调用的，则 error 值将填充在响应正文 中。

Note

对于来自 done(error, null) 和 fail(error) 的错误，Lambda 记录错误对象的前 256 KB 内容，对于较大的错误对象，AWS Lambda 将截断日志，并在错误对象旁边显示文本 Truncated by Lambda。

比较上下文和回调方法

如果您以前使用 Node.js 运行时 v0.10.42 创建了 Lambda 函数，您可以使用其中 context 对象方法 (done()、succeed() 和 fail()) 来终止 Lambda 函数。在 Node.js 运行时 v4.3 和 v6.10 中，支持这些方法的目的主要是向后兼容。我们建议您使用 callback (请参阅 [使用回调参数 \(p. 20\)](#))。以下 callback 示例与 context 对象方法等效：

- 以下示例显示了 context.done() 方法以及较新运行时中支持的对应等效 callback。

```
// Old way (Node.js runtime v0.10.42).
context.done(null, 'Success message');

// New way (Node.js runtime v4.3 or v6.10).
context.callbackWaitsForEmptyEventLoop = false;
callback(null, 'Success message');
```

Important

出于性能原因，AWS Lambda 可能会为 Lambda 函数的多次执行重用相同的 Node.js 进程。如果出现这种情况，AWS Lambda 在执行之间冻结 Node 进程，保留继续执行所需的状态信息。调用 context 方法时，AWS Lambda 立即冻结 Node 进程，而不等待与进程关联的事件循环清空。进程状态以及事件循环中的任何事件将冻结。再次调用函数时，如果 AWS Lambda 重用冻结进程，则函数使用其相同的全局状态继续执行（例如，开始处理保留在事件循环中的事件）。但是，在使用回调时，AWS Lambda 继续执行 Lambda 函数，直至事件循环清空。处理事件循环中的所有事件之后，AWS Lambda 冻结 Node 进程，包括 Lambda 函数中的任何状态变量。因此，如果您希望 context 方法具有相同行为，您必须将 context 对象属性 callbackWaitsForEmptyEventLoop 设置为 false。

- 以下示例显示了 context.succeed() 方法以及较新运行时中支持的对应等效 callback。

```
// Old way (Node.js runtime v0.10.42).
context.succeed('Success message');

// New way (Node.js runtime v4.3 or v6.10).
context.callbackWaitsForEmptyEventLoop = false;
callback(null, 'Success message');
```

- 以下示例显示了 context.fail() 方法以及较新运行时中支持的对应等效 callback。

```
// Old way (Node.js runtime v0.10.42).
context.fail('Fail object');

// New way (Node.js runtime v4.3 or v6.10).
context.callbackWaitsForEmptyEventLoop = false;
callback('Fail object', 'Failed result');
```

使用 Java 编写 Lambda 函数的编程模型

以下章节说明了在使用 Java 编写 Lambda 函数代码时常见的编程模式和核心概念的适用情况。

主题

- [Lambda 函数处理程序 \(Java\) \(p. 33\)](#)
- [Context 对象 \(Java\) \(p. 42\)](#)
- [日志记录 \(Java\) \(p. 44\)](#)
- [函数错误 \(Java\) \(p. 48\)](#)
- [对 Log4j™ 1.2 使用以前的自定义 Appender \(不推荐\) \(p. 50\)](#)
- [\(可选\) 创建用 Java 编写的 Lambda 函数 \(p. 52\)](#)

此外，请注意，AWS Lambda 提供了以下库：

- [aws-lambda-java-core](#) - 此库提供了 Context 对象、RequestStreamHandler 接口和 RequestHandler 接口。Context 对象 ([Context 对象 \(Java\) \(p. 42\)](#)) 提供有关您的 Lambda 函数的运行时信息。预定义接口提供一种定义 Lambda 函数处理程序的方法。有关更多信息，请参阅 [利用预定义接口创建处理程序 \(Java\) \(p. 39\)](#)。
- [aws-lambda-java-events](#) - 此库提供一些预定义类型，供您在编写 Lambda 函数以处理由 Amazon S3、Kinesis、Amazon SNS 和 Amazon Cognito 发布的事件时使用。这些类可帮助您处理事件，而不必自行编写自定义序列化逻辑。
- [Custom Appender for Log4j2.8](#) – 您可使用由 AWS Lambda 提供的自定义 Log4j (请参阅 [Apache Log4j 2](#)) Appender 从您的 Lambda 函数记录日志。每个对 Log4j 方法的调用 (如 log.debug() 或 log.error()) 都将生成一个 CloudWatch Logs 事件。自定义 Appender 称为 LambdaAppender 并且必须在 log4j2.xml 文件中使用。您必须在部署程序包 (.jar 文件) 中包含 aws-lambda-java-log4j2 项目 (artifactId : aws-lambda-java-log4j2)。有关更多信息，请参阅 [日志记录 \(Java\) \(p. 44\)](#)。
- [Custom Appender for Log4j1.2](#) – 您可使用由 AWS Lambda 提供的自定义 Log4j (请参阅 [Apache Log4j 1.2](#)) Appender 从您的 Lambda 函数记录日志。有关更多信息，请参阅 [日志记录 \(Java\) \(p. 44\)](#)。

Note

我们已结束对 Log4j v1.2 自定义 Appender 的支持。它不会再有例行更新，我们也不建议再使用它。

可通过 [Maven Central 存储库](#) 及在 [GitHub](#) 上找到这些库。

Lambda 函数处理程序 (Java)

在创建 Lambda 函数时，需要指定一个处理程序，以供 AWS Lambda 服务在代您执行 Lambda 函数时调用。

Lambda 支持以下两种处理程序创建方法：

- 直接加载处理程序方法（不必实现任何接口）。本章节介绍这种方法。
- 实现作为 `aws-lambda-java-core` 库一部分提供的标准接口（接口方法）。有关更多信息，请参阅 [利用预定义接口创建处理程序 \(Java\) \(p. 39\)](#)。

处理程序的一般语法如下：

```
outputType handler-name(inputType input, Context context) {  
    ...  
}
```

为使 AWS Lambda 成功调用处理程序，调用时的输入数据必须能够序列化为 `input` 参数的数据类型。

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）。为使 AWS Lambda 成功调用该处理程序，调用函数时的输入数据必须能够序列化为 `input` 参数的数据类型。
- **outputType** - 如果打算同步调用 Lambda 函数（使用 `RequestResponse` 调用类型），则可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 `outputType` 应为 `void`。例如，将 AWS Lambda 搭配 Amazon S3 或 Amazon SNS 之类的事情源使用时，这些事情源会使用 `Event` 调用类型调用 Lambda 函数。

- **inputType** 和 **outputType** 可为以下类型之一：
 - Java 基元类型（如 `String` 或 `int`）。
 - `aws-lambda-java-events` 库中的预定义 AWS 事件类型。

例如，`S3Event` 是该库中预定义的一种 POJO，它提供了从传入的 Amazon S3 事件读取信息的便捷方法。

- 此外，您也可以编写自己的 POJO 类。AWS Lambda 会根据该 POJO 类型自动序列化和反序列化输入、输出 JSON。

有关更多信息，请参阅 [处理程序输入/输出类型 \(Java\) \(p. 35\)](#)。

- 如果不需要，可以省略处理程序方法签名中的 `Context` 对象。有关更多信息，请参阅 [Context 对象 \(Java\) \(p. 42\)](#)。

例如，考虑以下 Java 示例代码。

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class Hello implements RequestHandler<Integer, String>{
    public String myHandler(int myCount, Context context) {
        return String.valueOf(myCount);
    }
}
```

在此示例中，输入为 `Integer` 类型，输出为 `String` 类型。如果您打包了此代码及依赖项并创建了 Lambda 函数，则应指定 `example.Hello::myHandler` ([package.class::method-reference](#)) 作为处理程序。

在此示例 Java 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 之类的事情源发布），也可以是您提供的自定义输入（如本示例中的 `Integer` 对象）或任何自定义数据对象。

有关使用此 Java 代码创建 Lambda 函数的说明，请参阅[\(可选\) 创建用 Java 编写的 Lambda 函数 \(p. 52\)](#)。

处理程序重载解决方案

如果 Java 代码中包含多个名称为 `handler` 的方法，则 AWS Lambda 依据以下规则选择调用方法：

1. 选择参数最多的方法。
2. 如果有两个或多个方法携带相同数量的参数，则 AWS Lambda 选择以 `context` 作为最后一个参数的方法。

如果这些方法都不带或都带 `Context` 参数，则行为不确定。

附加信息

以下主题提供了有关处理程序的更多信息。

- 有关处理程序输入和输出类型的更多信息，请参阅[处理程序输入/输出类型 \(Java\) \(p. 35\)](#)。
- 有关使用预定义接口创建处理程序的信息，请参阅[利用预定义接口创建处理程序 \(Java\) \(p. 39\)](#)。

如果您实现了这些接口，则可以在编译时验证处理程序方法签名。

- 如果 Lambda 函数引发了异常，AWS Lambda 将记录 CloudWatch 中指示出错的指标。有关更多信息，请参阅[函数错误 \(Java\) \(p. 48\)](#)。

处理程序输入/输出类型 (Java)

AWS Lambda 在执行 Lambda 函数时会调用该处理程序。第一个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）。

AWS Lambda 支持处理程序使用以下输入/输出类型：

- 简单 Java 类型（AWS Lambda 支持 String、Integer、Boolean、Map 和 List 类型）
- POJO (Plain Old Java Object) 类型
- 流类型（如果不使用 POJO，或如果 Lambda 的序列化方法不能满足需求，则您可以使用字节流实现。有关更多信息，请参阅[示例：使用流进行处理程序输入/输出 \(Java\) \(p. 38\)](#)。）

处理程序输入/输出：String 类型

下面的 Java 类展示了一个名为 myHandler 的使用 String 类型进行输入、输出的处理程序。

```
package example;

import com.amazonaws.services.lambda.runtime.Context;

public class Hello {
    public String myHandler(String name, Context context) {
        return String.format("Hello %s.", name);
    }
}
```

您可以为其他简单 Java 类型编写类似的处理程序函数。

Note

当您异步调用 Lambda 函数时，系统将忽略由 Lambda 函数返回的任何值。因此，您可能需要将返回类型设置为 void 以便在代码中表明这一点。有关更多信息，请参阅[Invoke \(p. 425\)](#)。

有关测试端到端示例的信息，请参阅[\(可选\) 创建用 Java 编写的 Lambda 函数 \(p. 52\)](#)。

处理程序输入/输出：POJO 类型

下面的 Java 类展示了一个名为 myHandler 的使用 POJO 进行输入、输出的处理程序。

```
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 {
        ...
    }
}
```

```
}

public static class ResponseClass {
    ...
}

public static ResponseClass myHandler(RequestClass request, Context context) {
    String greetingString = String.format("Hello %s, %s.", request.getFirstName(),
request.getLastName());
    return new ResponseClass(greetingString);
}
}
```

AWS Lambda 基于标准 bean 命名约定（请参阅 [Java EE 6 教程](#)）进行序列化。您应使用可变 POJO 及公有 getter 和 setter。

Note

您不应依赖序列化框架的任何其他功能，如注释。如果需要自定义序列化行为，您可以借助原始字节流自行实现序列化。

使用 POJO 进行输入、输出时，需要提供 RequestClass 和 ResponseClass 类型的实现。有关示例，请参阅 [示例：使用 POJO 进行处理程序输入/输出 \(Java\) \(p. 36\)](#)。

[示例：使用 POJO 进行处理程序输入/输出 \(Java\)](#)

假定应用程序事件会生成包含名字和姓氏的数据，如下所示：

```
{ "firstName": "John", "lastName": "Doe" }
```

在本示例中，处理程序将接收此 JSON 并返回字符串 "Hello John Doe"。

```
public static ResponseClass handleRequest(RequestClass request, Context context){
    String greetingString = String.format("Hello %s, %s.", request.firstName,
request.lastName);
    return new ResponseClass(greetingString);
}
```

要使用该处理程序创建 Lambda 函数，必须提供输入和输出类型的实现，如下面的 Java 示例所示。HelloPojo 类定义 handler 方法。

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class HelloPojo implements RequestHandler<RequestClass, ResponseClass>{

    public ResponseClass handleRequest(RequestClass request, Context context){
        String greetingString = String.format("Hello %s, %s.", request.firstName,
request.lastName);
        return new ResponseClass(greetingString);
    }
}
```

为了实施输入类型，将以下代码添加到单独的文件中并将其命名为 RequestClass.java。将其放在您的目录结构中 HelloPojo.java 类的旁边：

```
package example;
```

```
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() {  
    }  
}
```

为了实施输出类型，将以下代码添加到单独的文件中并将其命名为 ResponseClass.java。将其放在您的目录结构中 HelloPojo.java 类的旁边：

```
package example;  
  
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() {  
    }  
}
```

Note

必须提供 `get` 和 `set` 方法，以便 POJO 使用 AWS Lambda 内置的 JSON 串行器。通常不需要提供无参数的构造函数，但本例提供了其他构造函数，因而需要显式地提供零参数构造函数。

可以将该代码上传为 Lambda 函数，并进行如下测试：

- 使用上述代码文件创建部署程序包。
- 将部署程序包上传到 AWS Lambda 并创建 Lambda 函数。可使用控制台或 AWS CLI 完成此操作。

- 使用控制台或 CLI 手动调用该 Lambda 函数。手动调用 Lambda 函数时，可以使用所提供的示例 JSON 事件数据。例如：

```
{ "firstName": "John", "lastName": "Doe" }
```

有关更多信息，请参阅 [\(可选\) 创建用 Java 编写的 Lambda 函数 \(p. 52\)](#)。注意以下不同：

- 创建部署程序包时，不要忘了 aws-lambda-java-core 库依赖项。
- 创建 Lambda 函数时，指定 example.HelloPojo::handleRequest (*package.class::method*) 作为处理程序值。

示例：使用流进行处理程序输入/输出 (Java)

如果不想使用 POJO，或如果 Lambda 的序列化方法不能满足需求，则可以使用字节流实现。在这种情况下，可以使用 `InputStream` 和 `OutputStream` 作为处理程序的输入和输出类型。示例处理程序函数如下所示：

```
public void handler(InputStream inputStream, OutputStream outputStream, Context context)
    throws IOException{
    ...
}
```

注意：在这种情况下，处理程序函数使用参数同时接收请求和响应流。

下面是一个 Lambda 函数示例，它实现使用 `InputStream` 和 `OutputStream` 类型作为 `input` 和 `output` 参数的处理程序。

Note

输入有效负载必须是有效的 JSON，但输出流没有此类限制。支持任意字节数。

```
package example;

import java.io.InputStream;
import java.io.OutputStream;
import com.amazonaws.services.lambda.runtime.RequestStreamHandler;
import com.amazonaws.services.lambda.runtime.Context;

public class Hello implements RequestStreamHandler{
    public static void handler(InputStream inputStream, OutputStream outputStream, Context context)
        throws IOException {
        int letter;
        while((letter = inputStream.read()) != -1)
        {
            outputStream.write(Character.toUpperCase(letter));
        }
    }
}
```

您可以执行以下操作来测试代码：

- 使用上述代码创建部署程序包。
- 将部署程序包上传到 AWS Lambda 并创建 Lambda 函数。可使用控制台或 AWS CLI 完成此操作。
- 可以通过提供示例输入的方式手动调用代码。例如：

```
test
```

按照“入门”中提供的说明进行操作。有关更多信息，请参阅 [\(可选\) 创建用 Java 编写的 Lambda 函数 \(p. 52\)](#)。注意以下不同：

- 创建部署程序包时，不要忘了 aws-lambda-java-core 库依赖项。
- 创建 Lambda 函数时，指定 example.Hello::handler (*package.class::method*) 作为处理程序值。

利用预定义接口创建处理程序 (Java)

您可以使用 AWS Lambda Java 核心库 (aws-lambda-java-core) 提供的预定义接口之一来创建 Lambda 函数处理程序，以作为使用任意名称和参数编写您自己的处理程序方法的替代方案。有关处理程序的更多信息，请参阅 [Lambda 函数处理程序 \(Java\) \(p. 33\)](#)。

您可以实现预定义接口之一 (RequestStreamHandler 或 RequestHandler) 并为该接口提供的 handleRequest 方法提供实现。您可以根据使用标准 Java 类型还是自定义 POJO 类型作为处理程序的输入/输出 (AWS Lambda 自动序列化和反序列化输入和输出，以匹配您的数据类型) 来实现这些接口中的某一个，或使用 Stream 类型自定义序列化。

Note

aws-lambda-java-core 库中提供这些接口。

如果实现标准接口，它们可帮助您在编译时验证方法签名。

如果实现这些接口中的某一个，请在 Java 代码中指定 *package*。*class* 作为处理程序 (创建 Lambda 函数时)。例如，下面是“入门”中修改过的 create-function CLI 命令。注意 --handler 参数指定了“example.Hello”值：

```
aws lambda create-function \
--region region \
--function-name getting-started-lambda-function-in-java \
--zip-file fileb://deployment-package (zip or jar)
  path \
--role arn:aws:iam::account-id:role/lambda_basic_execution \
--handler example.Hello \
--runtime java8 \
--timeout 15 \
--memory-size 512
```

下面几节提供实现这些接口的示例。

示例 1：创建带自定义 POJO 输入/输出的处理程序 (利用 RequestHandler 接口)

本节中的示例 Hello 类实现 RequestHandler 接口。该接口定义 handleRequest() 方法，该方法接受事件数据作为 Request 类型的输入参数，返回 Response 类型的 POJO 对象：

```
public Response handleRequest(Request request, Context context) {
    ...
}
```

下面显示了带有 handleRequest() 方法示例实现的 Hello 类。对于该示例，我们假定事件数据由名字和姓氏组成。

```
package example;

import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.Context;

public class Hello implements RequestHandler<Request, Response> {

    public Response handleRequest(Request request, Context context) {
```

```
        String greetingString = String.format("Hello %s %s.", request.firstName,
request.lastName);
        return new Response(greetingString);
    }
}
```

例如，如果 Request 对象中的事件数据为：

```
{
    "firstName": "value1",
    "lastName" : "value2"
}
```

则该方法返回下面的 Response 对象：

```
{
    "greetings": "Hello value1 value2."
}
```

接下来，您需要实现 Request 和 Response 类。您可以使用下面的实现进行测试：

Request 类：

```
package example;

public class Request {
    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 Request(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public Request() {
    }
}
```

Response 类：

```
package example;

public class Response {
    String greetings;
```

```
public String getGreetings() {
    return greetings;
}

public void setGreetings(String greetings) {
    this.greetings = greetings;
}

public Response(String greetings) {
    this.greetings = greetings;
}

public Response() {
}
}
```

您可以按以下方式从该代码创建 Lambda 函数并测试端到端体验：

- 使用上述代码创建部署程序包。有关更多信息，请参阅 [创建部署程序包 \(Java\) \(p. 88\)](#)。
- 将部署程序包上传到 AWS Lambda 并创建 Lambda 函数。
- 使用控制台或 CLI 测试 Lambda 函数。您可以指定符合 Request 类中 getter 和 setter 规范的任何示例 JSON 数据，例如：

```
{
    "firstName": "John",
    "lastName" : "Doe"
}
```

Lambda 函数将在响应中返回下面的 JSON。

```
{
    "greetings": "Hello John, Doe."
}
```

按照“入门”中提供的说明进行操作（请参阅 [\(可选\) 创建用 Java 编写的 Lambda 函数 \(p. 52\)](#)）。注意以下不同：

- 创建部署程序包时，不要忘了 aws-lambda-java-core 库依赖项。
- 创建 Lambda 函数时，指定 example.Hello (*package.class*) 作为处理程序值。

示例 2：创建带流输入/输出的处理程序（利用 RequestStreamHandler 接口）

本示例中的 Hello 类实现 RequestStreamHandler 接口。该接口定义 handleRequest 方法，如下所示：

```
public void handleRequest(InputStream inputStream, OutputStream outputStream, Context
    context)
        throws IOException {
    ...
}
```

下面显示了带有 handleRequest() 处理程序示例实现的 Hello 类。该处理程序对传入的事件数据（例如：字符串“hello”）进行处理：简单地将其转换为大写字符串并返回。

```
package example;
```

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import com.amazonaws.services.lambda.runtime.RequestStreamHandler;
import com.amazonaws.services.lambda.runtime.Context;

public class Hello implements RequestStreamHandler {
    public void handleRequest(InputStream inputStream, OutputStream outputStream, Context context)
        throws IOException {
        int letter;
        while((letter = inputStream.read()) != -1)
        {
            outputStream.write(Character.toUpperCase(letter));
        }
    }
}
```

您可以按以下方式从该代码创建 Lambda 函数并测试端到端体验：

- 使用之前的代码创建部署程序包。
- 将部署程序包上传到 AWS Lambda 并创建 Lambda 函数。
- 使用控制台或 CLI 测试 Lambda 函数。您可以指定任意示例字符串数据，例如：

```
"test"
```

Lambda 函数将在响应中返回 TEST。

按照“入门”中提供的说明进行操作（请参阅[\(可选\) 创建用 Java 编写的 Lambda 函数 \(p. 52\)](#)）。注意以下不同：

- 创建部署程序包时，不要忘了 aws-lambda-java-core 库依赖项。
- 创建 Lambda 函数时，指定 example.Hello ([package.class](#)) 作为处理程序值。

Context 对象 (Java)

您通过 context 参数与 AWS Lambda 执行环境进行交互。context 对象让您能够访问 Lambda 执行环境中的有用信息。例如，您可以使用 context 参数确定与该函数关联的 CloudWatch 日志流，或借助 context 对象的 clientContext 属性了解有关调用该 Lambda 函数的应用程序（通过 AWS 移动软件开发工具包调用时）的更多信息。

4

context 对象属性有：

- `getMemoryLimitInMB()`：为 Lambda 函数配置的内存限制 (MB)。
- `getFunctionName()`：正在运行的 Lambda 函数的名称。
- `getFunctionVersion()`：正在执行的 Lambda 函数版本。如果别名用于调用函数，`getFunctionVersion` 将为别名指向的版本。
- `getInvokedFunctionArn()`：用于调用此函数的 ARN。它可以是函数 ARN 或别名 ARN。非限定的 ARN 执行 \$LATEST 版本，别名执行它指向的函数版本。
- `getAwsRequestId()`：与请求关联的 AWS 请求 ID。这是返回到调用 `invoke()` 的客户端的 ID。可以使用该请求 ID 与 AWS support 进行跟进调查。注意：AWS Lambda 重试函数（例如，当 Lambda 函数处理 Kinesis 记录引发异常时）时，请求 ID 保存不变。
- `getLogStreamName()`：特定 Lambda 函数执行的 CloudWatch 日志流名称。如果提供的 IAM 用户没有执行 CloudWatch 操作的权限，则为空。

- `getLogGroupName()`：与调用的 Lambda 函数关联的 CloudWatch 日志组名称。如果提供的 IAM 用户没有执行 CloudWatch 操作的权限，则为空。
- `getClientContext()`：客户端应用程序和设备的相关信息（通过 AWS 移动软件开发工具包调用时）。它可以为空。客户端上下文提供了客户端信息，如客户端 ID、应用程序名称、版本名称、版本代码和应用程序包名称。
- `getIdentity()`：有关 Amazon Cognito 身份提供商的信息（通过 AWS 移动软件开发工具包调用时）。它可以为空。
- `getRemainingTimeInMillis()`：函数被终止前剩余的执行时间（单位为毫秒）。在创建 Lambda 函数时，您需要设置最大时间限制，到达该时间限制后，AWS Lambda 会终止函数的执行。有关函数剩余执行时间的信息可用于指定接近超时时的函数行为。
- `getLogger()`：返回与 Context 对象关联的 Lambda 记录器。有关更多信息，请参阅 [日志记录 \(Java\) \(p. 44\)](#)。

下面的 Java 代码片段显示了一个打印部分上下文信息的处理程序函数。

```
public static void handler(InputStream inputStream, OutputStream outputStream, Context context) {  
  
    ...  
    System.out.println("Function name: " + context.getFunctionName());  
    System.out.println("Max mem allocated: " + context.getMemoryLimitInMB());  
    System.out.println("Time remaining in milliseconds: " +  
context.getRemainingTimeInMillis());  
    System.out.println("CloudWatch log stream name: " + context.getLogStreamName());  
    System.out.println("CloudWatch log group name: " + context.getLogGroupName());  
}
```

示例：使用上下文对象 (Java)

下面的 Java 代码示例演示如何利用 Context 对象在 Lambda 函数运行期间检索该函数的运行时信息。

```
package example;  
import java.io.InputStream;  
import java.io.OutputStream;  
import com.amazonaws.services.lambda.runtime.Context;  
  
public class Hello {  
    public static void myHandler(InputStream inputStream, OutputStream outputStream,  
Context context) {  
  
        int letter;  
        try {  
            while((letter = inputStream.read()) != -1)  
            {  
                outputStream.write(Character.toUpperCase(letter));  
            }  
            Thread.sleep(3000); // Intentional delay for testing the  
getRemainingTimeInMillis() result.  
        }  
        catch (Exception e)  
        {  
            e.printStackTrace();  
        }  
  
        // For fun, let us get function info using the context object.  
        System.out.println("Function name: " + context.getFunctionName());  
        System.out.println("Max mem allocated: " + context.getMemoryLimitInMB());  
        System.out.println("Time remaining in milliseconds: " +  
context.getRemainingTimeInMillis());  
    }  
}
```

```
        System.out.println("CloudWatch log stream name: " + context.getLogStreamName());
        System.out.println("CloudWatch log group name: " + context.getLogGroupName());
    }
```

您可以执行以下操作来测试代码：

- 使用上述代码创建部署程序包。
- 将部署程序包上传到 AWS Lambda 以创建 Lambda 函数。可使用控制台或 AWS CLI 完成此操作。
- 要测试 Lambda 函数，请使用 Lambda 控制台提供的“Hello World”Sample event。

您可以键入任意字符串，该函数将返回该字符串的大写形式。此外，您还会获得 context 对象提供的有用的功能信息。

按照“入门”中提供的说明进行操作。有关更多信息，请参阅 [\(可选\) 创建用 Java 编写的 Lambda 函数 \(p. 52\)](#)。注意以下不同：

- 创建部署程序包时，不要忘了 aws-lambda-java-core 库依赖项。
- 在创建 Lambda 函数时，指定 example.Hello::myHandler (*package.class::method*) 作为处理程序值。

日志记录 (Java)

您的 Lambda 函数可包含日志记录语句。AWS Lambda 将这些日志写入 CloudWatch。我们建议您使用以下一种方法写入日志。

适用于 Log4j™ 2 的自定义 Appender

AWS Lambda 推荐使用 Log4j 2 来提供自定义 Appender。您可使用由 Lambda 提供的自定义 Log4j (请参阅 [Apache log4j](#)) Appender 从您的 Lambda 函数记录日志。每个对 Log4j 方法的调用 (如 `log.debug()` 或 `log.error()`) 都将生成一个 CloudWatch Logs 事件。自定义 Appender 称为 `LambdaAppender` 并且必须在 `log4j2.xml` 文件中使用。您必须在部署程序包 (.jar 文件) 中包含 `aws-lambda-java-log4j2` 工件 (`artifactId:aws-lambda-java-log4j2`)。有关示例，请参阅 [示例 1：使用 Log4J v2.8 编写日志 \(p. 45\)](#)。

LambdaLogger.log()

每次调用 `LambdaLogger.log()` 都会得到一个 CloudWatch Logs 事件，假设事件大小在允许的限制内。有关 CloudWatch Logs 限制的信息，请参阅 Amazon CloudWatch 用户指南中的 [CloudWatch 日志限制](#)。有关示例，请参阅 [示例 2：使用 LambdaLogger \(Java\) 编写日志 \(p. 47\)](#)。

此外，您还可以在您的 Lambda 函数代码中使用以下语句来生成日志条目：

- `System.out()`
- `System.err()`

但请注意，AWS Lambda 将 `System.out` 和 `System.err` 返回的每一行都视为独立的事件。要使输出行与日志条目一一对应时，使用该对象会很方便。当日志条目有多行输出时，AWS Lambda 会尝试使用换行符分析它们，以识别独立的事件。例如，下面的语句将两个词 (“Hello”和“world”) 记录为两个独立事件：

```
System.out.println("Hello \n world");
```

如何查找日志

可以通过以下方式找到您的 Lambda 函数写入的日志：

- 在 CloudWatch Logs 中查找日志。`context` 对象（在 `aws-lambda-java-core` 库中）提供了 `getLogStreamName()` 和 `getLogGroupName()` 方法。借助这些方法，您可以找到日志写入到了哪个特定日志流中。
- 如果通过控制台调用 Lambda 函数，则调用类型始终为 `RequestResponse`（即同步执行），控制台将显示 Lambda 函数使用 `LambdaLogger` 对象写入的日志。AWS Lambda 还返回来自 `System.out` 和 `System.err` 方法的日志。
- 通过编程方法调用 Lambda 函数时，可以添加 `LogType` 参数以检索写入到 CloudWatch Logs 的最后 4 KB 日志数据。有关更多信息，请参阅 [Invoke \(p. 425\)](#)。AWS Lambda 在响应的 `x-amz-log-results` 头中返回该日志信息。使用 AWS Command Line Interface 调用函数时，可以为 `--log-type` 参数指定值 `Tail`。

日志记录示例 (Java)

本节提供适用于 Log4j 的自定义 Appender 和用于记录信息的 `LambdaLogger` 对象的使用示例。

示例 1：使用 Log4J v2.8 编写日志

- 下面显示了如何使用 Maven 构建项目以正确包含 Log4j v2.8 插件：
 - 对于 Maven `pom.xml`：

```
<dependencies>
  ...
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-lambda-java-log4j2</artifactId>
    <version>1.0.0</version>
  </dependency>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.8.2</version>
  </dependency>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.8.2</version>
  </dependency>
  ...
</dependencies>
```

- 如果使用 Maven shade 插件，请如下设置插件配置：

```
<plugins>
  ...
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-shade-plugin</artifactId>
    <version>2.4.3</version>
    <executions>
      <execution>
        <phase>package</phase>
        <goals>
          <goal>shade</goal>
        </goals>
        <configuration>
          <transformers>
            <transformer
```

```
implementation="com.github.edwgiz.mavenShadePlugin.log4j2CacheTransformer.PluginsCacheFileTransformer
    </transformer>
    </transformers>
</configuration>
</execution>
</executions>
<dependencies>
    <dependency>
        <groupId>com.github.edwgiz</groupId>
        <artifactId>maven-shade-plugin.log4j2-cachefile-transformer</artifactId>
        <version>2.8.1</version>
    </dependency>
</dependencies>
</plugin>
...
</plugins>
```

- 以下 Java 代码示例说明了如何对 Lambda 使用 Log4j：

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Hello {
    // Initialize the Log4j logger.
    static final Logger logger = LogManager.getLogger(Hello.class);

    public String myHandler(String name, Context context) {
        // System.out: One log statement but with a line break (AWS Lambda writes two
        // events to CloudWatch).
        System.out.println("log data from stdout \n this is continuation of
        system.out");

        // System.err: One log statement but with a line break (AWS Lambda writes two
        // events to CloudWatch).
        System.err.println("log data from stderr. \n this is a continuation of
        system.err");

        // Use log4j to log the same thing as above and AWS Lambda will log only one
        // event in CloudWatch.
        logger.debug("log data from log4j debug \n this is continuation of log4j
        debug");

        logger.error("log data from log4j err. \n this is a continuation of
        log4j.err");

        // Return will include the log stream name so you can look
        // up the log later.
        return String.format("Hello %s. log stream = %s", name,
        context.getLogStreamName());
    }
}
```

- 前面的示例使用以下 log4j2.xml 文件来加载属性

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<Configuration packages="com.amazonaws.services.lambda.runtime.log4j2.LambdaAppender">
    <Appenders>
        <Lambda name="Lambda">
            <PatternLayout>
                <pattern>%d{yyyy-MM-dd HH:mm:ss} %X{AWSRequestId} %-5p %c{1}:%L - %m%n</pattern>
            </PatternLayout>
        </Lambda>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="Lambda" />
        </Root>
    </Loggers>
</Configuration>
```

示例 2：使用 LambdaLogger (Java) 编写日志

以下 Java 代码示例使用 System 方法和 LambdaLogger 对象编写日志，说明它们在 AWS Lambda 将日志信息记录到 CloudWatch 时的差异。

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;

public class Hello {
    public String myHandler(String name, Context context) {

        // System.out: One log statement but with a line break (AWS Lambda writes two
        events to CloudWatch).
        System.out.println("log data from stdout \n this is continuation of system.out");

        // System.err: One log statement but with a line break (AWS Lambda writes two
        events to CloudWatch).
        System.err.println("log data from stderr \n this is continuation of system.err");

        LambdaLogger logger = context.getLogger();
        // Write log to CloudWatch using LambdaLogger.
        logger.log("log data from LambdaLogger \n this is continuation of logger.log");

        // Return will include the log stream name so you can look
        // up the log later.
        return String.format("Hello %s. log stream = %s",
            name,
            context.getLogStreamName());
    }
}
```

下面是 CloudWatch Logs 中的日志条目示例。

```
Event Data
▼ START RequestId: a2a11d08-71f2-11e5-94d5-c52dd5b87ed2 Version: $LATEST
  ▼ log data from stdout
    ▼ this is continuation of system.out
  ▼ log data from stderr
    ▼ this is continuation of system.err
  ▼ log data from LambdaLogger
    this is continuation of logger.log
  ▼ END RequestId: a2a11d08-71f2-11e5-94d5-c52dd5b87ed2
  ▼ REPORT RequestId: a2a11d08-71f2-11e5-94d5-c52dd5b87ed2 Duration: 50.24 ms Billed Duration: 100 ms Memory Size: 512 MB Max Memory Used: 28 MB
```

注意：

- 由于存在换行符，AWS Lambda 将每个 `System.out.println()` 和 `System.err.println()` 语句日志记录中的日志字符串分析为两个独立事件（请注意屏幕截图中的两个向下箭头）。
- `LambdaLogger.log()` 生成一个 CloudWatch 事件。

您可以执行以下操作来测试代码：

- 使用这些代码创建部署程序包。
- 将部署程序包上传到 AWS Lambda 以创建 Lambda 函数。
- 使用字符串 ("this is a test") 作为示例事件来测试您的 Lambda 函数。处理程序代码接收示例事件但不对其执行任何操作。它只演示如何写入日志。

按照“入门”中提供的说明进行操作。有关更多信息，请参阅 [\(可选\) 创建用 Java 编写的 Lambda 函数 \(p. 52\)](#)。注意以下不同：

- 创建部署程序包时，不要忘了 `aws-lambda-java-core` 库依赖项。
- 在创建 Lambda 函数时，指定 `example.Hello::myHandler` (`package.class::method`) 作为处理程序值。

函数错误 (Java)

如果 Lambda 函数引发异常，AWS Lambda 会识别失败并将异常信息序列化为 JSON 并将其返回。下面是一个错误消息示例：

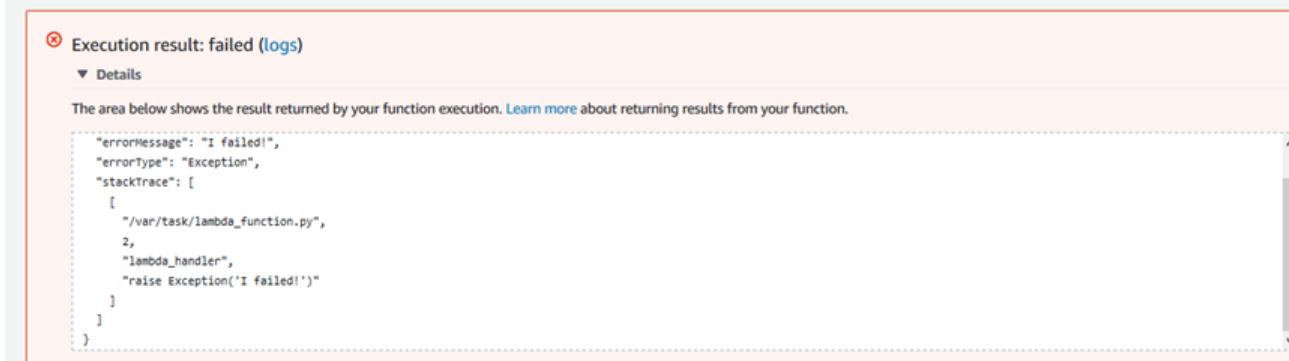
```
{
  "errorMessage": "Name John Doe is invalid. Exception occurred...",
  "errorType": "java.lang.Exception",
  "stackTrace": [
    "example.Hello.handler(Hello.java:9)",
    "sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)",
    "sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)",
    "sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)",
    "java.lang.reflect.Method.invoke(Method.java:497)"
  ]
}
```

注意：堆栈跟踪以 `stackTrace` JSON 数组的形式返回堆栈跟踪元素。

取回错误信息所用的方法取决于调用函数时指定的调用类型：

- `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 Execution result 部分中显示 AWS Lambda 返回的错误信息，如下图所示。



- 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。例如，如果 Lambda 函数的事件源为 Kinesis，则 AWS Lambda 会重试失败的函数，直到 Lambda 函数成功或流中的记录过期为止。

函数错误处理

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，并直接在 AWS Step Functions 状态机中进行处理（重试或捕获）。有关更多信息，请参阅[使用状态机处理错误情况](#)。

请将 CreateAccount 状态看作使用 Lambda 函数将客户的详细信息写入数据库的**任务**。

- 如果任务成功，会创建账户并发送欢迎电子邮件。
- 如果用户尝试创建的账户用户名已存在，Lambda 函数将出错，状态机会建议其他用户名，并重试账户创建过程。

以下代码示例演示了如何执行此操作。请注意，Java 中的自定义错误必须扩展 Exception 类型。

```
package com.example;

public static class AccountAlreadyExistsException extends Exception {
    public AccountAlreadyExistsException(String message) {
        super(message);
    }
}

package com.example;

import com.amazonaws.services.lambda.runtime.Context;

public class Handler {
    public static void CreateAccount(String name, Context context) throws
    AccountAlreadyExistsException {
        throw new AccountAlreadyExistsException ("Account is in use!");
    }
}
```

您可以配置 Step Functions，使用 Catch 规则捕获错误。Lambda 会自动将错误名称设置为运行时异常的完全限定类名称：

```
{
```

```
"StartAt": "CreateAccount",
"States": {
    "CreateAccount": {
        "Type": "Task",
        "Resource": "arn:aws:lambda:us-east-1:123456789012:function:CreateAccount",
        "Next": "SendWelcomeEmail",
        "Catch": [
            {
                "ErrorEquals": ["com.example.AccountAlreadyExistsException"],
                "Next": "SuggestAccountName"
            }
        ]
    },
    ...
}
```

AWS Step Functions 可于运行时捕获错误，按照 Next 转换中指定的方式，[转换至 SuggestAccountName 状态](#)。

自定义错误处理机制使创建[无服务器](#)应用程序变得更加容易。此功能与 [Lambda 编程模型 \(p. 18\)](#) 支持的所有语言相集成，您可以任选编辑语言设计您的应用程序，并进行混合搭配。

要进一步了解如何使用 AWS Step Functions 和 AWS Lambda 创建您自己的无服务器应用程序，请参阅 [AWS Step Functions](#)。

对 Log4j™ 1.2 使用以前的自定义 Appender (不推荐)

Note

我们已结束对 Log4j v1.2 自定义 Appender 的支持。它不会再有例行更新，我们也不建议再使用它。有关更多信息，请参阅 [Log4j 1.2](#)。

AWS Lambda 通过提供自定义 Appender 来支持 Log4j 1.2。您可使用由 Lambda 提供的自定义 Log4j (请参阅 [Apache log4j 1.2](#)) Appender 从您的 Lambda 函数记录日志。每个对 Log4j 方法的调用 (如 log.debug() 或 log.error()) 都将生成一个 CloudWatch Logs 事件。自定义 Appender 称为 LambdaAppender 并且必须在 log4j.properties 文件中使用。您必须在部署程序包 (.jar 文件) 中包含 aws-lambda-java-log4j 工件 (artifactId:aws-lambda-java-log4j)。有关示例，请参阅[示例：使用 Log4J v1.2 编写日志 \(不推荐\) \(p. 50\)](#)。

示例：使用 Log4J v1.2 编写日志 (不推荐)

Note

Log4j V1.x 已标记为终止使用。有关更多信息，请参阅 [Log4j 1.2](#)。

以下 Java 代码示例使用 System 方法和 Log4j 编写日志，说明它们在 AWS Lambda 将日志信息记录到 CloudWatch 时的差异。

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import org.apache.logging.log4j.Logger;

public class Hello {
    // Initialize the Log4j logger.
    static final Logger log = Logger.getLogger(Hello.class);

    public String myHandler(String name, Context context) {
```

```
// System.out: One log statement but with a line break (AWS Lambda writes two events to CloudWatch).
System.out.println("log data from stdout \n this is continuation of system.out");

// System.err: One log statement but with a line break (AWS Lambda writes two events to CloudWatch).
System.err.println("log data from stderr. \n this is a continuation of system.err");

// Use log4j to log the same thing as above and AWS Lambda will log only one event in CloudWatch.
log.debug("log data from log4j debug \n this is continuation of log4j debug");

log.error("log data from log4j err. \n this is a continuation of log4j.err");

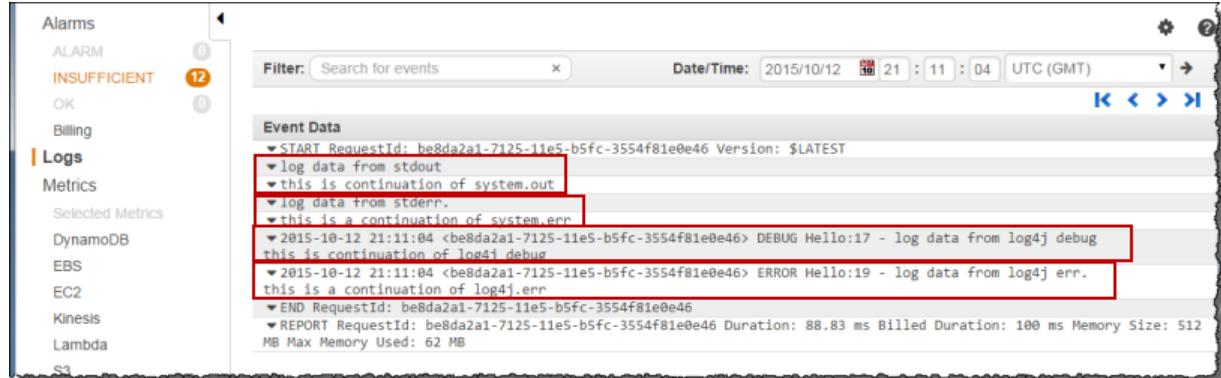
// Return will include the log stream name so you can look
// up the log later.
return String.format("Hello %s. log stream = %s",
context.getLogStreamName());
}
```

该示例使用 log4j.properties 文件 ([project-dir/src/main/resources/](#) directory)。

```
log = .
log4j.rootLogger = DEBUG, LAMBDA

#Define the LAMBDA appender
log4j.appender.LAMBDA=com.amazonaws.services.lambda.runtime.log4j.LambdaAppender
log4j.appender.LAMBDA.layout=org.apache.log4j.PatternLayout
log4j.appender.LAMBDA.layout.conversionPattern=%d{yyyy-MM-dd HH:mm:ss} <%X{AWSRequestId}>
%-5p %c{1}:%m%n
```

下面是 CloudWatch Logs 中的日志条目示例。



注意：

- 由于存在换行符，AWS Lambda 将每个 System.out.println() 和 System.err.println() 语句日志记录中的日志字符串分析为两个独立事件（请注意屏幕截图中的两个向下箭头）。
- Log4j 方法（log.debug() 和 log.error()）生成一个 CloudWatch 事件。
- AWS Lambda 运行时将 MDC 中的 AWSRequestId 添加到 log4j 上下文（请参阅 [log4j v 1.2 的 Class MDC 和 Class ThreadContext](#)）。为在所示日志中获取此值，我们在 %X{AWSRequestId} 文件中以转换模式添加了 log4j.properties。

您可以执行以下操作来测试代码：

- 使用这些代码创建部署程序包。在您的项目中，不要忘记将 `log4j.properties` 文件添加到 `project-dir/src/main/resources/` 目录中。
- 将部署程序包上传到 AWS Lambda 以创建 Lambda 函数。
- 使用字符串 ("this is a test") 作为示例事件来测试您的 Lambda 函数。处理程序代码接收示例事件但不对其实执行任何操作。它只演示如何写入日志。

按照“入门”中提供的说明进行操作。有关更多信息，请参阅 [\(可选\) 创建用 Java 编写的 Lambda 函数 \(p. 52\)](#)。注意以下不同：

- 创建部署程序包时，不要忘了 Log4j 1.2 的 `aws-lambda-java-log4j` 依赖项。
- 在创建 Lambda 函数时，指定 `example.Hello::myHandler` (`package.class::method`) 作为处理程序值。

(可选) 创建用 Java 编写的 Lambda 函数

蓝图提供了用 Python 或 Node.js 编写的示例代码。您可以在控制台中使用内联编辑器轻松修改示例。但是，如果您要使用 Java 为 Lambda 函数编写代码，则不会提供任何蓝图。此外，也不会为您提供用于在 AWS Lambda 控制台中编写 Java 代码的任何内联编辑器。

这意味着，您必须编写 Java 代码，还必须在控制台外部创建部署程序包。创建部署程序包后，您可以使用控制台来将该程序包上传到 AWS Lambda 以创建您的 Lambda 函数。您也可以使用控制台，通过手动调用该函数对其进行测试。

在本节中，您将使用以下 Java 代码示例创建 Lambda 函数。

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;

public class Hello {
    public String myHandler(int myCount, Context context) {
        LambdaLogger logger = context.getLogger();
        logger.log("received : " + myCount);
        return String.valueOf(myCount);
    }
}
```

编程模型详细介绍了如何编写 Java 代码，例如：AWS Lambda 支持的输入/输出类型。有关编程模型的更多信息，请参阅[使用 Java 编写 Lambda 函数的编程模型 \(p. 33\)](#)。现在，就该代码而言，您只需要注意以下事项：

- 打包并上传该代码以创建 Lambda 函数时，需要将 `example.Hello::myHandler` 方法引用指定为处理程序。
- 本示例中的处理程序使用 `int` 类型作为输入、使用 `String` 类型作为输出。

AWS Lambda 支持 JSON 可序列化类型和 `InputStream/OutputStream` 类型的输入/输出。调用此函数时，您应传递一个示例 `int`（例如，`123`）。

- 在本练习中，您将使用控制台手动调用此 Lambda 函数。控制台始终使用 `RequestResponse` 调用类型（同步），因此，您将在控制台中看到响应。
- 处理程序包含可选的 `Context` 参数。在本代码中，我们使用 `Context` 对象提供的 `LambdaLogger` 来将日志条目写入到 CloudWatch 日志中。有关使用 `Context` 对象的信息，请参阅[Context 对象 \(Java\) \(p. 42\)](#)。

首先，您需要将该代码及所有依赖项打包到部署程序包中。然后，您可以使用入门练习上传该程序包，以使用控制台创建并测试您的 Lambda 函数。

用于使用 Python 编写 Lambda 函数的编程模型

以下部分说明用 Python 编写 Lambda 函数代码时如何应用[常见的编程模式和核心概念](#)。

主题

- [Lambda 函数处理程序 \(Python\) \(p. 53\)](#)
- [Context 对象 \(Python\) \(p. 54\)](#)
- [日志记录 \(Python\) \(p. 56\)](#)
- [函数错误 \(Python\) \(p. 58\)](#)

Lambda 函数处理程序 (Python)

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。
- **context** - AWS Lambda 使用此参数向您的处理程序提供运行时信息。此参数为 `LambdaContext` 类型。
- (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 `NONE`，AWS Lambda 将返回 `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。

将此代码作为 Lambda 函数进行上传并测试

1. 保存该文件（例如，保存为 `hello_python.py`）。
2. 将该文件和所有依赖项打包到 `.zip` 文件中。在创建 `zip` 文件时，仅包含代码及其依赖项，而不包含文件夹。

有关说明，请参阅[创建部署程序包 \(Python\) \(p. 94\)](#)。

3. 使用控制台或 AWS CLI 上传 .zip 文件以创建 Lambda 函数。您在 Python 代码中指定创建 Lambda 函数时要用作处理程序的函数名称。有关使用控制台创建 Lambda 函数的说明，请参阅 [创建简单的 Lambda 函数 \(p. 8\)](#)。在此示例中，处理程序是 `hello_python.my_handler(file-name.function-name)`。请注意，[入门 \(p. 3\)](#) 使用了为 Lambda 函数提供示例代码的蓝图。在这种情况下，您已经有一个部署程序包。因此，在配置函数步骤中，您应选择上传一个 zip 文件。

以下 `create-function` AWS CLI 命令创建 Lambda 函数。在其他参数中，它指定 `--handler` 参数以指定处理程序名称。请注意，`--runtime` 参数指定 `python3.6`。您还可以使用 `python2.7`。

```
aws lambda create-function \
--region region \
--function-name HelloPython \
--zip-file fileb://deployment-package.zip \
--role arn:aws:iam::account-id:role/lambda_basic_execution \
--handler hello_python.my_handler \
--runtime python3.6 \
--timeout 15 \
--memory-size 512
```

Context 对象 (Python)

主题

- [示例 \(p. 54\)](#)
- [Context 对象方法 \(Python\) \(p. 55\)](#)
- [Context 对象属性 \(Python\) \(p. 55\)](#)

在执行 Lambda 函数时，它可以与 AWS Lambda 服务进行交互以获取有用的运行时信息，例如：

- AWS Lambda 终止您的 Lambda 函数之前的剩余时间量（超时是 Lambda 函数配置属性之一）。
- 与正在执行的 Lambda 函数关联的 CloudWatch 日志组和日志流。
- 返回到调用了 Lambda 函数的客户端的 AWS 请求 ID。可以使用此请求 ID 向 AWS Support 进行任何跟进查询。
- 如果通过 AWS 移动软件开发工具包调用 Lambda 函数，则可了解有关调用 Lambda 函数的移动应用程序的更多信息。

AWS Lambda 通过 `context` 对象提供此信息，服务将此对象作为第二个参数传递给 Lambda 函数处理程序。有关更多信息，请参阅 [Lambda 函数处理程序 \(Python\) \(p. 53\)](#)。

以下部分提供了使用 `context` 对象的示例 Lambda 函数，然后列出了所有可用的方法和属性。

示例

考虑以下 Python 示例。它有一个函数，此函数也是处理程序。处理程序通过作为参数传递的 `context` 对象接收运行时信息。

```
from __future__ import print_function

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())
```

此示例中的处理程序代码只打印部分运行时信息。每个打印语句均在 CloudWatch 中创建一个日志条目。如果您使用 Lambda 控制台调用函数，则控制台会显示日志。利用 `from __future__` 语句，可以编写与 Python 2 或 3 兼容的代码。

在 AWS Lambda 控制台中测试此代码

1. 在控制台中，使用 hello-world 蓝图创建 Lambda 函数。在 runtime 中，选择 Python 2.7。在 Handler 中，将 `lambda_function.lambda_handler` 替换为 `lambda_function.get_my_log_stream`。有关如何执行此操作的说明，请参阅 [创建简单的 Lambda 函数 \(p. 8\)](#)。
2. 测试此函数，然后也可更新代码以获取更多上下文信息。

以下部分提供了可用的 `context` 对象方法和属性的列表，可使用这些方法和属性来获取 Lambda 函数的运行时信息。

Context 对象方法 (Python)

`context` 对象提供了以下方法：

`get_remaining_time_in_millis()`

返回在 AWS Lambda 终止函数前剩余的执行时间（以毫秒为单位）。

Context 对象属性 (Python)

`context` 对象提供了以下属性：

`function_name`

正在执行的 Lambda 函数的名称。

`function_version`

正在执行的 Lambda 函数版本。如果别名用于调用函数，`function_version` 将为别名指向的版本。

`invoked_function_arn`

ARN 用于调用此函数。它可以是函数 ARN 或别名 ARN。非限定的 ARN 执行 `$LATEST` 版本，别名执行它指向的函数版本。

`memory_limit_in_mb`

为 Lambda 函数配置的内存限制（以 MB 为单位）。您在创建 Lambda 函数时设置内存限制，并且随后可更改此限制。

`aws_request_id`

与请求关联的 AWS 请求 ID。这是返回到调用了 `invoke` 方法的客户端的 ID。

Note

如果 AWS Lambda 重试调用（例如，在处理 Kinesis 记录的 Lambda 函数引发异常的情况下）时，请求 ID 保持不变。

`log_group_name`

CloudWatch 日志组的名称，可从该日志组中查找由 Lambda 函数写入的日志。

log_stream_name

CloudWatch 日志流的名称，可从该日志流中查找由 Lambda 函数写入的日志。每次调用 Lambda 函数时，日志流可能会更改，也可能不更改。

如果 Lambda 函数无法创建日志流，则该值为空。当向 Lambda 函数授予必要权限的执行角色未包括针对 CloudWatch Logs 操作的权限时，可能会发生这种情况。

identity

通过 AWS 移动软件开发工具包进行调用时的 Amazon Cognito 身份提供商的相关信息。它可以为空。

- identity.cognito_identity_id
- identity.cognito_identity_pool_id

client_context

通过 AWS 移动软件开发工具包进行调用时的客户端应用程序和设备的相关信息。它可以为空。

- client_context.client.installation_id
- client_context.client.app_title
- client_context.client.app_version_name
- client_context.client.app_version_code
- client_context.client.app_package_name
- client_context.custom

由移动客户端应用程序设置的自定义值的 dict。

- client_context.env

由 AWS 移动软件开发工具包提供的环境信息的 dict。

除了上面列出的选项，您还可以使用适用于 [Python \(p. 308\)](#) 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。

日志记录 (Python)

您的 Lambda 函数可包含日志记录语句。AWS Lambda 将这些日志写入 CloudWatch。如果您使用 Lambda 控制台调用 Lambda 函数，控制台将显示相同的日志。

以下 Python 语句生成日志条目：

- print 语句。
- logging 模块中的 Logger 函数（例如，logging.Logger.info 和 logging.Logger.error）。

print 和 logging.* 函数将日志写入 CloudWatch Logs 中，而 logging.* 函数将额外信息写入每个日志条目中，例如时间戳和日志级别。

例如，考虑以下 Python 代码示例。

```
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
def my_logging_handler(event, context):
    logger.info('got event{}'.format(event))
    logger.error('something went wrong')
    return 'Hello from Lambda!'
```

由于代码示例使用 `logging` 模块将信息写入日志中，因此您也可以在日志中获取额外信息，例如时间戳和日志级别。日志级别标识日志的类型，例如 `[INFO]`、`[ERROR]` 和 `[DEBUG]`。

您也可以在 CloudWatch 中找到这些日志。有关更多信息，请参阅 [访问 AWS Lambda 的 Amazon CloudWatch 日志 \(p. 299\)](#)。

可以在代码中使用 `print` 语句而不是使用 `logging` 模块，如以下 Python 示例所示：

```
from __future__ import print_function
def lambda_handler(event, context):
    print('this will also show up in cloud watch')
    return 'Hello World!'
```

在此示例中，仅将传递到打印方法的文本发送到 CloudWatch。日志条目将不会具有 `logging.*` 函数返回的额外信息。利用 `from __future__` 语句，可以编写与 Python 2 或 3 兼容的代码。

The screenshot shows the AWS Lambda execution results for a successful function call. It includes a summary of the function's SHA-256 code hash, request ID, duration, and billed duration. The log output section displays the single log entry: "Hello World!" and the CloudWatch log group details.

Summary	Log output
Code SHA-256 Mj6sNoYuSX2UjGoXUy 64+R/UvzWB0xJc7VdX HRN6hqY=	START RequestId: 6a6a240b-6cc4-11e7-8734-6f6a02ed0559 Version: \$LATEST this will also show up in cloud watch END RequestId: 6a6a240b-6cc4-11e7-8734-6f6a02ed0559 REPORT RequestId: 6a6a240b-6cc4-11e7-8734-6f6a02ed0559 Duration: 0.18 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 17 MB

当调用此函数时，控制台使用 `RequestResponse` 调用类型（同步调用）。因此它从控制台显示的 AWS Lambda 中取回返回值（“Hello world！”）。

测试之前的 Python 代码（控制台）

1. 在控制台中，使用 `hello-world-python` 蓝图创建 Lambda 函数。在 `runtime` 中，选择 Python 2.7。在 `Handler` 中，将 `lambda_function.lambda_handler` 替换为 `lambda_function.my_other_logging_handler`，然后在 `Role` 中，选择 `Basic execution role`。您还可以将蓝图提供的代码替换为本节中的代码。有关使用控制台创建 Lambda 函数的分步说明，请参阅 [创建简单的 Lambda 函数 \(p. 8\)](#)。
2. 将模板代码替换为此部分中提供的代码。
3. 使用 Lambda 控制台中提供的名为 Hello World 的 Sample event template 测试 Lambda 函数。

查找日志

可查找 Lambda 函数写入的日志，如下所示：

- 在 AWS Lambda 控制台中 - AWS Lambda 控制台中的 Log output 部分显示这些日志。
 - 在响应标头中，当您以编程方式调用 Lambda 函数时 - 如果您以编程方式调用 Lambda 函数，则可添加 LogType 参数以检索已写入 CloudWatch 日志的最后 4 KB 的日志数据。AWS Lambda 在响应的 x-amz-log-results 标头中返回该日志信息。有关更多信息，请参阅 [Invoke \(p. 425\)](#)。
- 如果您使用 AWS CLI 调用该函数，则可指定带有值 Tail 的 --log-type parameter 来检索相同信息。
- 在 CloudWatch 日志中 - 要在 CloudWatch 中查找您的日志，您需要知道日志组名称和日志流名称。可以使用代码中的 context.logGroupName 和 context.logStreamName 属性来获取此信息。在运行 Lambda 函数时，控制台或 CLI 中生成的日志将会向您显示日志组名称和日志流名称。

函数错误 (Python)

如果 Lambda 函数引发异常，AWS Lambda 会识别失败，将异常信息序列化为 JSON 并将其返回。考虑以下示例：

```
def always_failed_handler(event, context):
    raise Exception('I failed!')
```

在调用此 Lambda 函数时，它将引发异常，并且 AWS Lambda 返回以下错误消息：

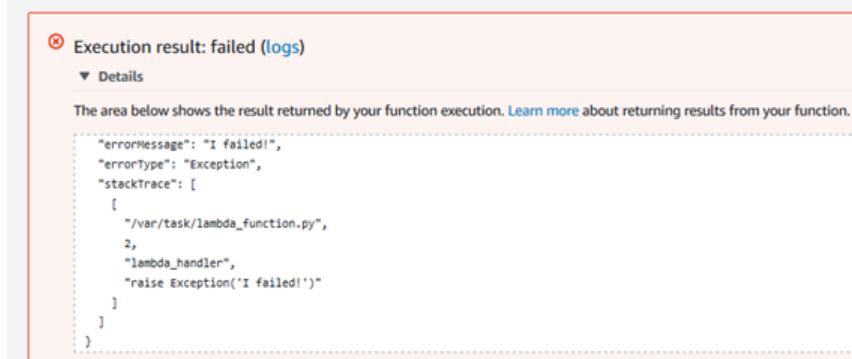
```
{
    "errorMessage": "I failed!",
    "stackTrace": [
        [
            "/var/task/lambda_function.py",
            3,
            "my_always_fails_handler",
            "raise Exception('I failed!')"
        ]
    ],
    "errorType": "Exception"
}
```

注意：堆栈跟踪以 stackTrace JSON 数组的形式返回堆栈跟踪元素。

取回错误信息的方式取决于在函数调用时客户端指定的调用类型：

- 如果客户端指定 RequestResponse 调用类型（即同步执行），则它会将结果返回发出此调用的客户端。

例如，控制台始终使用 RequestResponse 调用类型，因此控制台将在 Execution result 部分中显示错误，如下所示：



相同的信息也将发送到 CloudWatch，并且 Log output 部分显示相同的日志。

The screenshot shows the AWS Lambda CloudWatch Log group interface. It displays a summary table with details like Request ID, Billed duration, and Max memory used. Below the summary is a section titled "Log output" which contains a log entry. The log entry starts with "START RequestId: b5f4b0df-d47a-11e7-bac5-d145e32e9b46 Version: \$LATEST", followed by an exception trace: "I failed: Exception", "Traceback (most recent call last):", "File "/var/task/lambda_function.py", line 2, in lambda_handler", "raise Exception('I failed!')", "Exception: I failed!". The log entry ends with "END RequestId: b5f4b0df-d47a-11e7-bac5-d145e32e9b46" and "REPORT RequestId: b5f4b0df-d47a-11e7-bac5-d145e32e9b46 Duration: 0.82 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 22 MB".

- 如果客户端指定 Event 调用类型（即异步执行），则 AWS Lambda 将不会返回任何信息。相反，它将错误信息记录到 CloudWatch 日志。您还可在 CloudWatch 指标中查看错误指标。

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。

测试之前的 Python 代码（控制台）

- 在控制台中，使用 hello-world 蓝图创建 Lambda 函数。在 runtime 中，选择 Python 3.6 或 Python 2.7。在 Handler 中，将 `lambda_function.lambda_handler` 替换为 `lambda_function.always_failed_handler`。有关如何执行此操作的说明，请参阅 [创建简单的 Lambda 函数 \(p. 8\)](#)。
- 将模板代码替换为此部分中提供的代码。
- 使用 Lambda 控制台中提供的名为 Hello World 的 Sample event template 测试 Lambda 函数。

函数错误处理

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，并直接在 AWS Step Functions 状态机中进行处理（重试或捕获）。有关更多信息，请参阅[使用状态机处理错误情况](#)。

请将 `CreateAccount` 状态看作使用 Lambda 函数将客户的详细信息写入数据库的[任务](#)。

- 如果任务成功，会创建账户并发送欢迎电子邮件。
- 如果用户尝试创建的账户用户名已存在，Lambda 函数将出错，状态机会建议其他用户名，并重试账户创建过程。

以下代码示例演示了如何执行此操作。请注意，Python 中的自定义错误必须扩展 `Exception` 类型。

```
class AccountAlreadyExistsException(Exception): pass

def create_account(event, context):
    raise AccountAlreadyExistsException('Account is in use!')
```

您可以配置 Step Functions，使用 `Catch` 规则捕获错误。Lambda 会自动将错误名称设置为运行时异常的简单类名称：

```
{
  "StartAt": "CreateAccount",
  "States": {
    "CreateAccount": {
```

```
"Type": "Task",
"Resource": "arn:aws:lambda:us-east-1:123456789012:function:CreateAccount",
"Next": "SendWelcomeEmail",
"Catch": [
    {
        "ErrorEquals": ["AccountAlreadyExistsException"],
        "Next": "SuggestAccountName"
    }
],
...
}
```

AWS Step Functions 可于运行时捕获错误，按照 Next 转换中指定的方式，[转换至 SuggestAccountName 状态](#)。

自定义错误处理机制使创建[无服务器](#)应用程序变得更加容易。此功能与 Lambda 编程模型 (p. 18) 支持的所有语言相集成，您可以任选编辑语言设计您的应用程序，并进行混合搭配。

要进一步了解如何使用 AWS Step Functions 和 AWS Lambda 创建您自己的无服务器应用程序，请参阅 [AWS Step Functions](#)。

使用 Go 编写 Lambda 函数的编程模型

以下章节说明了在使用 Go 编写 Lambda 函数代码时[常见的编程模式和核心概念](#)的适用情况。

主题

- [Lambda 函数处理程序 \(Go\) \(p. 60\)](#)
- [Context 对象 \(Go\) \(p. 63\)](#)
- [日志记录 \(Go\) \(p. 65\)](#)
- [函数错误 \(Go\) \(p. 66\)](#)
- [使用环境变量 \(Go\) \(p. 68\)](#)

此外，请注意 AWS Lambda 将提供以下内容：

- [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 [Lambda 函数处理程序 \(Go\) \(p. 60\)](#)。
- [github.com/aws/aws-lambda-go/lambdacontext](#)：访问 [Context 对象 \(Go\) \(p. 63\)](#) 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。

Lambda 函数处理程序 (Go)

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：
 - context：Context 对象 (Go) (p. 63)。
 - 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 string) (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，以利用通过 Context 对象 (Go) (p. 63) 提供的信息。
 - name string：变量名称为 name 的输入类型，其值将在 return 语句中返回。
 - string error：返回标准错误信息。有关自定义错误处理的更多信息，请参阅函数错误 (Go) (p. 66)。
 - return fmt.Sprintf("Hello %s!", name), nil：只返回格式化“Hello”问候语和您在处理程序签名中提供的姓名。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。

Note

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：

```
# request
{
    "Answer": "Jim is 33 years old!"
}
```

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 [函数错误 \(Go\) \(p. 66\)](#)。

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)`

使用全局状态最大化性能

要最大限度地提高性能，您应声明并修改独立于您的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。这意味着，如果您可以安全地更改全局状态，则可以确信这些更改将需要新的执行上下文，且不会从在前一个执行上下文定向的函数调用引入锁定或不稳定的行为。有关更多信息，请参阅 [使用 AWS Lambda 函数的最佳实践 \(p. 362\)](#)。

```
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 {
    invokeCount = invokeCount + 1
    log.Print(myObjects)
    return invokeCount
}

func main() {
    lambda.Start(LambdaHandler)
}
```

下一步

[Context 对象 \(Go\) \(p. 63\)](#)

Context 对象 (Go)

在执行 Lambda 函数时，它可以与 AWS Lambda 进行交互以获取有用的运行时信息，例如：

- AWS Lambda 终止您的 Lambda 函数之前的剩余时间量（超时是 Lambda 函数配置属性之一）。
- 与正在执行的 Lambda 函数关联的 [CloudWatch](#) 日志组和日志流。
- 返回到调用了 Lambda 函数的客户端的 AWS 请求 ID。可以使用此请求 ID 向 AWS Support 进行任何跟进查询。
- 如果通过 AWS 移动软件开发工具包调用 Lambda 函数，则可了解有关调用 Lambda 函数的移动应用程序的更多信息。
- 除了下面列出的选项，您还可以使用适用于 [转到 \(p. 310\)](#) 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。

AWS Lambda 通过 `context.Context` 对象提供此信息，服务将此对象作为参数传递给 Lambda 函数处理程序。有关更多信息，请参阅 [有效处理程序签名 \(p. 62\)](#)。

以下部分提供了使用 `context` 对象的示例 Lambda 函数，然后列出了所有可用的方法和属性。

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.CognitoPoolID)
}

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.CognitoPoolID)` 将打印该信息（在本例中为 `CognitoPoolID`）。

监控函数的执行时间

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 {
    deadline, _ := ctx.Deadline()
    for {
        select {
        case <- time.Until(deadline).Truncate(100 * time.Millisecond):
            return "Finished before timing out."
        default:
            log.Print("hello!")
            time.Sleep(50 * time.Millisecond)
        }
    }
}

func main() {
    lambda.Start(LongRunningHandler)
}
```

Lambda 上下文库提供了以下全局变量：

- `MemoryLimitInMB`：为 Lambda 函数配置的内存限制（MB）。
- `FunctionName`：正在运行的 Lambda 函数的名称。
- `FunctionVersion`：正在执行的 Lambda 函数版本。如果别名用于调用函数，`FunctionVersion` 将为别名指向的版本。
- `LogStreamName`：特定 Lambda 函数执行的 CloudWatch 日志流名称。如果提供的 IAM 用户没有执行 CloudWatch 操作的权限，则为空。
- `LogGroupName`：与调用的 Lambda 函数关联的 CloudWatch 日志组名称。如果提供的 IAM 用户没有执行 CloudWatch 操作的权限，则为空。

Lambda context 对象还包含以下属性：

- `AwsRequestID`：与请求关联的 AWS 请求 ID。这是返回调用 Lambda 函数的客户端的 ID。可以使用此请求 ID 向 AWS Support 进行任何跟进查询。注意：AWS Lambda 重试函数（例如，当 Lambda 函数处理 Kinesis 记录引发异常时）时，请求 ID 保存不变。
- `ClientContext`：客户端应用程序和设备的相关信息（通过 AWS 移动软件开发工具包调用时）。它可以为空。客户端上下文提供了客户端信息，如客户端 ID、应用程序名称、版本名称、版本代码和应用程序包名称。
- `Identity`：在上述示例中记录。有关 Amazon Cognito 身份提供商的信息（通过 AWS 移动软件开发工具包调用时）。它可以为空。
- `InvokedFunctionArn`：用于调用此函数的 ARN。它可以是函数 ARN 或别名 ARN。非限定的 ARN 执行 `$LATEST` 版本，别名执行它指向的函数版本。

下一步

[日志记录 \(Go\) \(p. 65\)](#)

日志记录 (Go)

您的 Lambda 函数可包含日志记录语句。AWS Lambda 将这些日志写入 CloudWatch。如果您使用 Lambda 控制台调用 Lambda 函数，控制台将显示相同的日志。

例如，请考虑以下示例。

```
package main

import (
    "log"
    "github.com/aws/aws-lambda-go/lambda"
)

func HandleRequest() {
    log.Print("Hello from Lambda")
}

func main() {
    lambda.Start(HandleRequest)
}
```

通过导入 `log` 模块，Lambda 将写入其他日志信息，例如时间戳。

您也可以在 CloudWatch 中分析这些日志。有关更多信息，请参阅 [访问 AWS Lambda 的 Amazon CloudWatch 日志 \(p. 299\)](#)。

可以在代码中使用 `print` 语句而不是使用 `log` 模块，如下所示：

```
package main

import (
    "fmt"
    "github.com/aws/aws-lambda-go/lambda"
)

func HandleRequest() {
    fmt.Print("Hello from Lambda")
}

func main() {
    lambda.Start(HandleRequest)
}
```

```
}
```

在此示例中，仅将传递到打印方法的文本发送到 CloudWatch。日志条目将不会具有 `log.Println` 函数返回的额外信息。此外，写入 `stdout` 或 `stderr` 的任何记录器将与 Go 函数无缝集成，这些日志将自动被发送到 CloudWatch 日志。

当调用此函数时，控制台使用 `RequestResponse` 调用类型（同步调用）。因此它从 AWS Lambda 获取返回值（“Hello from Lambda!”）。

查找日志

可查找 Lambda 函数写入的日志，如下所示：

- 在 AWS Lambda 控制台中 - AWS Lambda 控制台中的 Log output 部分显示日志。
- 在响应标头中，当您以编程方式调用 Lambda 函数时 - 如果您以编程方式调用 Lambda 函数，则可添加 `LogType` 参数以检索已写入 CloudWatch 日志的最后 4 KB 的日志数据。AWS Lambda 在响应的 `x-amz-log-results` 标头中返回该日志信息。有关更多信息，请参阅[Invoke \(p. 425\)](#)。

如果您使用 AWS CLI 调用该函数，则可指定带有值 `Tail` 的 `--log-type parameter` 来检索相同信息。

- 在 CloudWatch 日志中 - 要在 CloudWatch 中查找您的日志，您需要知道日志组名称和日志流名称。您可以使用[Context 对象 \(Go\) \(p. 65\)](#) 库中的 `context.logGroupName` 和 `context.logStreamName` 全局变量获取此信息。在运行 Lambda 函数时，控制台或 CLI 中生成的日志将会向您显示日志组名称和日志流名称。

下一步

[函数错误 \(Go\) \(p. 66\)](#)

函数错误 (Go)

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)
}
```

这将返回：

```
{ "errorMessage": "something went wrong!" }
```

函数错误处理

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，并直接在 AWS Step Functions 状态机中进行处理（重试或捕获）。有关更多信息，请参阅[使用状态机处理错误情况](#)。

请将 `CreateAccount` 状态看作使用 Lambda 函数将客户的详细信息写入数据库的**任务**。

- 如果任务成功，会创建账户并发送欢迎电子邮件。
- 如果用户尝试创建的账户用户名已存在，Lambda 函数将出错，状态机会建议其他用户名，并重试账户创建过程。

以下代码示例演示了如何执行此操作。

```
package main

type CustomError struct {}

func (e *CustomError) Error() string {
    return "bad stuff happened..."
}

func MyHandler() (string, error) {
    return "", &CustomError{}
}
```

AWS Step Functions 可于运行时捕获错误，按照 Next 转换中指定的方式，[转换至 SuggestAccountName 状态](#)。

自定义错误处理机制使创建[无服务器](#)应用程序变得更加容易。此功能与 Lambda [编程模型 \(p. 18\)](#) 支持的所有语言相集成，您可以任选编辑语言设计您的应用程序，并进行混合搭配。

要进一步了解如何使用 AWS Step Functions 和 AWS Lambda 创建您自己的无服务器应用程序，请参阅[AWS Step Functions](#)。

处理意外错误

Lambda 可能会由于您无法控制的原因而失败，例如网络中断。这些是特殊情况。在 Go 中，[panic](#) 解决了这些问题。如果您的代码 panic，Lambda 将尝试捕获错误并将其序列化为标准错误 json 格式。Lambda 还会尝试将 panic 的值插入函数的 CloudWatch 日志。在返回响应后，Lambda 将自动重新创建该函数。如果您觉得有必要，您可以在您的代码中包含 panic 函数，以自定义错误响应。

```
package main

import (
    "errors"

    "github.com/aws/aws-lambda-go/lambda"
)

func handler(string) (string, error) {
    panic(errors.New("Something went wrong"))
}

func main() {
    lambda.Start(handler)
}
```

这将返回以下 json 格式的堆栈：

```
{
    "errorMessage": "Something went wrong",
    "errorType": "errorString",
    "stackTrace": [
        {
            "function": "main.main.func1",
            "line": 10,
            "file": "main.main"
        }
    ]
}
```

```
        "path": "github.com/aws/aws-lambda-go/lambda/function.go",
        "line": 27,
        "label": "(*Function).Invoke.function"
    },
    ...
]
}
```

使用环境变量 (Go)

要在 Go 中访问[环境变量](#) (p. 354) , 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 %. years old\n", os.Getenv("NAME"), os.Getenv("AGE"))
}
```

默认情况下 , Lambda 会配置以下环境变量 : [适用于 Lambda 函数的环境变量](#) (p. 366)。

使用 C# 编写 Lambda 函数的编程模型

以下章节说明了在使用 C# 编写 Lambda 函数代码时[常见的编程模式和核心概念](#)的适用情况。

主题

- [Lambda 函数处理程序 \(C#\)](#) (p. 69)
- [Context 对象 \(C#\)](#) (p. 72)
- [日志记录 \(C#\)](#) (p. 73)
- [函数错误 \(C#\)](#) (p. 74)

此外 , 请注意 AWS Lambda 将提供以下内容 :

- `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 \(Context 对象 \(C#\)\)](#) (p. 72) 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.SNSEvents

这些程序包都可以在 [Nuget 程序包](#) 中找到。

Lambda 函数处理程序 (C#)

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。

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 \(p. 71\)](#)。

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 [创建部署程序包 \(C#\) \(p. 79\)](#)。

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": "1.0.0"  
    },  
    "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 内核 CLI \(p. 79\)](#)。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。

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。通过使用该签名创建处理程序，Lambda 将同步执行该函数并在返回或超时之前等待执行完成（最多 5 分钟）。例如：

```
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);
}
```

Context 对象 (C#)

通过将 `ILambdaContext` 参数添加到方法中，您可以获取有关 Lambda 函数如何与 AWS Lambda 运行时交互的有用信息。作为回报，AWS Lambda 将提供运行时详细信息，例如与该函数有关的 CloudWatch 日志流或者可调用函数的客户端的 ID，您可以通过 `context` 对象的属性访问这些信息。

为实现此目的，请使用以下签名创建方法：

```
public void Handler(string Input, ILambdaContext context)
```

`context` 对象属性有：

- `MemoryLimitInMB`：为 Lambda 函数配置的内存限制 (MB)。
- `FunctionName`：正在运行的 Lambda 函数的名称。
- `FunctionVersion`：正在执行的 Lambda 函数版本。如果别名用于调用函数，`FunctionVersion` 将为别名指向的版本。
- `InvokedFunctionArn`：用于调用此函数的 ARN。它可以是函数 ARN 或别名 ARN。非限定的 ARN 执行 `$LATEST` 版本，别名执行它指向的函数版本。
- `AwsRequestId`：与请求关联的 AWS 请求 ID。这是返回调用 Lambda 函数的客户端的 ID。可以使用该请求 ID 与 AWS support 进行跟进调查。注意：AWS Lambda 重试函数（例如，当 Lambda 函数处理 Kinesis 记录引发异常时）时，请求 ID 保存不变。

- `LogStreamName`：特定 Lambda 函数执行的 CloudWatch 日志流名称。如果提供的 IAM 用户没有执行 CloudWatch 操作的权限，则为空。
- `LogGroupName`：与调用的 Lambda 函数关联的 CloudWatch 日志组名称。如果提供的 IAM 用户没有执行 CloudWatch 操作的权限，则为空。
- `ClientContext`：客户端应用程序和设备的相关信息（通过 AWS 移动软件开发工具包调用时）。它可以为空。客户端上下文提供了客户端信息，如客户端 ID、应用程序名称、版本名称、版本代码和应用程序包名称。
- `Identity`：有关 Amazon Cognito 身份提供商的信息（通过 AWS 移动软件开发工具包调用时）。它可以为空。
- `RemainingTime`：函数被终止前剩余的执行时间。在创建 Lambda 函数时，您需要设置最大时间限制，到达该时间限制后，AWS Lambda 会终止函数的执行。有关函数剩余执行时间的信息可用于指定接近超时的函数行为。这是 `TimeSpan` 域。
- `Logger`：与 `ILambdaContext` 对象关联的 Lambda 记录器。有关更多信息，请参阅 [日志记录 \(C#\) \(p. 73\)](#)。

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);
}
```

日志记录 (C#)

您的 Lambda 函数可包含记录语句，而 AWS Lambda 会将这些日志写入 CloudWatch Logs。

在 C# 编程模型中，有三种在函数中记录数据的方法：

- 使用 C# `Console` 类提供的静态的 `Write` 或 `WriteLine` 方法。CloudWatch Logs 将记录使用 `Console.Write` 或类似方法写入标准输出或标准错误的所有信息。

```
public class ProductService
{
    public async Task<Product> DescribeProduct(DescribeProductRequest request)
    {
        Console.WriteLine("DescribeProduct invoked with Id " + request.Id);
        return await catalogService.DescribeProduct(request.Id);
    }
}
```

- 使用 `Amazon.Lambda.Core.LambdaLogger` 类的 `Log` 方法。这是可在应用程序内任意位置使用的静态类。要使用该类，您必须纳入 `Amazon.Lambda.Core` 库。

```
using Amazon.Lambda.Core;

public class ProductService
{
    public async Task<Product> DescribeProduct(DescribeProductRequest request)
    {
        LambdaLogger.Log("DescribeProduct invoked with Id " + request.Id);
        return await catalogService.DescribeProduct(request.Id);
    }
}
```

每次调用 `LambdaLogger.Log` 都会得到一个 CloudWatch Logs 事件，假设事件大小在允许的限制内。有关 CloudWatch Logs 限制的信息，请参阅 Amazon CloudWatch 用户指南中的 [CloudWatch 日志限制](#)。

- 使用 `ILambdaContext` 中的记录器。您的方法中的 `ILambdaContext` 对象（如已指定）包含一个表示 `LambdaLogger` 的 `Logger` 属性。下面是如何使用此方法的示例：

```
public class ProductService
{
    public async Task<Product> DescribeProduct(DescribeProductRequest request,
        ILambdaContext context)
    {
        context.Logger.Log("DescribeProduct invoked with Id " + request.Id);
        return await catalogService.DescribeProduct(request.Id);
    }
}
```

如何查找日志

可以通过以下方式找到您的 Lambda 函数写入的日志：

- 在 CloudWatch Logs 中查找日志。`ILambdaContext` 对象提供 `LogStreamName` 和 `LogGroupName` 属性。借助这些属性，您可以找到写入日志的特定日志流。
- 如果通过控制台调用 Lambda 函数，则调用类型始终为 `RequestResponse`（即同步执行），控制台将显示 Lambda 函数使用 `LambdaLogger` 对象写入的日志。AWS Lambda 还返回来自 `Console.WriteLine` 和 `Console.Write` 方法的日志。
- 通过编程方法调用 Lambda 函数时，可以添加 `LogType` 参数以检索写入到 CloudWatch Logs 的最后 4 KB 日志数据。有关更多信息，请参阅 [Invoke \(p. 425\)](#)。AWS Lambda 在响应的 `x-amz-log-results` 头中返回该日志信息。使用 AWS Command Line Interface 调用函数时，可以为 `--log-type` 参数指定值 `Tail`。

函数错误 (C#)

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 \(p. 71\)](#)），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& bNeedsSecurityCheck)",
            "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": {
            "errorType": "WebException",
            "errorMessage": "An error occurred while sending the request. SSL peer certificate or
SSH remote key was not OK",
            "stackTrace": [
                "at System.Net.HttpWebRequest.EndGetResponse(IAsyncResult asyncResult)",
                "at System.Threading.Tasks.TaskFactory`1.FromAsyncCoreLogic(IAsyncResult iar,
Func`2 endFunction, Action`1 endAction, Task`1 promise, Boolean requiresSynchronization)",
                "---- End of stack trace from previous location where exception was thrown ----",
                "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
                "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task
task)",
                "at System.Runtime.CompilerServices.TaskAwaiter`1.GetResult()"
            ]
        }
    }
}
```

```
"at LambdaDemo107.LambdaEntryPoint.<GetUriResponse>d__1.MoveNext()"
],
"cause": {
    "errorType": "HttpRequestException",
    "errorMessage": "An error occurred while sending the request.",
    "stackTrace": [
        "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
        "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)",
        "at System.Net.Http.HttpClient.<FinishSendAsync>d__58.MoveNext()",
        "--- End of stack trace from previous location where exception was thrown ---",
        "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
        "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)",
        "at System.Net.HttpWebRequest.<SendRequest>d__63.MoveNext()",
        "--- End of stack trace from previous location where exception was thrown ---",
        "at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)",
        "at
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)",
        "at System.Net.HttpWebRequest.EndGetResponse(IAsyncResult asyncResult)"
    ],
"cause": {
    "errorType": "CurlException",
    "errorMessage": "SSL peer certificate or SSH remote key was not OK",
    "stackTrace": [
        "at System.Net.Http.CurlHandler.ThrowIfCURLError(CURLcode error)",
        "at
System.Net.Http.CurlHandler.MultiAgent.FinishRequest(StrongToWeakReference`1 easyWrapper,
CURLcode messageResult)"
    ]
}
}
}
}
```

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 [了解重试行为 \(p. 146\)](#)。

函数错误处理

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，并直接在 AWS Step Functions 状态机中进行处理（重试或捕获）。有关更多信息，请参阅[使用状态机处理错误情况](#)。

请将 `CreateAccount` 状态看作使用 Lambda 函数将客户的详细信息写入数据库的**任务**。

- 如果任务成功，会创建账户并发送欢迎电子邮件。
- 如果用户尝试创建的账户用户名已存在，Lambda 函数将出错，状态机会建议其他用户名，并重试账户创建过程。

以下代码示例演示了如何执行此操作。请注意，C# 中的自定义错误必须扩展 `Exception` 类型。

```
namespace Example {
    public class AccountAlreadyExistsException : Exception {
        public AccountAlreadyExistsException(String message) :
            base(message)
    }
}

namespace Example {
    public class Handler {
        public static void CreateAccount() {
            throw new AccountAlreadyExistsException("Account is in use!");
        }
    }
}
```

您可以配置 Step Functions，使用 `Catch` 规则捕获错误。Lambda 会自动将错误名称设置为运行时异常的简单类名称：

```
{
  "StartAt": "CreateAccount",
  "States": {
    "CreateAccount": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:CreateAccount",
      "Next": "SendWelcomeEmail",
      "Catch": [
        {
          "ErrorEquals": [ "AccountAlreadyExistsException" ],
          "Next": "SuggestAccountName"
        }
      ]
    },
    ...
  }
}
```

AWS Step Functions 可于运行时捕获错误，按照 `Next` 转换中指定的方式，[转换至 SuggestAccountName 状态](#)。

自定义错误处理机制使创建[无服务器](#)应用程序变得更加容易。此功能与 [Lambda 编程模型 \(p. 18\)](#) 支持的所有语言相集成，您可以任选编辑语言设计您的应用程序，并进行混合搭配。

要进一步了解如何使用 AWS Step Functions 和 AWS Lambda 创建您自己的无服务器应用程序，请参阅[AWS Step Functions](#)。

创建部署程序包

要创建 Lambda 函数，首先需要创建 Lambda 函数部署程序包（包含代码和所有依赖项的 `.zip` 或 `.jar` 文件）。在创建 `zip` 文件时，仅包含代码及其依赖项，而不包含文件夹。

- [创建部署程序包 \(Node.js\) \(p. 78\)](#)
- [创建部署程序包 \(C#\) \(p. 79\)](#)
- [创建部署程序包 \(Go\) \(p. 87\)](#)
- [创建部署程序包 \(Java\) \(p. 88\)](#)
- [创建部署程序包 \(Python\) \(p. 94\)](#)

创建部署程序包 (Node.js)

要创建 Lambda 函数，首先需要创建 Lambda 函数部署程序包（包含代码和所有依赖项的 .zip 文件）。

您可自行创建部署程序包或直接在 Lambda 控制台中编写代码，在后一种情况下，控制台将为您创建并上传部署程序包，从而创建您的 Lambda 函数。请记下以下内容来确定您是否可使用该控制台创建 Lambda 函数：

- 简单场景 - 如果自定义代码只需要 AWS 软件开发工具包库，则可以使用 AWS Lambda 控制台中的内联编辑器。使用控制台可以编辑代码并将代码上传到 AWS Lambda。控制台会将代码及相关的配置信息压缩到 Lambda 服务能够运行的部署程序包中。

您还可以在控制台中测试代码（使用示例事件数据手动调用代码）。

Note

Lambda 服务预装了适用于 Node.js 的 AWS 开发工具包。

- 高级场景 - 如果编写的代码需要用到其他资源（如使用图形库进行图像处理），或需要使用 AWS CLI 代替控制台，则需要先创建 Lambda 函数部署程序包，然后再使用控制台或 CLI 上传部署程序包。

Note

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。

以下是创建部署程序包的示例过程（在控制台外）。假设您需要创建包含 filename.js 代码文件的部署程序包，并且您的代码使用 async 库。

- 打开文本编辑器，并编写您的代码。保存文件（例如，filename.js）。

在创建 Lambda 函数时，您将使用此文件名指定处理程序。

- 在同一目录中，使用 npm 安装您的代码所依赖的库。例如，如果您的代码使用 async 库，则使用以下 npm 命令。

```
npm install async
```

- 之后，您的目录将具有以下结构：

```
filename.js
node_modules/async
node_modules/async/lib
node_modules/async/lib/async.js
node_modules/async/package.json
```

- 压缩文件夹的内容，即您的部署程序包（例如，sample.zip）。

然后，在创建您的 Lambda 函数时指定此 .zip 文件名作为部署程序包。

如果您希望包含您自己的二进制文件（包括本机二进制文件），只需将这些文件打包为上传的 Zip 文件，然后在从 Node.js 或您之前开始的其他过程中调用这些文件时引用它们（包括所创建的 Zip 文件中的相对路径）。确保在函数代码开头包含以下内容：`process.env['PATH'] = process.env['PATH'] + ':' + process.env['LAMBDA_TASK_ROOT']`

有关在 Lambda 函数程序包中包含本机二进制文件的更多信息，请参阅[在 AWS Lambda 中运行可执行文件](#)。

创建部署程序包 (C#)

.NET 内核 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 内核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 核心运行时。.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。

Note

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 \(p. 85\)](#) 或 [.NET 内核 CLI \(p. 79\)](#) 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 内核 CLI \(p. 79\)](#)
- [AWS Toolkit for Visual Studio \(p. 85\)](#)

.NET 内核 CLI

借助 .NET 内核 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 内核 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 之外创建独立于平台的项目，这种做法将特别有用。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 内核运行时的位置，然后输入以下内容：

```
dotnet new -all
```

您将看到以下内容：

Templates	Language	Tags	Short Name	
Console Application	F#, VB	Common/Console	console	[C#],
Class library	F#, VB	Common/Library	classlib	[C#],
Unit Test Project	F#, VB	Test/MSTest	mstest	[C#],

xUnit Test Project	xunit	[C#],
F#, VB Test/xUnit		
ASP.NET Core Empty	web	[C#],
F# Web/Empty		
ASP.NET Core Web App (Model-View-Controller)	mvc	[C#],
F# Web/MVC		
ASP.NET Core Web App	razor	[C#]
Web/MVC/Razor Pages		
ASP.NET Core with Angular	angular	[C#]
Web/MVC/SPA		
ASP.NET Core with React.js	react	[C#]
Web/MVC/SPA		
ASP.NET Core with React.js and Redux	reactredux	[C#]
Web/MVC/SPA		
ASP.NET Core Web API	webapi	[C#],
F# Web/WebAPI		
global.json file	globaljson	
Config		
NuGet Config	nugetconfig	
Config		
Web Config	webconfig	
Config		
Solution File	sln	
Solution		
Razor Page	page	
Web/ASP.NET		
MVC ViewImports	viewimports	
Web/ASP.NET		
MVC ViewStart	viewstart	
Web/ASP.NET		

Examples:

```
dotnet new mvc --auth None --framework netcoreapp1.1

dotnet new mvc --framework netcoreapp1.1
dotnet new --help
```

例如，如果您想要创建一个控制台项目，您需要执行以下操作：

1. 使用以下命令创建一个目录，用于在其中创建您的项目：mkdir *example*
2. 使用以下命令导航至该目录：cd *example*
3. 输入以下命令：dotnet new console -o myproject

此操作将在您的 *example* 目录中创建以下文件：

- Program.cs，即您在其中写入 Lambda 函数代码的文件。
- MyProject.csproj，列出构成您的 .NET 应用程序的文件和依赖项的 XML 文件。

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::*
```

命令中结尾的 ::* 标志着安装最新版本。一旦安装完成，Lambda 模板将作为 dotnet new 的一部分显示。要验证这一点，请再次运行以下命令：

```
dotnet new -all
```

您现在应该看到以下内容：

```
dotnet new -all
Usage: new [options]
```

Options:

```

-h, --help           Displays help for this command.
-l, --list            Lists templates containing the specified name. If no name is
specified, lists all templates.
-n, --name            The name for the output being created. If no name is specified, the
name of the current directory is used.
-o, --output          Location to place the generated output.
-i, --install          Installs a source or a template pack.
-u, --uninstall        Uninstalls a source or a template pack.
--type                Filters templates based on available types. Predefined values are
"project", "item" or "other".
--force               Forces content to be generated even if it would change existing
files.
-lang, --language      Specifies the language of the template to create.

```

Templates		Short Name	
Language	Tags		
Lambda Detect Image Labels	AWS/Lambda/Function	lambda.DetectImageLabels	[C#]
Lambda Empty Function	AWS/Lambda/Function	lambda.EmptyFunction	[C#]
Lex Book Trip Sample	AWS/Lambda/Function	lambda.LexBookTripSample	[C#]
Lambda Simple DynamoDB Function	AWS/Lambda/Function	lambda.DynamoDB	[C#]
Lambda Simple Kinesis Firehose Function	AWS/Lambda/Function	lambda.KinesisFirehose	[C#]
Lambda Simple Kinesis Function	AWS/Lambda/Function	lambda.Kinesis	[C#]
Lambda Simple S3 Function	AWS/Lambda/Function	lambda.S3	[C#]
Lambda ASP.NET Core Web API	AWS/Lambda/Serverless	lambda.AspNetCoreWebAPI	[C#]
Lambda DynamoDB Blog API	AWS/Lambda/Serverless	lambda.DynamoDBBlogAPI	[C#]
Lambda Empty Serverless	AWS/Lambda/Serverless	lambda.EmptyServerless	[C#]
Step Functions Hello World	AWS/Lambda/Serverless	lambda.StepFunctionsHelloWorld	[C#]
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		
ASP.NET Core Empty		web	[C#],
F#	Web/Empty		
ASP.NET Core Web App (Model-View-Controller)		mvc	[C#],
F#	Web/MVC		
ASP.NET Core Web App		razor	[C#],
	Web/MVC/Razor Pages		
ASP.NET Core with Angular		angular	[C#],
	Web/MVC/SPA		
ASP.NET Core with React.js		react	[C#],
	Web/MVC/SPA		
ASP.NET Core with React.js and Redux		reactredux	[C#],
	Web/MVC/SPA		
ASP.NET Core Web API		webapi	[C#],
F#	Web/WebAPI		
global.json file		globaljson	
Config			

NuGet Config	nugetconfig
Config	
Web Config	webconfig
Config	
Solution File	sln
Solution	
Razor Page	page
Web/ASP.NET	
MVC ViewImports	viewimports
Web/ASP.NET	
MVC ViewStart	viewstart
Web/ASP.NET	

Examples:

```
dotnet new mvc --auth Individual
dotnet new console
dotnet new --help
```

要检查有关特定模板的详细信息，请使用以下命令：
`dotnet new lambda.EmptyFunction --help`

请注意以下几点：

```
-p|--profile  The AWS credentials profile set in aws-lambda-tools-defaults.json and used
as the default profile when interacting with AWS.
string - Optional

-r|--region   The AWS region set in aws-lambda-tools-defaults.json and used as the default
region when interacting with AWS.
string - Optional
```

当您创建 Lambda 函数时，您可以设置这些可选值，然后它们将自动写入 `aws-lambda-tools-defaults.json` 文件，该文件是作为函数创建过程的一部分构建的。下面介绍了其含义：

- `--profile`：您的执行角色。

创建 IAM 角色（执行角色）：

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 按照 IAM 用户指南 中 [IAM 角色](#) 的步骤操作，创建 IAM 角色（执行角色）。遵循步骤创建角色时，请注意以下事项：
 - 在 Select Role Type 中，选择 AWS Service Roles，然后选择 AWS Lambda。
 - 在 Attach Policy 中，选择最符合您的 Lambda 函数要求的策略。如果未与任何其他 AWS 服务交互，您将需要选择 `AWSLambdaBasicExecutionRole`。但是，假设您的 Lambda 函数与 Kinesis 交互，则您需要选择 `AWSLambdaKinesisExecutionRole`。

- `--region`：您的函数将驻留的 Amazon 地区。

例如，要创建 Lambda 函数，请运行以下命令，并将 `--region` 参数的值替换为您选择的地区，将 `--profile` 替换为您的 IAM 配置文件：

```
dotnet new lambda.EmptyFunction --name MyFunction --iam-profile default --region region
```

此过程应创建类似于下面的目录结构：

```
<dir>myfunction
    /src/myfunction
    /test/myfunction
```

在 src/myfunction 目录下，检查以下文件：

- aws-lambda-tools-default.json：这是您部署 Lambda 函数时指定命令行选项的位置。例如：

```
"profile": "iam_profile",
  "region" : "region",
  "configuration" : "Release",
  "framework" : "netcoreapp2.0",
  "function-runtime": "dotnetcore2.0",
  "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 \(p. 85\)](#)。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。请特别注意，它包含 Amazon.Lambda.Tools 程序包，提供前述 Lambda 模板的扩展。

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Amazon.Lambda.Core" Version="2.0.0" />
    <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="1.1.0" />
  </ItemGroup>

  <ItemGroup>
    <DotNetCliToolReference Include="Amazon.Lambda.Tools" Version="2.0.0" />
  </ItemGroup>

</Project>
```

- Readme：使用此文件记录您的 Lambda 函数。

在 myfunction/test 目录， 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="2.0.0" />
  ...

```

- 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);
        }
    }
}
```

您的函数一旦通过其测试，您就可以通过从父 example 目录运行以下命令对其进行构建和部署：

```
dotnet restore
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
```

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 Json.NET。Lambda 将使用 JSON.NET 作为 NuGet 程序包，从而提供 JSON 串行器的实现实例。
- 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="2.0.0" />
  <PackageReference Include="Amazon.Lambda.Serialization.Json" Version="1.0.1" />
</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 \(p. 69\)](#)。

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。该插件作为 Nuget 程序包的一部分提供。

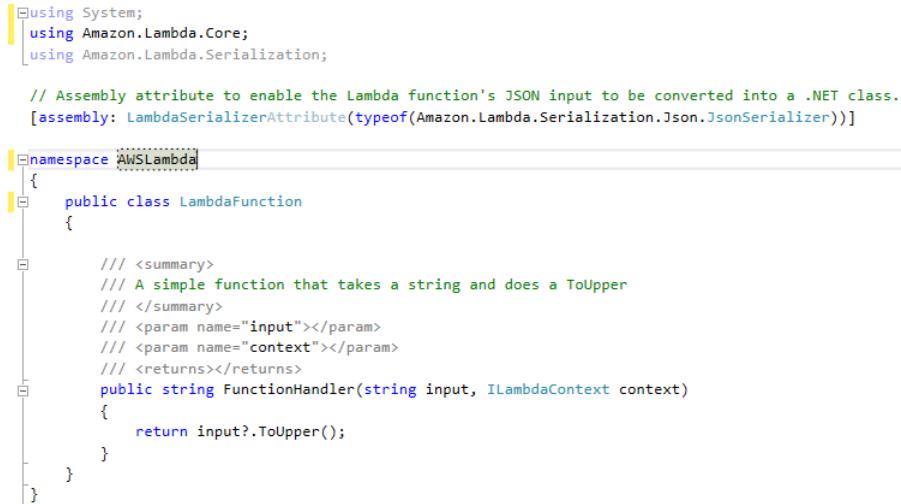
步骤 1：创建并构建项目

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 内核)，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，请选择空白函数，然后选择完成。
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：您执行 Lambda 函数时所需的 IAM 角色。如果您尚未创建执行角色，请执行以下操作：
 1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
 2. 按照 IAM 用户指南 中 [创建角色以向 AWS 服务委派权限](#) 的步骤创建 IAM 角色（执行角色）。遵循步骤创建角色时，请注意以下事项：
 - 在 Role Name 中，使用在 AWS 账户内唯一的名称。
 - 在 Select Role Type 中，选择 AWS Service Roles，然后选择授予该服务权限以担任此角色的服务角色。
 - 在 Attach Policy 中，选择适合执行您的 Lambda 函数的权限策略。
 - function-handler：这是指定函数 `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": "iam-execution-profile",  
    "region" : "region",  
    "configuration" : "Release",  
    "framework" : "netcoreapp2.0",  
    "function-runtime": "dotnetcore2.0",  
    "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.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项目节点，然后选择发布至 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. 在高级函数详细信息窗口中，执行以下操作：
- 指定 Role Name：，前面所述的 IAM 角色。
 - （可选）在环境：中指定您要使用的任意环境变量。有关更多信息，请参阅 [环境变量 \(p. 354\)](#)。

- (可选) 指定内存 (MB) 或超时(秒)配置。
- (可选) 如果您的 Lambda 函数需要访问在 VPC 内部运行的资源，请指定 VPC 配置。有关更多信息，请参阅 [配置 Lambda 函数以访问 Amazon VPC 中的资源 \(p. 129\)](#)。
- 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 内核 CLI 部署 AWS Lambda 项目。

创建部署程序包 (Go)

要创建 Lambda 函数，首先需要创建 Lambda 函数部署程序包（包含代码和所有依赖项的 .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。

有关在 Go 中编写的 Lambda 函数，请下载适用于 Go 的 Lambda 库，方法是导航到 Go 运行时目录并输入以下命令：

```
go get github.com/aws/aws-lambda-go
```

然后通过 CLI 使用以下命令构建、打包和部署 Go Lambda 函数。请注意，您的 *function-name* 必须与您的 *Lambda handler* 名称匹配。

```
GOOS=linux go build lambda_handler.go
zip handler.zip ./lambda_handler
# --handler is the path to the executable inside the .zip
aws lambda create-function \
--region region \
--function-name lambda-handler \
--memory 128 \
--role arn:aws:iam::account-id:role/execution_role \
--runtime go1.x \
--zip-file fileb://path-to-your-zip-file/handler.zip \
--handler lambda-handler
```

Note

如果您使用的是非 Linux 环境（如 Windows 或 macOS），请确保您的处理程序函数与 Lambda 执行上下文兼容，方法是在编译您的处理程序函数代码时将 GOOS（Go 操作系统）环境变量设置为“linux”。

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 -o main.zip main
```

在 Powershell 中，运行以下命令：

```
$env:GOOS = "linux"
go build -o main main.go
~\Go\Bin\build-lambda-zip.exe -o main.zip main
```

创建部署程序包 (Java)

部署程序包可以是 .zip 文件或独立的 jar；具体如何选择由您决定。您可以使用自己熟悉的任何构建和打包工具来创建部署程序包。

我们提供了使用 Maven 创建独立 jar 和使用 Gradle 创建 .zip 文件的示例。有关更多信息，请参阅以下主题：

主题

- [使用 Maven 但不借助任何 IDE 创建 .jar 部署程序包 \(Java\) \(p. 88\)](#)
- [使用 Maven 和 Eclipse IDE 创建 .jar 部署程序包 \(Java\) \(p. 90\)](#)
- [创建 .zip 部署程序包 \(Java\) \(p. 92\)](#)
- [使用 Eclipse IDE 和 AWS 软件开发工具包插件编写 Lambda 函数 \(Java\) \(p. 94\)](#)

使用 Maven 但不借助任何 IDE 创建 .jar 部署程序包 (Java)

本章节介绍如何在命令行中使用 Maven 将 Java 代码打包到部署程序包中。

主题

- [在您开始之前 \(p. 88\)](#)
- [项目结构概述 \(p. 88\)](#)
- [步骤 1：创建项目 \(p. 89\)](#)
- [步骤 2：构建项目（创建部署程序包）\(p. 90\)](#)

在您开始之前

您需要安装 Maven 命令行构建工具。有关更多信息，请转到 [Maven](#)。如果使用的是 Linux，请检查包管理器。

```
sudo apt-get install mvn
```

如果使用的是 Homebrew

```
brew install maven
```

项目结构概述

设置该项目后，您应拥有如下所示的文件夹结构：

```
project-dir/pom.xml
project-dir/src/main/java/ (your code goes here)
```

代码将位于 `/java` 文件夹中。例如，如果程序包名称为 `example`，其中有一个 `Hello.java` 类，则结构为：

```
project-dir/src/main/java/example/Hello.java
```

构建项目后，生成的 `.jar` 文件（即部署程序包）将位于 `project-dir/target` 子目录中。

步骤 1：创建项目

按照本章节中的步骤创建 Java 项目。

1. 创建项目目录 (`project-dir`)。
2. 在 `project-dir` 目录中，创建以下内容：

- 项目对象模型文件，`pom.xml`。添加以下项目信息和配置详细信息，以便 Maven 构建该项目。

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>doc-examples</groupId>
  <artifactId>lambda-java-example</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>lambda-java-example</name>

  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-lambda-java-core</artifactId>
      <version>1.1.0</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>2.3</version>
        <configuration>
          <createDependencyReducedPom>false</createDependencyReducedPom>
        </configuration>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

Note

- 在 `dependencies` 部分中，`groupId`（即 `com.amazonaws`）是 Maven Central 存储库中 Maven 项目的 Amazon AWS 组 ID。`artifactId`（即 `aws-lambda-java-core`）是

AWS Lambda 的核心库，其中提供了 RequestHandler、RequestStreamHandler 和 Context AWS Lambda 接口的定义，以供您在 Java 应用程序中使用。在构建时，Maven 会解析这些依赖项。

- 在插件部分中，Apache maven-shade-plugin 是一种插件，Maven 会在构建期间下载并使用它。该插件用于打包 jar 以创建独立的 .jar (一种 .zip 文件) - 即部署程序包。
- 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. 在 *project-dir* 中，创建下面的结构：

```
project-dir/src/main/java
```

4. 在 /java 子目录下，添加 Java 文件和文件夹结构（如果有）。例如，如果 Java 程序包名称为 example，源代码为 Hello.java，则目录结构如下所示：

```
project-dir/src/main/java/example>Hello.java
```

步骤 2：构建项目（创建部署程序包）

现在，您可以在命令行中使用 Maven 构建该项目了。

1. 在命令提示符处，将目录更换到项目目录 (*project-dir*)。
2. 运行下面的 mvn 命令来构建该项目：

```
$ mvn package
```

生成的 .jar 将保存为 *project-dir*/target/lambda-java-example-1.0-SNAPSHOT.jar。该 .jar 名称是通过将 pom.xml 文件中的 artifactId 和 version 串接起来生成的。

构建命令利用 pom.xml 中的信息进行必要的转换，最终生成了该 .jar。该独立 .jar (.zip 文件) 包含了所有的依赖项。此即为部署程序包，您可以将其上传到 AWS Lambda 以创建 Lambda 函数。

使用 Maven 和 Eclipse IDE 创建 .jar 部署程序包 (Java)

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。

主题

- [在您开始之前 \(p. 90\)](#)
- [步骤 1：创建并构建项目 \(p. 90\)](#)

在您开始之前

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 窗口中的 Work with: 框中键入 <http://download.eclipse.org/technology/m2e/releases>，然后选择 Add。
3. 按照步骤操作以完成安装。

步骤 1：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 : 1.1.0

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 : **example**
 - Name : **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 窗口的 Goals 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
 - Version : 2.3
- 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`。

创建 .zip 部署程序包 (Java)

本节提供了创建 .zip 文件作为部署程序包的示例。您可以使用任何构建和打包工具来创建该 zip。不管使用何种工具，生成的 .zip 文件都必须采用下面的结构：

- 所有编译的类文件和资源文件都必须位于根级别。
- 运行代码所必需的全部 jar 都必须位于 `/lib` 目录中。

Note

此外，您还可以构建独立的 jar（也称作压缩文件）作为部署程序包。有关使用 Maven 创建独立 .jar 的示例，请参阅[创建部署程序包 \(Java\) \(p. 88\)](#)。

以下示例使用 Gradle 构建和部署工具创建 .zip。

Important

要求安装 Gradle 2.0 版或更高版本。

在您开始之前

您需要下载 Gradle。有关说明，请转到 gradle 网站：<https://gradle.org/>。

示例 1：使用 Gradle 和 Maven Central 存储库创建 .zip

本演练结束时，项目目录 (`project-dir`) 将包含如下所示的内容结构：

```
project-dir/build.gradle
project-dir/src/main/java/
```

`/java` 文件夹包含您的代码。例如，如果程序包名称为 `example`，其中有一个 `Hello.java` 类，则结构为：

```
project-dir/src/main/java/example/Hello.java
```

构建项目后，生成的 .zip 文件（即部署程序包）将位于 `project-dir/build/distributions` 子目录中。

1. 创建项目目录 (*project-dir*)。
2. 在 *project-dir* 中，创建 build.gradle 文件并添加以下内容：

```
apply plugin: 'java'

repositories {
    mavenCentral()
}

dependencies {
    compile (
        'com.amazonaws:aws-lambda-java-core:1.1.0',
        'com.amazonaws:aws-lambda-java-events:1.1.0'
    )
}

task buildZip(type: Zip) {
    from compileJava
    from processResources
    into('lib') {
        from configurations.runtime
    }
}
build.dependsOn buildZip
```

Note

- 存储库部分指的是 Maven Central 存储库。在构建时，它从 Maven Central 获取依赖项（即：两个 AWS Lambda 库）。
- buildZip 任务描述如何创建部署程序包 .zip 文件。

例如，如果解压缩生成的 .zip 文件，可在根级别找到所有编译好的类文件和资源文件。此外，还可以找到一个包含运行代码所需的 jar 的 /lib 目录。

- 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. 在 *project-dir* 中，创建下面的结构：

```
project-dir/src/main/java/
```

4. 在 /java 子目录下，添加 Java 文件和文件夹结构（如果有）。例如，如果 Java 程序包名称是 example，源代码为 Hello.java，则目录结构如下所示：

```
project-dir/src/main/java/example/Hello.java
```

5. 运行下面的 gradle 命令以构建项目并将其打包到一个 .zip 文件中。

```
project-dir> gradle build
```

6. 验证生成的 *project-dir.zip* 文件位于 *project-dir/build/distributions* 子目录中。
7. 现在，您可以将部署程序包（即 .zip 文件）上传到 AWS Lambda，以创建 Lambda 函数并使用示例事件数据手动调用它来进行测试。有关说明，请参阅 [\(可选\) 创建用 Java 编写的 Lambda 函数 \(p. 52\)](#)。

示例 2：使用 Gradle 和本地 Jar 创建 .zip

您可以选择不使用 Maven Central 存储库，而是将所有依赖项都包含在项目文件夹中。在这种情况下，项目文件夹 (*project-dir*) 的结构如下：

```
project-dir/jars/          (all jars go here)
project-dir/build.gradle
project-dir/src/main/java/ (your code goes here)
```

因此，如果 Java 代码包含 example 程序包和 Hello.java 类，则代码将位于下面的子目录中：

```
project-dir/src/main/java/example>Hello.java
```

您的 build.gradle 文件应为：

```
apply plugin: 'java'

dependencies {
    compile fileTree(dir: 'jars', include: '*.jar')
}

task buildZip(type: Zip) {
    from compileJava
    from processResources
    into('lib') {
        from configurations.runtime
    }
}

build.dependsOn buildZip
```

注意：依赖项指定 fileTree，后者将 project-dir/jars 标识为包含所有必需 jar 的子目录。

现在，您可以构建程序包了。运行下面的 gradle 命令以构建项目并将其打包到一个 .zip 文件中。

```
project-dir> gradle build
```

使用 Eclipse IDE 和 AWS 软件开发工具包插件编写 Lambda 函数 (Java)

AWS SDK Eclipse Toolkit 提供了一个 Eclipse 插件，使用它既可以创建也可以上传部署程序包，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](#)。

创建部署程序包 (Python)

要创建 Lambda 函数，首先需要创建 Lambda 函数部署程序包（包含代码和所有依赖项的 .zip 文件）。

您可自行创建部署程序包或直接在 Lambda 控制台中编写代码，在后一种情况下，控制台将为您创建并上传部署程序包，从而创建您的 Lambda 函数。请记下以下内容来确定您是否可使用该控制台创建 Lambda 函数：

- 简单场景 - 如果自定义代码只需要 AWS 软件开发工具包库，则可以使用 AWS Lambda 控制台中的内联编辑器。使用控制台可以编辑代码并将代码上传到 AWS Lambda。控制台会将代码及相关的配置信息压缩到 Lambda 服务能够运行的部署程序包中。

您还可以在控制台中测试代码（使用示例事件数据手动调用代码）。

Note

Lambda 服务预装了适用于 Python 的 AWS 开发工具包。

- 高级场景 - 如果编写的代码需要用到其他资源（如使用图形库进行图像处理），或需要使用 AWS CLI 代替控制台，则需要先创建 Lambda 函数部署程序包，然后再使用控制台或 CLI 上传部署程序包。

Note

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。

以下是创建部署程序包的示例过程（在控制台外）。

Note

此过程应适合 Python 的大多数标准安装，并且在 Lambda 函数中使用纯 Python 模块的情况下也适合 pip。如果您要包含一些模块（这些模块具有本机依赖项或具有随 Homebrew 一起安装到 OS X 上的 Python），则应参阅下一部分，该部分提供了在使用 Virtualenv 时创建部署程序包的说明。有关更多信息，请参阅[使用通过 Virtualenv 创建的 Python 环境创建部署程序包 \(p. 95\)](#)和[Virtualenv](#) 网站。

您将使用 pip 安装依赖项/库。有关安装 pip 的信息，请转到[安装](#)。

1. 创建一个目录，例如 project-dir。
2. 将所有 Python 源文件 (.py 文件) 保存在此目录的根级。
3. 使用 pip 安装所有库。同样，在该目录的根级安装这些库。

```
pip install module-name -t /path/to/project-dir
```

例如，以下命令会将 requests HTTP 库安装在 project-dir 目录中。

```
pip install requests -t /path/to/project-dir
```

如果使用的是 Mac OS X，并且您通过 Homebrew（请参阅[Homebrew](#)）安装了 Python，那么前述命令不适用。一个简单的变通方法是使用以下内容将在您的 setup.cfg 中添加 /path/to/project-dir 文件。

```
[install]
prefix=
```

4. 压缩 project-dir 目录（您的部署程序包）的内容。

Important

压缩目录的内容，而不是目录本身。Zip 文件内容可用作 Lambda 函数的当前工作目录。例如：/project-dir/codefile.py/lib/yourlibraries

Note

AWS Lambda 包含适用于 Python 的 AWS 软件开发工具包 (Boto 3)，因此您无需将其包含在您的部署程序包中。但是，如果您要使用默认包含的 Boto3 版本之外的版本，则可以将其包含在部署程序包中。

使用通过 Virtualenv 创建的 Python 环境创建部署程序包

本部分介绍了您使用通过 Virtualenv 工具创建的 Python 环境时如何创建部署程序包。考虑以下示例：

- 使用 Virtualenv 工具创建了以下隔离的 Python 环境并激活了该环境：

```
virtualenv /path/to/my/virtual-env
```

您可以按如下方式在 Windows、OS X 和 Linux 中激活该环境：

- 在 Windows 中，使用 `activate.bat` 进行激活：

```
path\to\my\virtual-env\Scripts\activate.bat
```

- 在 OS X 和 Linux 中，寻求 `activate` 脚本的来源：

```
source path/to/my/virtual-env/bin/activate
```

- 此外，要在已激活的环境中安装请求程序包，请执行以下操作：

```
pip install requests
```

现在，要创建部署程序包，可执行以下操作：

1. 首先，使用要上传到 AWS Lambda 的 Python 代码创建 `.zip` 文件。
2. 将库从前面的已激活虚拟环境添加到 `.zip` 文件。也就是说，将以下目录的内容添加到 `.zip` 文件（再次注意，应添加目录的内容而不是目录本身）。

对于 Windows，该目录为：

```
%VIRTUAL_ENV%\Lib\site-packages
```

对于 OS X、Linux，该目录为：

```
$VIRTUAL_ENV/lib/python3.6/site-packages
```

Note

如果您在您的虚拟环境的 `site-packages` 目录中找不到该程序包，则可能会在 `dist-packages` 目录中找到它。

有关创建 Python 部署程序包的示例，请参阅 [Python \(p. 175\)](#)。

使用 SAM Local 在本地测试您的无服务应用程序 (公开测试版)

Note

此功能在公开测试版中提供，可能随时更改。

AWS SAM 是用于部署无服务应用程序的快速简单的方式，您可以用它来编写简单的模板，描述您的函数及其事件源 (Amazon API Gateway、Amazon S3、Kinesis 等)。SAM Local 是以 AWS SAM 为基础的 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。无论您使用 Linux、Mac 或 Microsoft Windows 进行开发，均可使用 SAM Local 创建模拟 AWS 运行时环境的本地测试环境。这样可以帮助您解决性能之类的问题。使用 SAM Local 还可以更快地迭代开发您的 Lambda 函数代码，因为无需将您的应用程序包重新部署到 AWS Lambda 运行时。有关更多信息，请参阅 [使用 SAM Local 构建一个简单的应用程序 \(p. 99\)](#)。

SAM Local 与 **AWS SAM** 配合使用，可以直接调用或通过 API 网关 终端节点调用利用 SAM 模板定义的函数。您可以使用 SAM Local 功能在您自己的测试环境中分析无服务应用程序的性能并进行相应更新。以下示例利用操作代码样本介绍了使用 SAM Local 的其他好处。例如，您可以执行以下操作：

- 生成示例函数负载 (例如 Amazon S3 事件)。

```
$ sam local generate-event s3 --bucket bucket-name --key key-name  
> event_file.json
```

- 利用您的 Lambda 函数对示例函数负载进行本地测试。

```
$ sam local invoke function-name -e event_file.json
```

- 引发本地 API 网关以测试 HTTP 请求和响应功能。可以使用热重载功能测试和迭代您的函数，而无需重启或将它们重新加载到 AWS 运行时。

```
$ sam local start-api
```

SAM Local 将在您的 SAM 模板内自动查找任何定义了 API 事件源的函数，并将其安装在定义的 HTTP 路径中。在以下示例中，Ratings 函数将在 /ratings 中为 GET 请求安装 ratings.py:handler()。

```
Ratings:  
  Type: AWS::Serverless::Function  
  Properties:  
    Handler: ratings.handler  
    Runtime: python3.6  
    Events:  
      Api:  
        Type: Api  
        Properties:  
          Path: /ratings  
          Method: get
```

默认情况下，SAM Local 使用[代理集成](#)并期望来自 Lambda 函数的响应包含以下各项中的一个或多个：statusCode、headers 和/或 body。例如：

```
// Example of a Proxy Integration response  
exports.handler = (event, context, callback) => {  
  callback(null, {  
    statusCode: 200,  
    headers: { "x-custom-header" : "my custom header value" },  
    body: "hello world"  
  });  
}
```

如果您的 Lambda 函数不返回有效的[代理集成](#)响应，您将会在访问您的函数时收到 HTTP 500 (Internal Server Error) 响应。SAM Local 还会打印以下错误日志消息来帮助您诊断问题：

```
ERROR: Function ExampleFunction returned an invalid response (must include one of: body,  
headers  
or statusCode in the response object)
```

- 验证是否遵循所有运行时约束，例如可使用的最大内存或 Lambda 函数调用的超时限制。
- 检查 AWS Lambda 运行时日志，以及您在 Lambda 函数代码中指定的所有自定义日志记录输出 (例如 console.log)。SAM Local 会自动显示此输出。下面是一个示例。

```
START RequestId: 2137da9a-c79c-1d43-5716-406b4e6b5c0a Version: $LATEST  
2017-05-18T13:18:57.852Z 2137da9a-c79c-1d43-5716-406b4e6b5c0a  
Error: any error information  
END RequestId: 2137da9a-c79c-1d43-5716-406b4e6b5c0a  
REPORT RequestId: 2137da9a-c79c-1d43-5716-406b4e6b5c0a
```

Duration: 12.78 ms Billed Duration: 100 ms Memory Size: 128 MB
Max Memory Used: 29 MB

- 使用 AWS CLI 遵循您建立的安全凭证。这就意味着您的 Lambda 函数可以对组成无服务应用程序的 AWS 服务进行远程调用。如果您尚未安装 AWS CLI，请参阅[安装 AWS 命令行界面](#)。

SAM Local 与 AWS CLI 和开发工具包一样，会按照以下顺序查找凭证：

- 环境变量 (`AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY`)
- AWS 凭证文件，在 Linux、MacOS 或 Unix 上位于 `~/.aws/credentials`，在 Windows 上位于 `c:\Users\USERNAME\.aws\credentials`。
- 实例配置文件凭证 (如果在分配了实例角色的 Amazon EC2 实例上运行)

支持的运行时

SAM Local 支持以下 AWS 运行时：

- node.js 4.3
- node.js 6.10
- python 2.7
- python 3.6
- java8
- go 1.x

如果您尚未安装 SAM Local，请参阅[安装 SAM Local \(p. 6\)](#)。

SAM Local 使用入门

SAM Local 包含以下 CLI 操作：

- start-api：创建本地 HTTP 服务器，用于托管您的所有 Lambda 函数。如果使用浏览器或 CLI 访问，此操作会在本地启动 Docker 容器来调用您的函数。它会读取 `AWS::Serverless::Function` 资源的 `CodeUri` 属性，在您的文件系统中找到包含 Lambda 函数代码的路径。对于 Node.js 或 Python 等解释性语言，此路径可以是项目的根目录；此路径还可以是存储编译构件的构建目录；对于 Java 可以是 `.jar` 文件。

如果使用解释性语言，本地更改在同一 Docker 容器中可用。这种方法意味着无需重新部署即可重新调用您的 Lambda 函数。对于编译语言或需要复杂包装支持的项目，建议运行您自己的构建解决方案，并使 AWS SAM 指向包含所需的构建依赖关系文件的目录。

- invoke：调用一次本地 Lambda 函数，调用完成后即终止。

```
# Invoking function with event file
$ sam local invoke "Ratings" -e event.json

# Invoking function with event via stdin
$ echo '{"message": "Hey, are you there?" }' | sam local invoke "Ratings"

# For more options
$ sam local invoke --help
```

- generate-event：生成模拟无服务事件。您可以使用这些操作在本地开发并测试响应异步事件的函数，例如 Amazon S3、Kinesis 和 DynamoDB 中的函数。以下为 `generate-event` 操作可用的命令选项。

```
sam local generate-event
NAME:
```

```
    sam local generate-event - Generates Lambda events (e.g. for S3/Kinesis etc) that can
    be piped to 'sam local invoke'

    USAGE:
        sam local generate-event command [command options] [arguments...]

    COMMANDS:
        s3      Generates a sample Amazon S3 event
        sns     Generates a sample Amazon SNS event
        kinesis Generates a sample Amazon Kinesis event
        dynamodb Generates a sample Amazon DynamoDB event
        api     Generates a sample Amazon API Gateway event
        schedule Generates a sample scheduled event

    OPTIONS:
        --help, -h  show help
```

- validate : 根据官方 [AWS 无服务应用程序模型规范](#) 验证您的模板。以下是示例。

```
$ sam validate
ERROR: Resource "HelloWorld", property "Runtime": Invalid value node.
Valid values are "nodejs4.3", "nodejs6.10", "java8", "python2.7",
"python3.6"(line: 11; col: 6)

# Let's fix that error...
$ sed -i 's/node/nodejs6.10/g' template.yaml

$ sam validate
Valid!
```

- package 和 deploy : `sam package` 和 `sam deploy` 隐式调用 AWS CloudFormation 的 [package](#) 和 [deploy](#) 命令。有关 SAM 应用程序包装和部署的更多信息，请参阅[打包和部署 \(p. 286\)](#)。

以下内容演示了如何在 SAM Local 中使用 `package` 和 `deploy` 命令。

```
# Package SAM template
$ sam package --template-file sam.yaml --s3-bucket mybucket --output-template-file
packaged.yaml

# Deploy packaged SAM template
$ sam deploy --template-file ./packaged.yaml --stack-name mystack --capabilities
CAPABILITY_IAM
```

使用 SAM Local 构建一个简单的应用程序

假设您希望构建一个简单的 RESTful API 操作，来创建、读取、更新和删除一组产品。开始需要创建以下目录结构：

```
dir/products.js
dir/template.yaml
```

`template.yaml` 文件是 AWS SAM 模板，描述可以处理所有 API 请求的单个 Lambda 函数。

Note

默认情况下，`start-api` 与 `invoke` 命令会在您的工作目录中搜索 `template.yaml` 文件。如果您引用其他目录中的 `template.yaml` 文件，请在这些操作中添加 `-t` 或 `--template` 参数，并传递该文件的绝对或相对路径。

复制并在 `template.yaml` 文件中粘贴以下内容。

```
AWS::TemplateFormatVersion : '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: My first serverless application.

Resources:

Products:
  Type: AWS::Serverless::Function
  Properties:
    Handler: products.handler
    Runtime: nodejs6.10
    Events:
      ListProducts:
        Type: Api
        Properties:
          Path: /products
          Method: get
      CreateProduct:
        Type: Api
        Properties:
          Path: /products
          Method: post
      Product:
        Type: Api
        Properties:
          Path: /products/{product}
          Method: any
```

上述示例配置以下 RESTful API 终端节点：

- 向 /products 发送 PUT 请求，创建新产品。
- 向 /products 发送 GET 请求，列出所有产品。
- 向 /products/{product} 发送 GET、PUT 或 DELETE 请求，读取、更新或删除一个产品。

接下来，将以下代码复制并粘贴到 products.js 文件中。

```
'use strict';

exports.handler = (event, context, callback) => {

  let id = (event.pathParameters || {}).product || false;
  switch(event.httpMethod){

    case "GET":

      if(id) {
        callback(null, {body: "This is a READ operation on product ID " + id});
        return;
      }

      callback(null, {body: "This is a LIST operation, return all products"});
      break;

    case "POST":
      callback(null, {body: "This is a CREATE operation"});
      break;

    case "PUT":
      callback(null, {body: "This is an UPDATE operation on product ID " + id});
      break;

    case "DELETE":
```

```
        callback(null, {body:"This is a DELETE operation on product ID " + id});
        break;

    default:
        // Send HTTP 501: Not Implemented
        console.log("Error: unsupported HTTP method (" + event.httpMethod + ")");
        callback(null, { statusCode: 501 })

    }
}
```

调用 `start-api` 命令，启动您的 API 操作的本地复本。

```
$ sam local start-api

2017/05/18 14:03:01 Successfully parsed template.yaml (AWS::Serverless-2016-10-31)
2017/05/18 14:03:01 Found 1 AWS::Serverless::Function
2017/05/18 14:03:01 Mounting products.handler (nodejs6.10) at /products [POST]
2017/05/18 14:03:01 Mounting products.handler (nodejs6.10) at /products/{product} [OPTIONS
  GET HEAD POST PUT DELETE TRACE CONNECT]
2017/05/18 14:03:01 Mounting products.handler (nodejs6.10) at /products [GET]
2017/05/18 14:03:01 Listening on http://localhost:3000

You can now browse to the above endpoints to invoke your functions.
You do not need to restart/reload while working on your functions,
changes will be reflected instantly/automatically. You only need to restart
if you update your AWS SAM template.
```

然后您可以使用浏览器或 CLI，在本地测试 API 终端节点。

```
$ curl http://localhost:3000/products
"This is a LIST operation, return all products"

$ curl -XDELETE http://localhost:3000/products/1
"This is a DELETE operation on product ID 1"
```

要查看更多示例，请参阅 [aws sam local/samples](#)。

本地日志

您可以使用 `invoke` 和 `start-api` 命令，将日志从 Lambda 函数的调用传递到文件中。如果要针对 SAM Local 运行自动化测试，并希望捕获日志以进行分析，这种方法就很有用。以下是示例。

```
$ sam local invoke --log-file ./output.log
```

使用环境变量文件

如果您的 Lambda 函数使用 [环境变量 \(p. 354\)](#)，SAM Local 可为 `invoke` 和 `start-api` 命令提供 `--env-vars` 参数。有了这个参数，您可以使用 JSON 文件，其中包含函数中定义的环境变量的值。JSON 文件的结构应与以下内容类似。

```
{
    "MyFunction1": {
        "TABLE_NAME": "localtable",
        "BUCKET_NAME": "testBucket"
    },
    "MyFunction2": {
        "TABLE_NAME": "localtable",
```

```
        "STAGE": "dev"
    },
}
```

然后您可以使用以下命令访问 JSON 文件：

```
$ sam local start-api --env-vars env.json
```

使用 Shell 环境

如果将您的 Shell 环境中定义的变量映射到您的 Lambda 函数中的变量，则会传递到 Docker 容器。函数可全局访问 Shell 变量。例如，假设您有两个函数，MyFunction1 和 MyFunction2，有一个变量名为 TABLE_NAME。在这种情况下，通过 Shell 环境提供的 TABLE_NAME 的值对于两个函数而言均可用。

以下命令针对两个函数将 TABLE_NAME 的值设为 myTable。

```
$ TABLE_NAME=mytable sam local start-api
```

Note

您可以组合使用 Shell 变量和具备环境变量的外部 JSON 文件，从而获得更高的灵活性。如果两处均定义了某一变量，外部文件的变量会覆盖 Shell 版本。以下是优先级从最高到最低的顺序：

- 环境变量文件
- Shell 环境
- SAM 模板中包含的硬编码值

使用 SAM Local 进行调试

`sam local invoke` 和 `sam local start-api` 均支持函数的本地调试。要启用 SAM Local 的调试支持，请通过命令行指定 `--debug-port` 或 `-d`。

```
# Invoke a function locally in debug mode on port 5858
$ sam local invoke -d 5858 function logical id

# Start local API Gateway in debug mode on port 5858
$ sam local start-api -d 5858
```

Note

如果使用 `sam local start-api`，本地 API 网关会公开您的所有 Lambda 函数。但是，因为只能指定一个调试端口，所以每次只能调试一个函数。

调试 Python 编写的函数

Python 与 Node.js 或 Java 不同，需要在您的 Lambda 函数代码中启用远程调试。如果您针对使用某种 Python 运行时 (2.7 或 3.6) 的函数启用调试 (使用上述 `--debug-port` 或 `-d` 选项)，SAM Local 会从您的主机通过该端口映射到 Lambda 容器。要启用远程调试，请使用 Python 程序包，例如 `remote-pdb`。

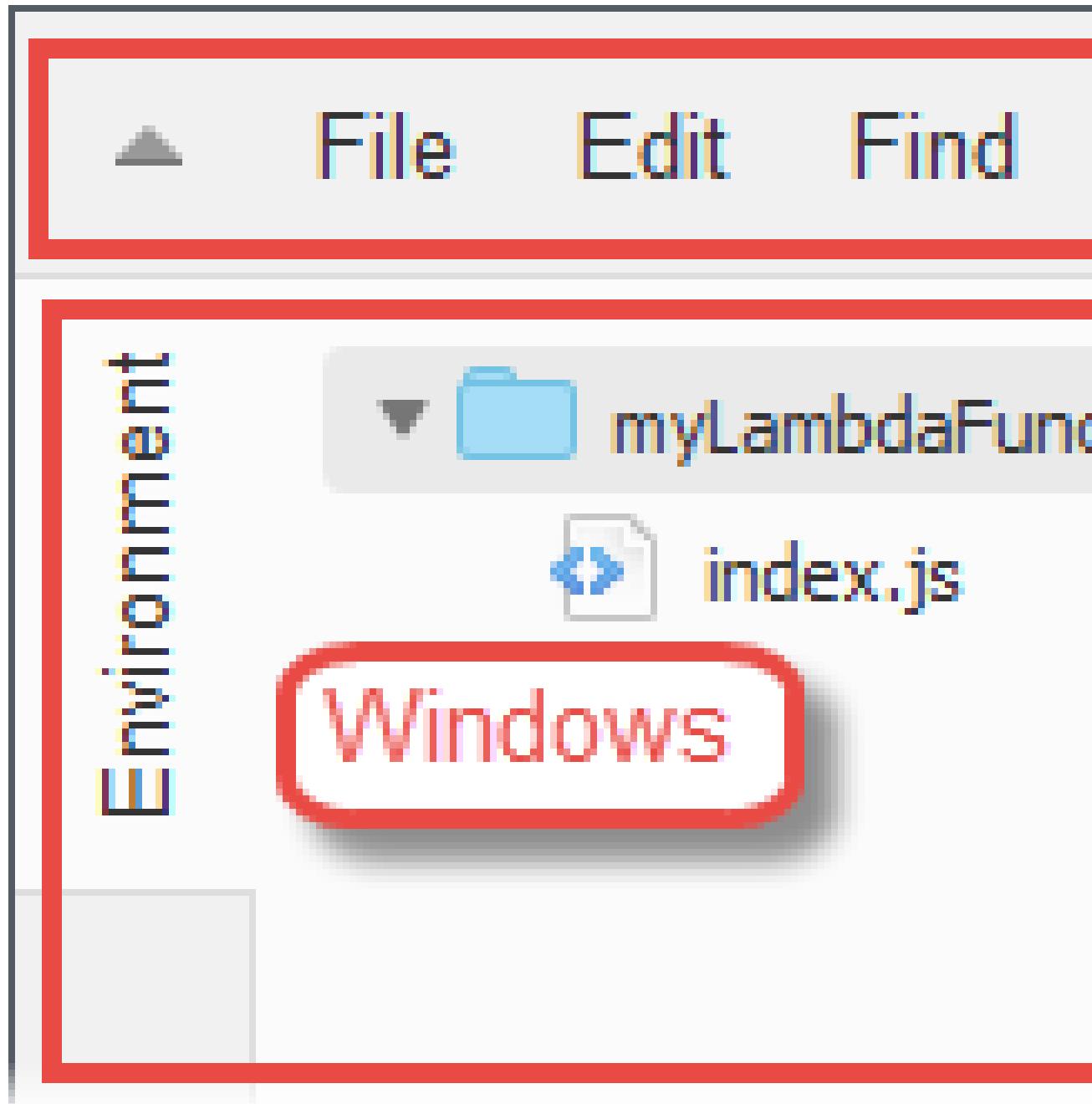
Important

配置主机时，调试程序会侦听您的代码，请确保使用 `0.0.0.0`，而不是 `127.0.0.1`。

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。



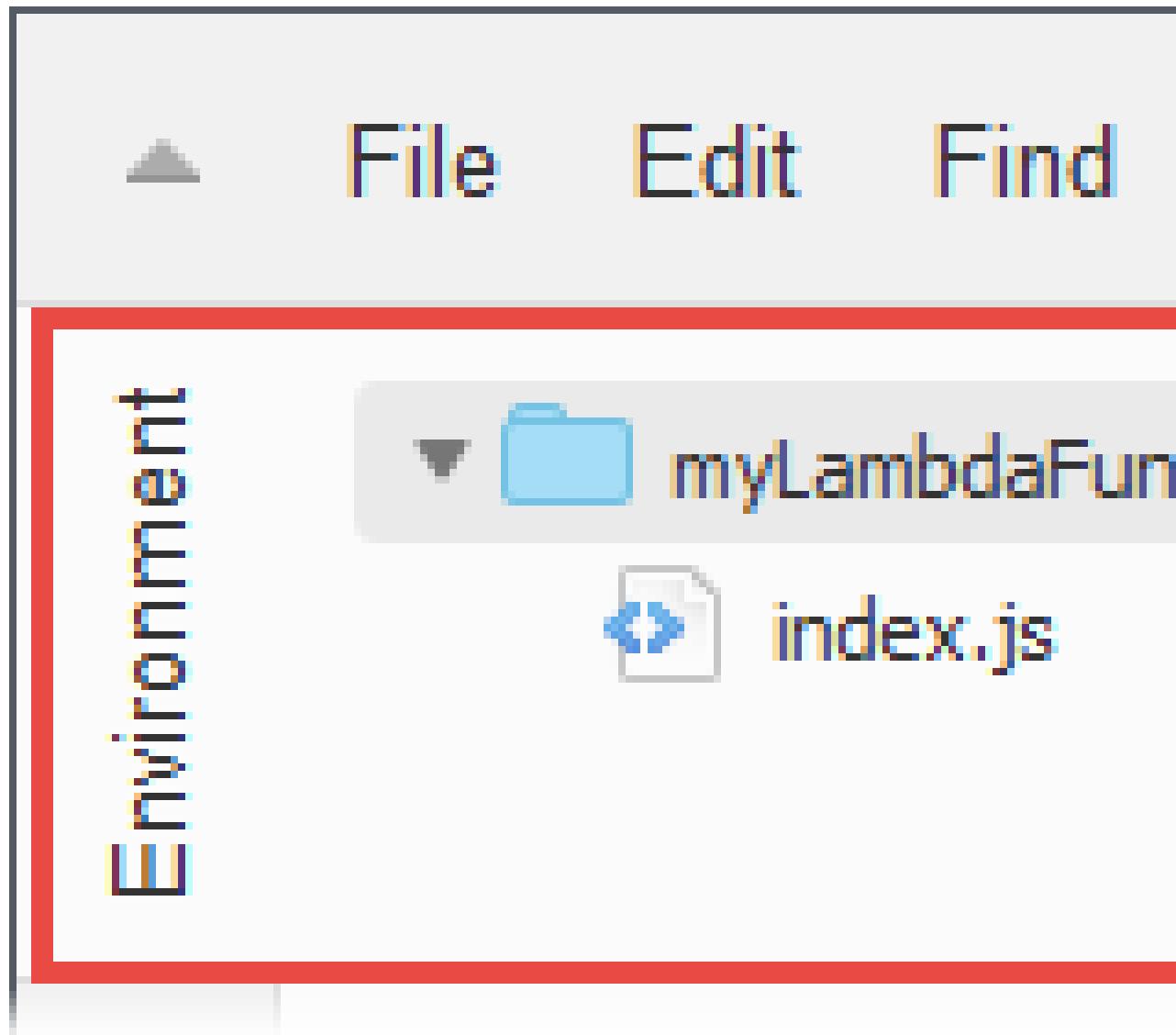
您可以使用菜单栏运行常见命令。有关更多信息，请参阅 [使用菜单栏 \(p. 121\)](#)。

您可以使用窗口处理文件、文件夹和其他命令。有关更多信息，请参阅 [处理文件和文件夹 \(p. 104\)](#) 和 [使用命令 \(p. 123\)](#)。

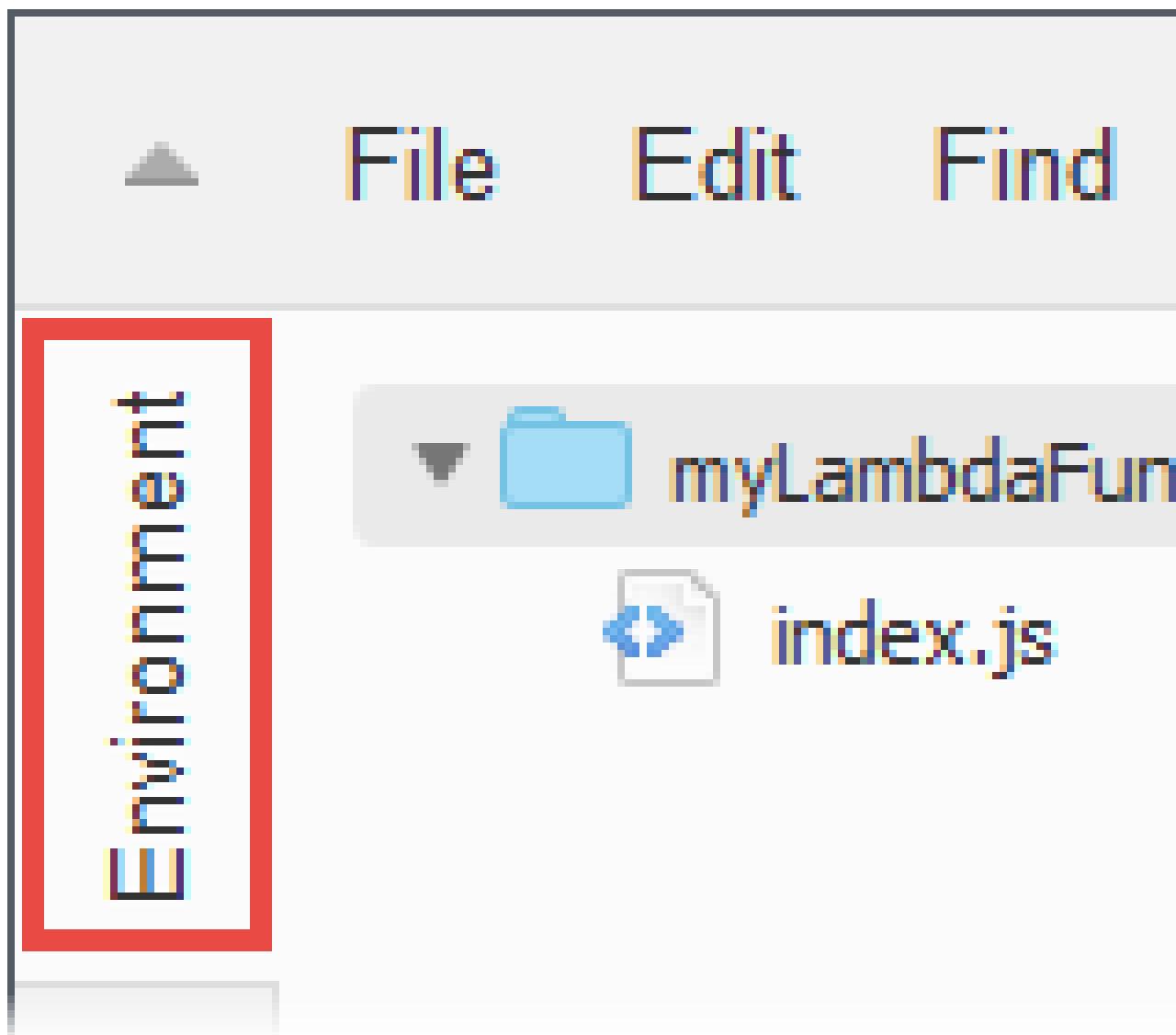
您可以使用编辑器窗格编写代码。有关更多信息，请参阅 [使用代码 \(p. 109\)](#)。

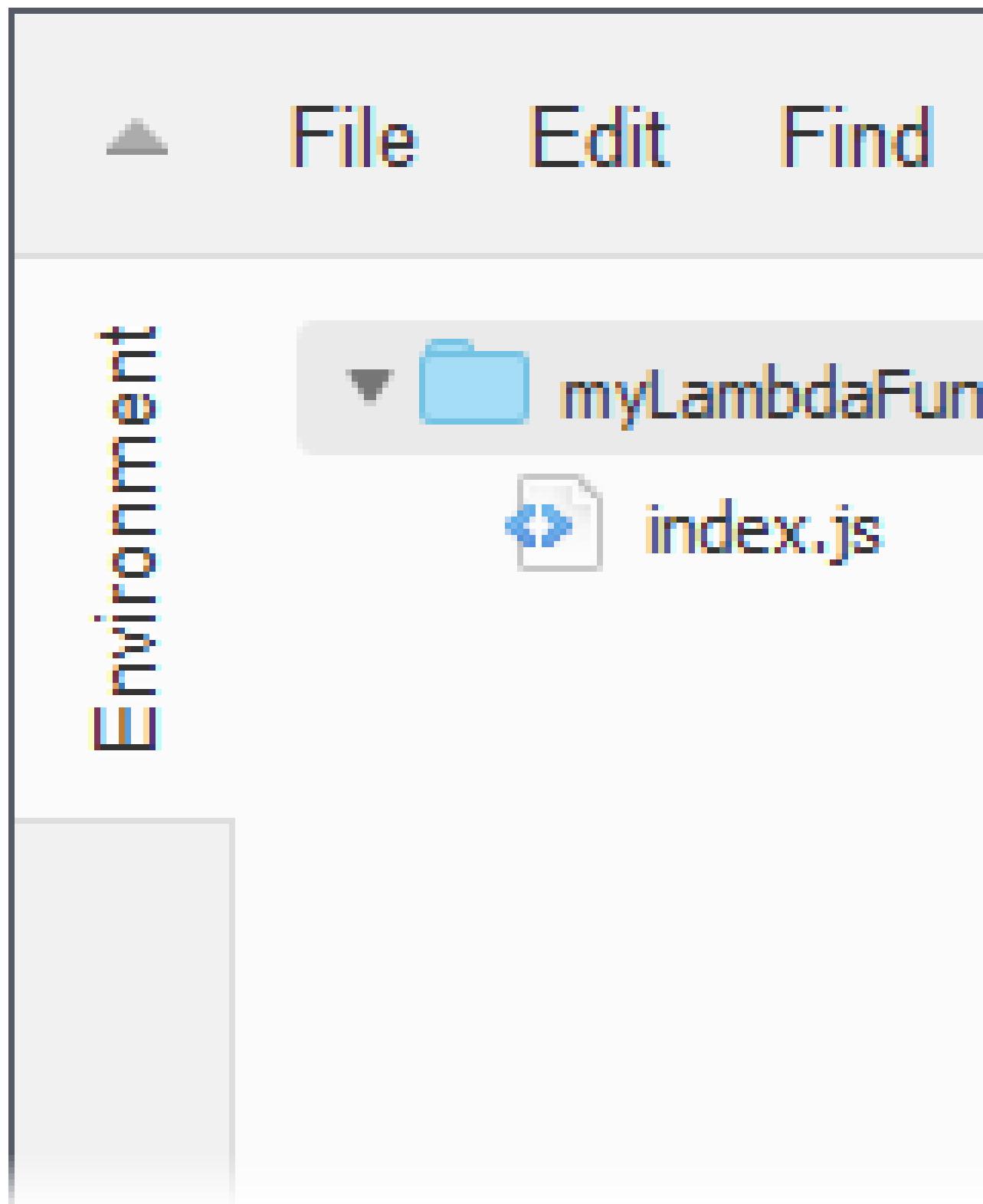
处理文件和文件夹

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。

The screenshot shows the AWS Lambda Function Editor interface. At the top, there are tabs for 'Home', 'Tools', and 'Window'. Below the tabs, there's a toolbar with icons for creating new files, opening existing files, saving, and deleting. A sidebar on the left lists files: 'index.js' (which is currently selected and highlighted in blue), 'lambda_function.py', and 'lambda_function.zip'. The main editor area displays the code for 'index.js':

```
1 exports.handler = function(event, context) {  
2     // TODO implement  
3     context.callback(  
4 };
```

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。

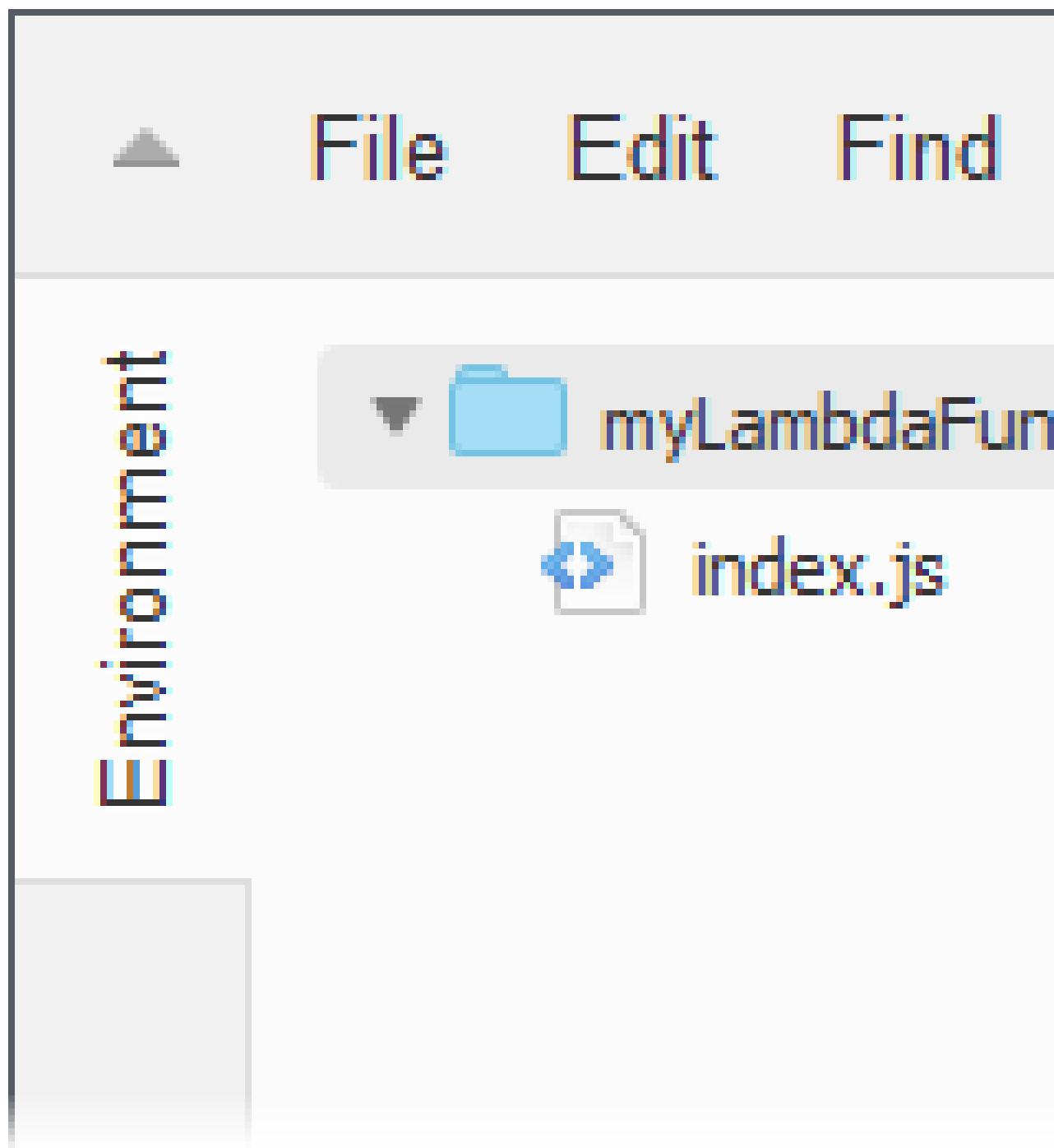
要保存某个文件，请在编辑器窗格中打开该文件并显示其内容，然后在菜单栏上选择 File、Save。

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。键入替代名称，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。

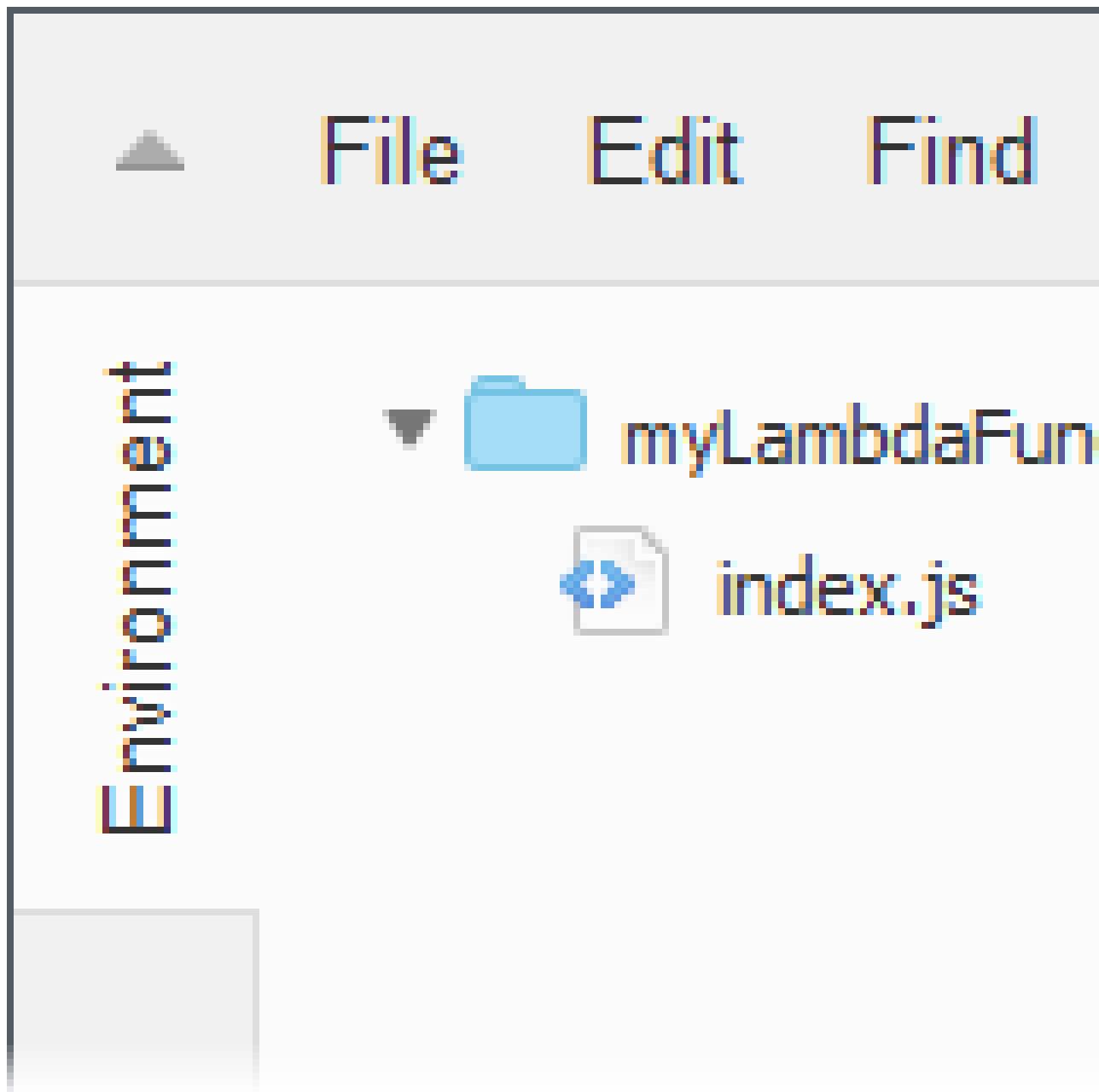


要显示已隐藏的文件，请选择 Environment 窗口中的齿轮图标，然后选择 Show Hidden Files。

您还可以使用 Commands 窗口创建、打开和管理文件。有关更多信息，请参阅 [使用命令 \(p. 123\)](#)。

使用代码

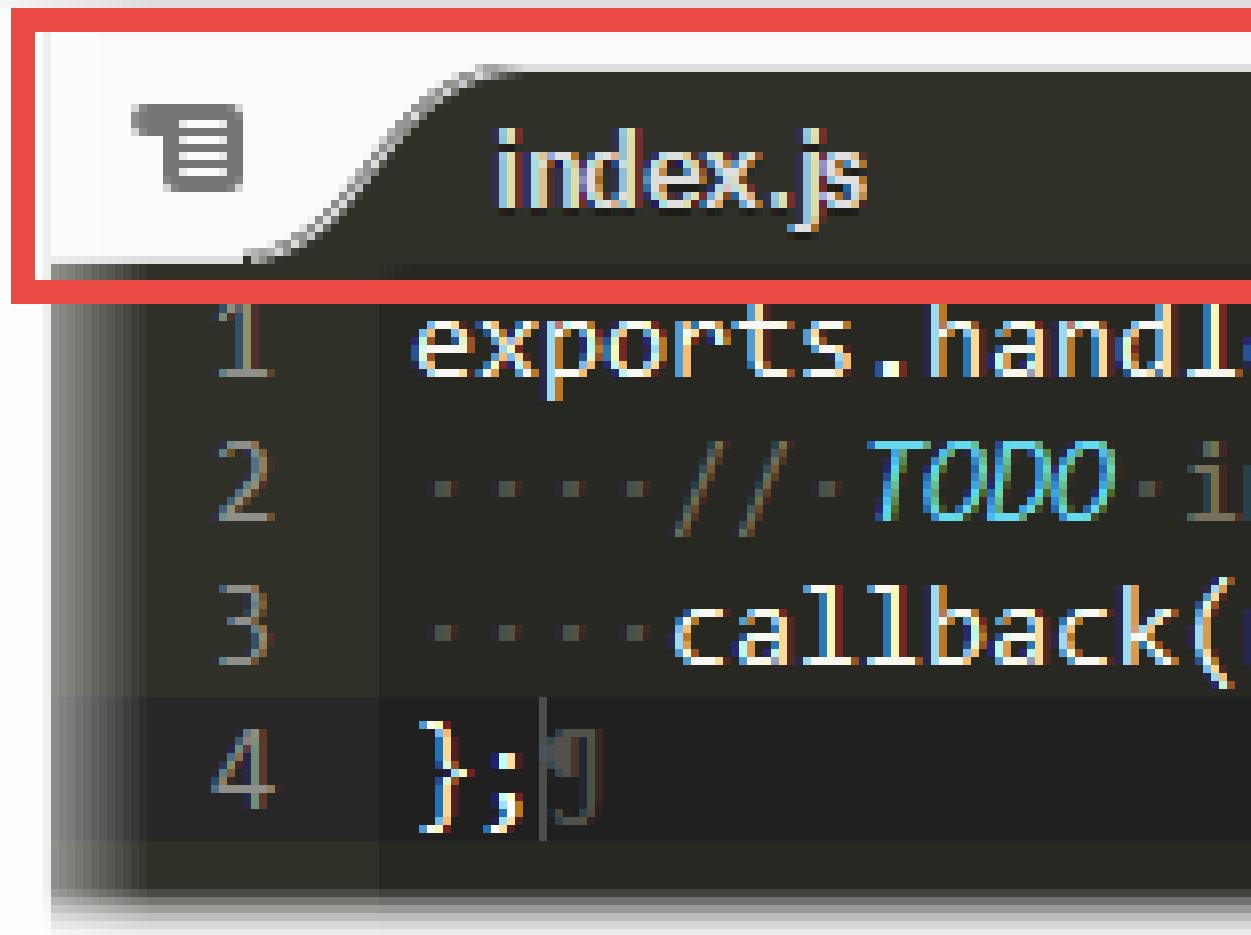
使用代码编辑器中的编辑器窗格可以查看和编写代码。



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。

oto Tools Window



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。

Goto Tools Window

1



index.js

Close Pane

Close All Tabs In All

Close All But Current

2

index.js

Split Pane in Two Rows

Split Pane in Two Col

要关闭某个已打开文件，请执行下列操作之一：

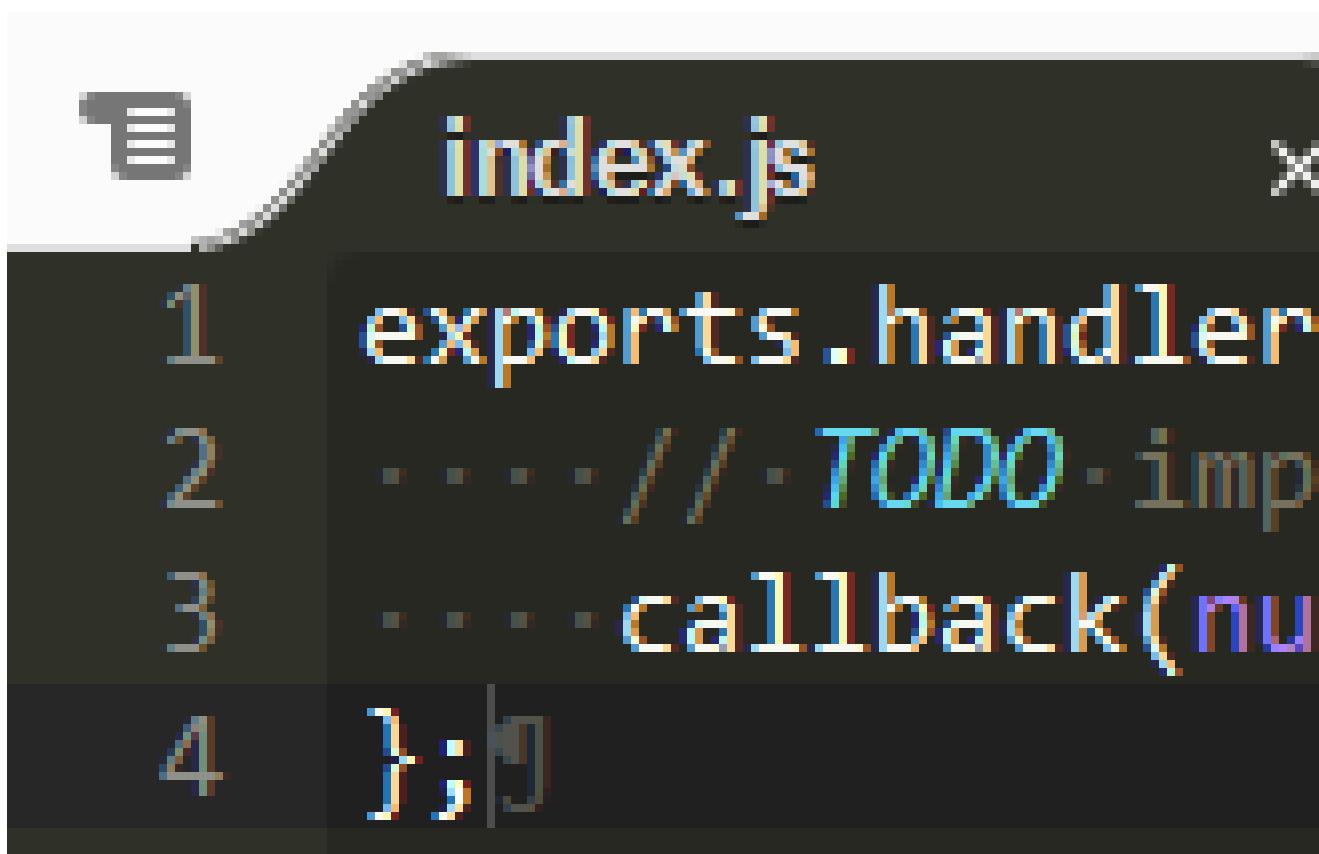
- 选择该文件的选项卡中的 X 图标。
- 选择该文件的选项卡。接下来，选择选项卡按钮栏中的下拉菜单按钮，然后选择 Close Pane。

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。



The screenshot shows a dark-themed code editor window for a Lambda function named "index.js". The code is as follows:

```
1 exports.handler
2     ... // - TODO - imp
3     ... callback(nu
4 };
```

The editor has a sidebar on the left with icons for file operations. The top right corner features a close button (X). The code area uses syntax highlighting with colors like orange, blue, and green.

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。



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。

• **JavaScript**

JSON

LESS

Lua

Perl

PHP

Python

Ruby

Scala

SCSS

SH

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。

✓ Soft Tabs (spaces)

Tab Size

2

3

4

8

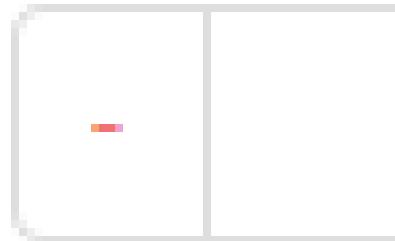
Other

-	4
---	---

要为所有文件更改是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。

- Show Invisibles
- Show Gutter
- Auto-pair Brackets, Quotations
- Wrap Lines
- ↵ to Print Margin

Font Size



使用菜单栏

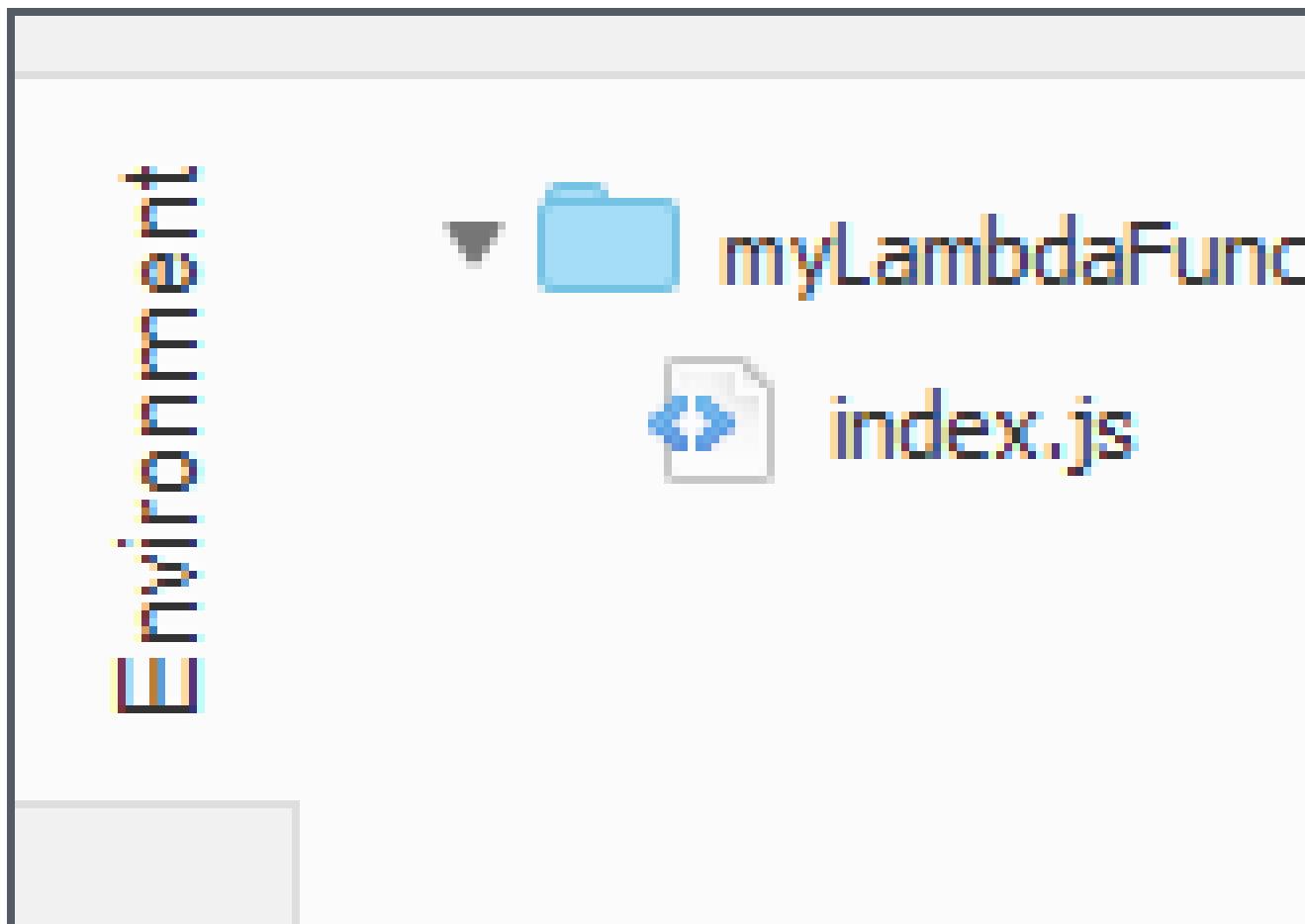
您可以使用菜单栏运行常见命令。



要隐藏菜单栏，请在菜单栏中选择上箭头。



要在菜单栏处于隐藏状态时显示菜单栏，请在菜单栏中选择下箭头。



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。

您还可以通过使用 Commands 窗口运行命令。有关更多信息，请参阅[使用命令 \(p. 123\)](#)。

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菜单：Save 和 Test。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。

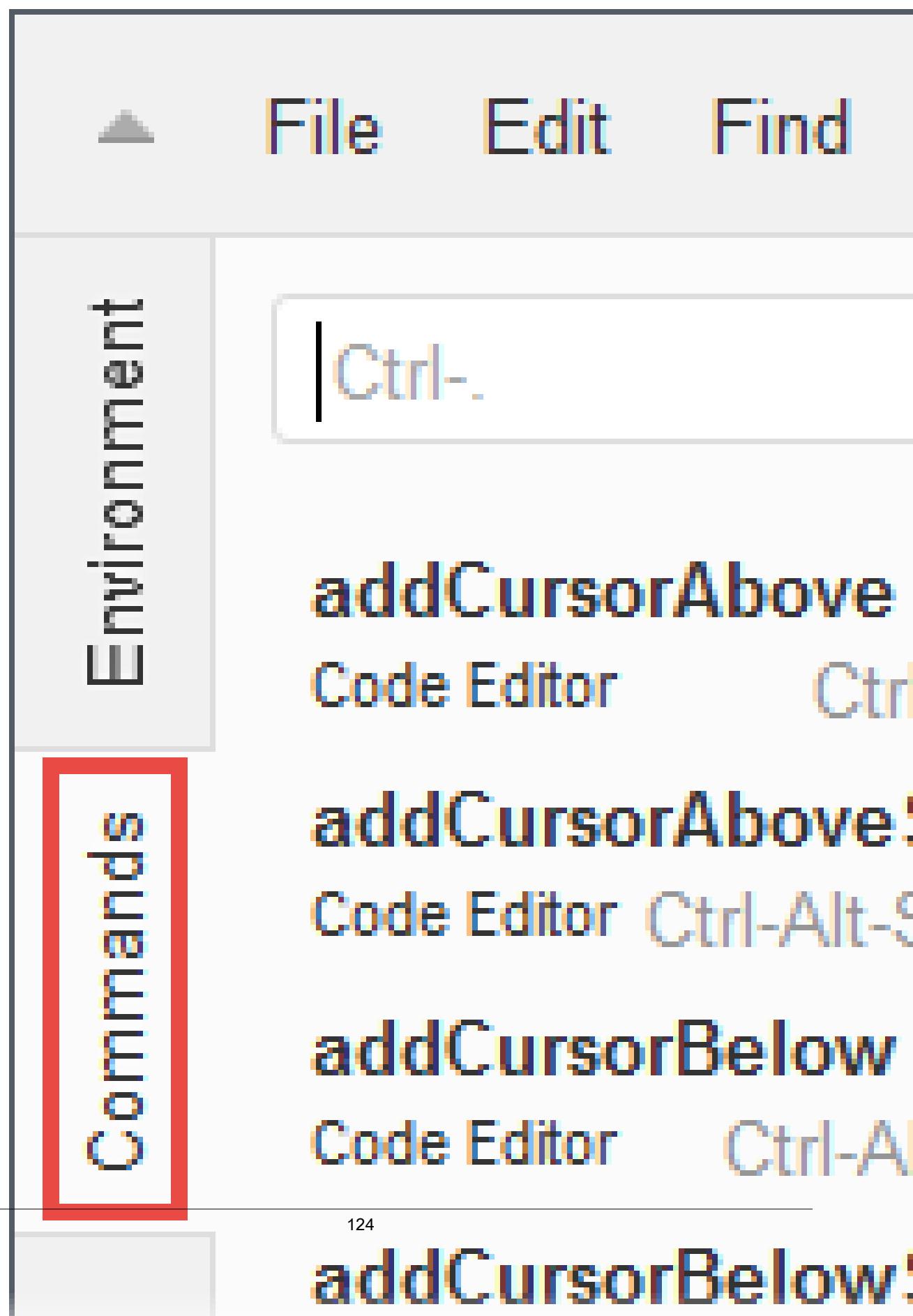
- [您可以执行的项目设置更改](#)
- [您可以执行的用户设置更改](#)

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。

使用命令

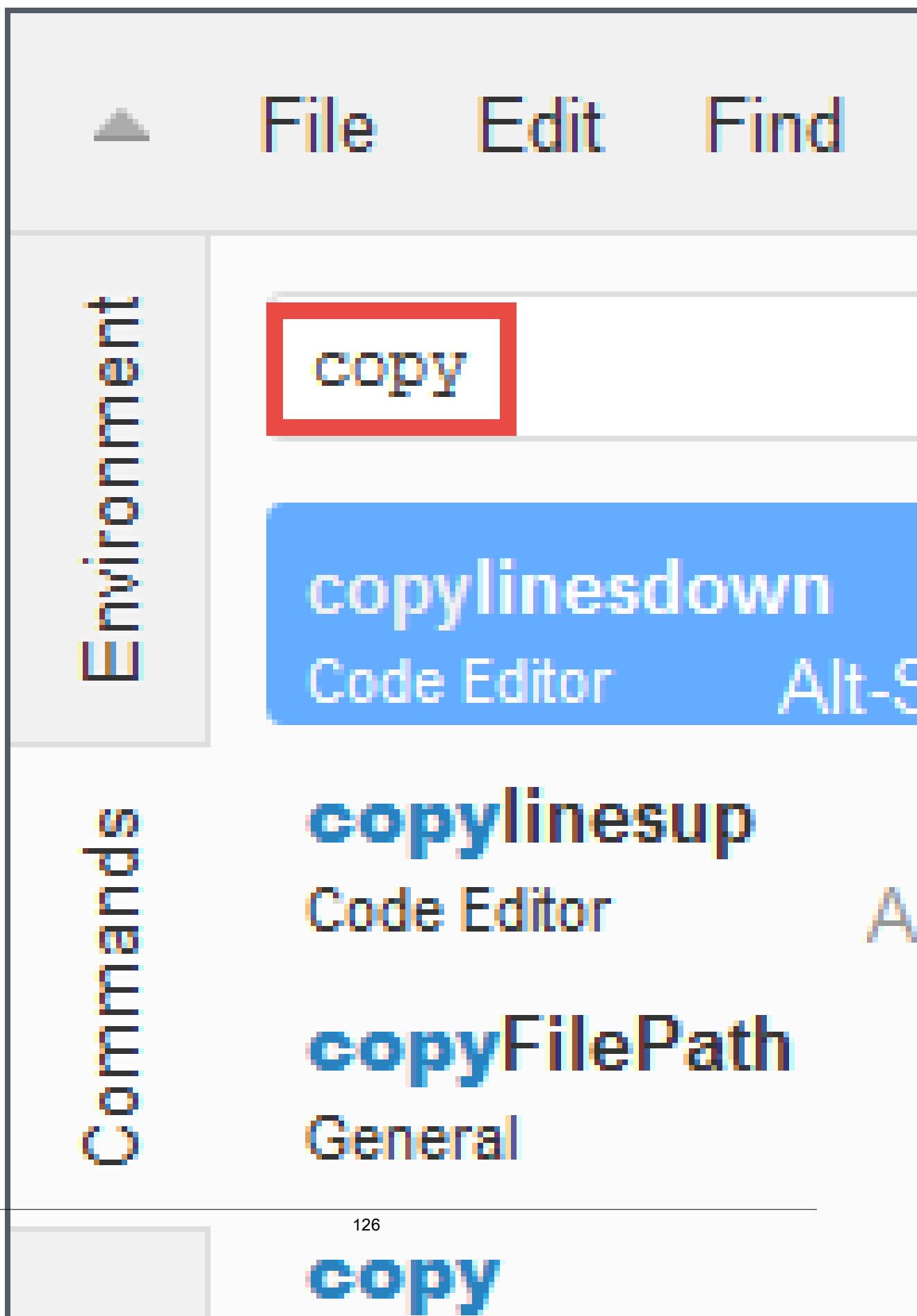
您可以使用 Commands 窗口运行各种命令，例如在菜单栏上、在 Environment 窗口中、在编辑器窗格中找到的那些命令。

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



要运行某个命令，请在 Commands 窗口中选择该命令。

要查找某个命令，请在搜索框中键入该命令的名称的一部分或全部。



有关命令可执行的操作的列表，请参阅 AWS Cloud9 用户指南 中的[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。

此部分中的其他主题包括：

- 配置 Lambda 函数 (p. 127)
- 从 Lambda 函数访问资源 (p. 128)
- AWS Lambda 执行模型 (p. 139)

配置 Lambda 函数

Lambda 函数包含代码以及任意关联依赖项。此外，Lambda 函数还有与之关联的配置信息。最初，您在创建 Lambda 函数时指定配置信息。Lambda 为您提供 API 来更新一些配置数据。Lambda 函数配置信息包括以下关键要素：

- 您需要的计算资源 - 您仅指定要为您的 Lambda 函数分配的内存量。AWS Lambda 分配与内存成比例的 CPU 处理能力，方式是使用与通用 Amazon EC2 实例类型（如 M3 类型）相同的比率。例如，如果为您的 Lambda 函数分配 256 MB 内存，则该函数获得的 CPU 份额将是仅分配 128 MB 内存时的两倍。

您可以更新配置，以 64 MB 的增量请求额外的内存（从 128MB 到 3008 MB）。有关相关限制的信息，请参阅[AWS Lambda 限制 \(p. 369\)](#)。

要更改您的 Lambda 函数所需的内存量，请执行以下操作：

1. 登录 AWS 管理控制台，然后导航到 AWS Lambda 控制台。
2. 选择您要更改其内存大小的函数。
3. 单击 Configuration 选项卡，然后展开 Advanced settings。
4. 在 Memory (MB) 列表中，选择您所需的内存量。

(可选) 您可以使用以下 AWS CLI 命令更新您的函数的内存大小（使用有效的 64 MB 增量）：

```
$ aws lambda update-function-configuration \
  --function-name your function name \
  --region region where your function resides \
  --memory-size memory amount \
  --profile adminuser
```

有关设置和使用 AWS CLI 的信息，请参阅[设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)。

- 最长执行时间（超时）- 为运行您的 Lambda 函数所用的 AWS 资源支付费用。为了防止您的 Lambda 无限期运行，您可以指定超时。在达到指定超时时间时，AWS Lambda 会终止您的 Lambda 函数。

Note

默认开发工具包客户端将自动使超出 50 秒的任何 Lambda 函数请求超时。要适应需要更多处理时间的函数，可通过修改开发工具包客户端配置来提高默认超时限制。

- IAM 角色（执行角色）- 这是 AWS Lambda 在代表您执行 Lambda 函数时使用的角色。有关更多信息，请参阅[AWS Lambda 权限模型 \(p. 339\)](#)。
- 处理程序名称- 处理程序是指您的代码中 AWS Lambda 开始执行的方法。AWS Lambda 将触发了调用的任意事件的信息作为参数传递到处理程序方法。

从 Lambda 函数访问资源

Lambda 不对您的函数逻辑施加任何限制 – 如果您可以对逻辑进行编码，则可在 Lambda 函数内部运行此逻辑。在您的函数中，您可能需要调用其他 API，或者访问其他 AWS 服务（如数据库）。

访问 AWS 服务

要访问其他 AWS 服务，您可以使用 AWS 软件开发工具包（[Node.js](#)、[Java](#)、[Python](#)、[C#](#) 或 [Go](#)），AWS Lambda 会自动将软件开发工具包所需的凭证设置为与您的函数关联的 IAM 角色的凭证，您无需采取任何其他步骤。例如，以下示例代码使用 Python 开发工具包访问 S3 对象：

```
import boto3
import botocore

BUCKET_NAME = 'my-bucket' # replace with your bucket name
KEY = 'my_image_in_s3.jpg' # replace with your object key

s3 = boto3.resource('s3')

try:
    s3.Bucket(BUCKET_NAME).download_file(KEY, 'my_local_image.jpg')
except botocore.exceptions.ClientError as e:
    if e.response['Error']['Code'] == "404":
        print("The object does not exist.")
    else:
        raise
```

Note

为方便起见，AWS Lambda 包含 AWS 开发工具包版本作为执行环境的一部分，这样一来，您无需将它包含在内。有关包含的开发工具包版本，请参阅[Lambda 执行环境和可用库 \(p. 366\)](#)。建议您包含自己的适用于生产应用程序的 AWS 开发工具包副本，以便能控制自己的依赖项。

访问非 AWS 服务

您可以包含任何开发工具包以将任何服务作为 Lambda 函数的一部分进行访问。有关更多信息，请参阅[创建部署程序包 \(p. 77\)](#)。例如，您可以包含适用于 Twilio 的开发工具包以访问 Twilio 账户中的信息。在对开发工具包的凭证进行加密后，您可以使用[环境变量 \(p. 354\)](#)存储凭证信息。

访问私有服务或资源

默认情况下，您的服务或 API 必须能通过公共 Internet 进行访问，以便 AWS Lambda 能够访问该服务或 API。但是，您可能拥有未通过此方式公开的 API 或服务。通常，您在 Amazon Virtual Private Cloud (Amazon VPC) 内部创建这些资源，以便这些资源不能通过公共 Internet 访问。这些资源可以是 AWS 服务资源，例如 Amazon Redshift 数据仓库、Amazon ElastiCache 集群或 Amazon RDS 实例。它们还可以是在您自己的 EC2 实例上运行的您自己的服务。默认情况下，无法从 Lambda 函数访问 VPC 中的资源。

默认情况下，AWS Lambda 在 VPC 中安全运行函数代码。不过，要让您的 Lambda 函数能够访问私有 VPC 中的资源，您必须提供包括 VPC 子网 ID 和安全组 ID 在内的其他 VPC 特定的配置信息。AWS Lambda 使用此信息设置弹性网络接口 (ENI)，此接口可让您的函数安全连接到私有 VPC 中的其他资源。

Important

AWS Lambda 不支持连接到专用租赁 VPC 中的资源。有关更多信息，请参阅[专用 VPC](#)。

要了解如何配置 Lambda 函数以访问 VPC 中的资源，请参阅[配置 Lambda 函数以访问 Amazon VPC 中的资源 \(p. 129\)](#)

配置 Lambda 函数以访问 Amazon VPC 中的资源

通常，您在 Amazon Virtual Private Cloud (Amazon VPC) 内部创建资源，以便这些资源不能通过公共 Internet 访问。这些资源可以是 AWS 服务资源，例如 Amazon Redshift 数据仓库、Amazon ElastiCache 群集或 Amazon RDS 实例。它们还可以是在您自己的 EC2 实例上运行的您自己的服务。默认情况下，无法从 Lambda 函数访问 VPC 中的资源。

默认情况下，AWS Lambda 在 VPC 中安全运行函数代码。不过，要让您的 Lambda 函数能够访问私有 VPC 中的资源，您必须提供包括 VPC 子网 ID 和安全组 ID 在内的其他 VPC 特定的配置信息。AWS Lambda 使用此信息设置弹性网络接口 (ENI)，此接口可让您的函数安全连接到私有 VPC 中的其他资源。

Important

AWS Lambda 不支持连接到专用租赁 VPC 中的资源。有关更多信息，请参阅[专用 VPC](#)。

配置 Lambda 函数以访问 Amazon VPC

您可以在创建 Lambda 函数时，使用 `vpcConfig` 参数将 VPC 信息添加到您的 Lambda 函数配置（请参阅[CreateFunction \(p. 387\)](#)），也可以将其添加到现有的 Lambda 函数配置（请参阅[UpdateFunctionConfiguration \(p. 477\)](#)）。下面是一些 AWS CLI 示例：

- 在您创建 Lambda 函数时，`create-function` CLI 命令将指定 `--vpc-config` 参数来提供 VPC 信息。请注意，`--runtime` 参数指定 `python3.6`。您还可以使用 `python2.7`。

```
$ aws lambda create-function \
--function-name ExampleFunction \
--runtime python3.6 \
--role execution-role-arn \
--zip-file fileb://path/app.zip \
--handler app.handler \
--vpc-config SubnetIds=comma-separated-vpc-subnet-ids,SecurityGroupIds=comma-separated-
security-group-ids \
--memory-size 1024
```

Note

Lambda 函数执行角色必须具有创建、描述和删除 ENI 的权限。AWS Lambda 提供了权限策略 `AWSLambdaVPCAccessExecutionRole`，该策略将授予对必要的 EC2 操作 (`ec2:CreateNetworkInterface`、`ec2:DescribeNetworkInterfaces` 和 `ec2:DeleteNetworkInterface`) 的权限，您可以在创建角色时使用这些权限。您可以在 IAM 控制台中查看策略。不要在您的 Lambda 函数执行后立即删除此角色。Lambda 函数执行与 ENI 删除之间有些延迟。如果您在函数执行后立即删除此角色，则您需要负责删除 ENI。

- 可以在 `update-function-configuration` CLI 命令中指定 `--vpc-config` 参数，以将 VPC 信息添加到现有的 Lambda 函数配置。

```
$ aws lambda update-function-configuration \
--function-name ExampleFunction \
--vpc-config SubnetIds=comma-separated-vpc-subnet-ids,SecurityGroupIds=security-group-ids
```

要从 Lambda 函数配置中删除与 VPC 相关的信息，请使用 `UpdateFunctionConfiguration` API，并提供子网 ID 和安全组 ID 的空白列表，如以下 CLI 命令示例所示。

```
$ aws lambda update-function-configuration \
--function-name ExampleFunction \
```

```
--vpc-config SubnetIds=[],SecurityGroupIds=[]
```

请注意以下其他几项注意事项：

- 当您将 VPC 配置添加到 Lambda 函数时，它只能访问该 VPC 中的资源。如果 Lambda 函数既要访问 VPC 资源又要访问公共 Internet，那么 VPC 内部必须具有网络地址转换 (NAT) 实例。
- 当 Lambda 函数配置为在 VPC 中运行时，它会带来额外的 ENI 启动性能损失。这意味着，在尝试连接到网络资源时，可能会延迟地址解析。

对 Lambda 函数的 Internet 访问

AWS Lambda 使用您提供的 VPC 信息来设置 ENI，使您的 Lambda 函数能够访问 VPC 资源。将向每个 ENI 分配一个来自您指定的子网中 IP 地址范围的私有 IP 地址，但不会分配任何公共 IP 地址。因此，如果您的 Lambda 函数需要访问 Internet（例如，访问没有 VPC 终端节点的 AWS 服务），您可以在 VPC 中配置 NAT 实例，也可以使用 Amazon VPC NAT 网关。有关更多信息，请参阅 Amazon VPC 用户指南中的 [NAT 网关](#)。您无法使用附加到 VPC 的 Internet 网关，因为这要求 ENI 具有公共 IP 地址。

Important

如果您的 Lambda 函数需要访问 Internet，请不要将它附加到公共子网或未接入 Internet 的私有子网。相反，仅通过 NAT 实例或 Amazon VPC NAT 网关将它附加到接入了 Internet 的私有子网。

设置支持 VPC 的 Lambda 函数的准则

您的 Lambda 函数会自动根据它处理的事件数量进行扩展。下面提供了设置支持 VPC 的 Lambda 函数来支持扩展行为的一般准则。

- 如果您的 Lambda 函数访问 VPC，则必须确保您的 VPC 具备足够的 ENI 容量，可以满足您的 Lambda 函数的扩展需求。您可以使用以下公式来确定大致的 ENI 容量。

```
Projected peak concurrent executions * (Memory in GB / 3GB)
```

其中：

- Projected peak concurrent execution - 使用 [管理并发 \(p. 350\)](#) 中的信息来确定此值。
- Memory - 您为 Lambda 函数配置的内存量。
- 您指定的子网应具有足量的可用 IP 地址以匹配 ENI 的数目。

我们还建议您在您的 Lambda 函数配置的每个可用区中至少指定一个子网。通过在每个可用区中指定子网，如果一个可用区出现故障或者 IP 地址不足，您的 Lambda 函数可以在其他可用区中运行。

Note

如果您的 VPC 没有足够的 ENI 或子网 IP，您的 Lambda 函数将不会随着请求的增加而扩展，并且您将会看到函数失败的次数不断增加。AWS Lambda 当前不会将 ENI 或子网 IP 不足而导致的错误记录到 CloudWatch Logs。如果您看到错误不断增加而没有对应的 CloudWatch Logs，则可以同步调用 Lambda 函数来获取错误响应（例如，在 AWS Lambda 控制台中测试您的 Lambda 函数，因为控制台会同步调用您的 Lambda 函数并显示错误）。

教程：配置 Lambda 函数以访问 Amazon VPC 中的资源

本节提供了端到端的示例教程，指导您创建并配置 Lambda 函数来访问 Amazon VPC 中的资源，例如 Amazon ElastiCache 群集或 Amazon RDS 数据库实例。

主题

- [教程：配置 Lambda 函数以访问 Amazon VPC 中的 Amazon ElastiCache \(p. 131\)](#)
- [教程：配置 Lambda 函数以访问 Amazon VPC 中的 Amazon RDS \(p. 135\)](#)

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 us-east-1 区域的默认 Amazon Virtual Private Cloud (Amazon VPC) 中创建 Amazon ElastiCache 群集。有关 Amazon ElastiCache 的更多信息，请参阅 [Amazon ElastiCache](#)。
- 创建 Lambda 函数以访问 ElastiCache 群集。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。
- 手动调用 Lambda 函数，并确保它访问了您的 VPC 中的 ElastiCache 群集。

Important

本教程使用了您账户的 us-east-1 区域中的默认 Amazon VPC。有关 Amazon VPC 的更多信息，请参阅 Amazon VPC 用户指南 中的 [Amazon VPC 入门](#)。

下一步

[步骤 1：创建 ElastiCache 群集 \(p. 131\)](#)

步骤 1：创建 ElastiCache 群集

在此步骤中，您在自己账户的 us-east-1 区域的默认 Amazon VPC 中创建 ElastiCache 群集。

1. 运行以下 AWS CLI 命令以在您账户的 us-east-1 区域的默认 VPC 中创建 Memcached 群集。

```
aws elasticache create-cache-cluster \
--cache-cluster-id ClusterForLambdaTest \
--cache-node-type cache.m3.medium \
--engine memcached \
--security-group-ids your-default-vpc-security-group \
--num-cache-nodes 1
```

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 函数将在该群集中添加和检索项目。

您还可以使用 Amazon ElastiCache 控制台启动缓存群集。有关说明，请参阅 Amazon ElastiCache 用户指南 中的 [Amazon ElastiCache 入门](#)。

2. 记下您启动的缓存群集的配置终端节点。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。

下一步

[步骤 2：创建 Lambda 函数 \(p. 132\)](#)

步骤 2：创建 Lambda 函数

在此步骤中，您将执行以下操作：

- 使用提供的示例代码创建 Lambda 函数部署程序包。
- 创建 IAM 角色（执行角色）。在上传部署程序包时，需要指定此角色以便 Lambda 代入该角色，然后代表您执行该函数。
权限策略将授予 AWS Lambda 设置弹性网络接口 (ENI) 的权限，从而使您的 Lambda 函数能够访问 VPC 中的资源。在本示例中，您的 Lambda 函数将访问 VPC 中的 ElastiCache 群集。
- 通过上传部署程序包来创建 Lambda 函数。

主题

- [步骤 2.1：创建部署程序包 \(p. 132\)](#)
- [步骤 2.2：创建执行角色（IAM 角色）\(p. 133\)](#)
- [步骤 2.3：创建 Lambda 函数（上传部署程序包）\(p. 133\)](#)

步骤 2.1：创建部署程序包

Note

目前，只提供使用 Python 编写的 Lambda 函数的示例代码。

Python

以下 Python 代码示例在 ElastiCache 群集中读取和写入项目。

1. 打开文本编辑器，然后复制以下代码。

Note

利用 `from __future__ import print_function` 语句，可以编写与 Python 2 或 3 兼容的代码。如果您使用的是运行时版本 3.6，则不必将它包含在内。

```
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-elasticacluster-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 = str(uuid.uuid4())
    #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 == 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
```

2. 将该文件保存为 app.py。
3. 使用 pip 安装以下库依赖项：

- 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（请参阅 [pymemcache](#)）。
- elasticache-auto-discovery - Lambda 函数使用此库来获取 Amazon ElastiCache 群集中的节点（请参阅 [elasticache-auto-discovery](#)）。

4. 将所有这些文件压缩成名为 app.zip 的文件来创建您的部署程序包。如需分步指导，请参阅 [创建部署程序包 \(Python\) \(p. 94\)](#)。

下一步

[步骤 2.2：创建执行角色 \(IAM 角色 \) \(p. 133\)](#)

步骤 2.2：创建执行角色 (IAM 角色)

在此步骤中，您将使用以下预定义的角色类型和访问权限策略创建 AWS Identity and Access Management (IAM) 角色：

- AWS Lambda (AWS 服务角色) - 此角色为 AWS Lambda 授予代入该角色的权限。
- AWSLambdaVPCAccessExecutionRole (访问权限策略) - 这是您附加到该角色的策略。该策略授予对 EC2 操作的权限，AWS Lambda 需要该权限来管理 ENI。您可以在 IAM 控制台中查看此 AWS 托管策略。

有关 IAM 角色的更多信息，请参阅 IAM 用户指南 中的 [IAM 角色](#)。使用以下程序创建 IAM 角色。

创建 IAM 角色 (执行角色)

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 按照 IAM 用户指南 中[创建角色以向 AWS 服务委派权限](#)的步骤创建 IAM 角色 (执行角色)。遵循步骤 创建角色时，请注意以下事项：
 - 在 Role Name 中，使用在 AWS 账户内唯一的名称（例如，lambda-vpc-execution-role）。
 - 在 Select Role Type 中，选择 AWS Service Roles，然后选择 AWS Lambda。这将为 AWS Lambda 服务授予代入 IAM 角色的权限。
 - 在 Attach Policy 中，选择 AWSLambdaVPCAccessExecutionRole。此策略中的权限足以用于本教程中的 Lambda 函数。
3. 记下角色 ARN。在下一步中，创建 Lambda 函数时需要用到它。

下一步

[步骤 2.3：创建 Lambda 函数 \(上传部署程序包 \) \(p. 133\)](#)

步骤 2.3：创建 Lambda 函数 (上传部署程序包)

在此步骤中，您将使用 `create-function` AWS CLI 命令创建 Lambda 函数 (AccessMemCache)。

在命令提示符处，使用 adminuser profile 运行下面的 Lambda CLI `create-function` 命令。

您需要提供 .zip 文件路径和执行角色 ARN 来更新以下 `create-function` 命令。`--runtime` 参数值可以是 `python3.6`、`python2.7`、`nodejs` 和 `java8`，具体取决于您编写代码时所用的语言。

Note

目前，只提供使用 Python 编写的 Lambda 函数的示例代码。

```
$ aws lambda create-function \
--function-name AccessMemCache \
--region us-east-1 \
--zip-file fileb://path-to/app.zip \
--role execution-role-arn \
--handler app.handler \
--runtime python3.6 \
--timeout 30 \
--vpc-config SubnetIds=comma-separated-vpc-subnet-ids,SecurityGroupIds=default-security-group-id \
--memory-size 1024
```

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。

或者，您也可以将 .zip 文件上传到同一 AWS 区域中的 Amazon S3 存储桶，然后在之前的命令中指定该存储桶和对象名称。您需要将 `--zip-file` 参数替换为 `--code` 参数，如下所示：

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

Note

您还可以使用 AWS Lambda 控制台创建 Lambda 函数。在创建函数时，为 Lambda 选择 VPC，然后从提供的字段中选择子网和安全组。

下一步

[步骤 3：测试 Lambda 函数（手动调用）\(p. 134\)](#)

[步骤 3：测试 Lambda 函数（手动调用）](#)

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，并将它写入到您的 Lambda 代码中指定的 ElastiCache 群集。然后，Lambda 函数将从缓存中检索项目。

1. 使用 AWS Lambda `invoke` 命令调用 Lambda 函数 (AccessMemCache)。

```
$ aws lambda invoke \
--function-name AccessMemCache \
--region us-east-1 \
--profile adminuser \
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。

接下来做什么？

您已经创建了 Lambda 函数来访问您的 VPC 中的 ElastiCache 群集，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 [使用案例 \(p. 165\)](#)。

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。
- 手动调用 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。

Important

本教程使用了您账户的 us-east-1 区域中的默认 Amazon VPC。有关 Amazon VPC 的更多信息，请参阅 Amazon VPC 用户指南 中的 [Amazon VPC 入门](#)。

下一步

[步骤 1：创建 Amazon RDS MySQL 实例和 ExampleDB 数据库 \(p. 135\)](#)

[步骤 1：创建 Amazon RDS MySQL 实例和 ExampleDB 数据库](#)

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。

Important

本教程使用在 us-east-1 区域的默认 VPC 中启动的 RDS MySQL 数据库引擎。

您可以使用以下方法之一启动 RDS MySQL 实例：

- 按照 Amazon Relational Database Service 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-instance-identifier MySQLForLambdaTest \
  --db-instance-class db.t2.micro \
  --engine MySQL \
  --allocated-storage 5 \
  --no-publicly-accessible \
  --db-name ExampleDB \
  --master-username username \
  --master-user-password password \
  --backup-retention-period 3
```

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 控制台获取该信息（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）。

下一步

[步骤 2：创建 Lambda 函数 \(p. 136\)](#)

步骤 2：创建 Lambda 函数

在此步骤中，您将执行以下操作：

- 使用提供的示例代码创建 Lambda 函数部署程序包。
- 创建您在创建 Lambda 函数时指定的 IAM 角色（执行角色）。这是 AWS Lambda 在执行 Lambda 函数时代入的角色。
与此角色关联的权限策略将授予 AWS Lambda 设置弹性网络接口 (ENI) 的权限，从而使您的 Lambda 函数能够访问 VPC 中的资源。
- 通过上传部署程序包来创建 Lambda 函数。

主题

- [步骤 2.1：创建部署程序包 \(p. 136\)](#)
- [步骤 2.2：创建执行角色 \(IAM 角色\) \(p. 137\)](#)
- [步骤 2.3：创建 Lambda 函数 \(上传部署程序包\) \(p. 138\)](#)

步骤 2.1：创建部署程序包

Note

目前，只提供使用 Python 编写的 Lambda 函数的示例代码。

Python

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。

1. 打开文本编辑器，然后复制以下代码。

```
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:
    logger.error("ERROR: Unexpected error: Could not connect to MySql instance.")
    sys.exit()

logger.info("SUCCESS: Connection to RDS mysql instance succeeded")
def handler(event, context):
    """
    This function fetches content from mysql RDS instance
    """
    pass
```

```
"""
item_count = 0

with conn.cursor() as cur:
    cur.execute("create table Employee3 ( EmpID  int NOT NULL, Name varchar(255) NOT NULL, PRIMARY KEY (EmpID))")
    cur.execute('insert into Employee3 (EmpID, Name) values(1, "Joe")')
    cur.execute('insert into Employee3 (EmpID, Name) values(2, "Bob")')
    cur.execute('insert into Employee3 (EmpID, Name) values(3, "Mary")')
    conn.commit()
    cur.execute("select * from Employee3")
    for row in cur:
        item_count += 1
        logger.info(row)
        #print(row)

return "Added %d items from RDS MySQL table" %(item_count)
```

Note

我们建议按照所示方式在处理程序之外执行 `pymysql.connect()`，以便获得更好的性能。

2. 将该文件保存为 `app.py`。
3. 使用 pip 安装以下库依赖项：
 - `pymysql` - Lambda 函数代码使用此库来访问您的 MySQL 实例（请参阅 [PyMySQL](#)）。
4. 创建包含以下信息的配置文件并将其保存为 `rds_config.py`：

```
#config file containing credentials for rds mysql instance
db_username = "username"
db_password = "password"
db_name = "databasename"
```

5. 将所有这些文件压缩成名为 `app.zip` 的文件来创建您的部署程序包。如需分步指导，请参阅 [创建部署程序包 \(Python\) \(p. 94\)](#)。

下一步

[步骤 2.2：创建执行角色 \(IAM 角色 \) \(p. 137\)](#)

步骤 2.2：创建执行角色 (IAM 角色)

在此步骤中，您将使用以下预定义的角色类型和访问权限策略，为您的 Lambda 函数创建执行角色 (IAM 角色)：

- AWS Lambda (AWS 服务角色) - 此角色为 AWS Lambda 授予代入该角色的权限。
- AWSLambdaVPCAccessExecutionRole (访问权限策略) - 此角色向 AWS Lambda 授予对 EC2 操作的权限以创建 ENI 并且使您的 Lambda 函数可以访问 VPC 资源，并授予对 CloudWatch Logs 操作的权限以写入日志。

有关 IAM 角色的更多信息，请参阅 IAM 用户指南 中的 [IAM 角色](#)。使用以下程序创建 IAM 角色。

创建 IAM 角色 (执行角色)

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 按照 IAM 用户指南 中[创建角色以向 AWS 服务委派权限](#)的步骤创建 IAM 角色 (执行角色)。遵循步骤 创建角色时，请注意以下事项：

- 在 Role Name 中，使用在 AWS 账户内唯一的名称（例如，lambda-vpc-execution-role）。
 - 在 Select Role Type 中，选择 AWS Service Roles，然后选择 AWS Lambda。这将为 AWS Lambda 服务授予代入 IAM 角色的权限。
 - 在 Attach Policy 中，选择 AWSLambdaVPCAccessExecutionRole。此策略中的权限足以用于本教程中的 Lambda 函数。
3. 记下角色 ARN。在下一步中，创建 Lambda 函数时需要用到它。

下一步

[步骤 2.3：创建 Lambda 函数（上传部署程序包）\(p. 138\)](#)

步骤 2.3：创建 Lambda 函数（上传部署程序包）

在此步骤中，您将使用 `create-function` AWS CLI 命令创建 Lambda 函数 (`ReadMySqlTable`)。

在命令提示符处，使用 `adminuser` profile 运行下面的 Lambda CLI `create-function` 命令。

您需要提供 `.zip` 文件路径和执行角色 ARN 来更新以下 `create-function` 命令。`--runtime` 参数值可以是 `python2.7`、`nodejs` 或 `java8`，具体取决于您编写代码所用的语言。

Note

目前，只提供使用 Python 编写的 Lambda 函数的示例代码。`--runtime` 参数可以使用 `python3.6` 或 `python2.7`。

```
$ aws lambda create-function \
--region us-east-1 \
--function-name CreateTableAddRecordsAndRead \
--zip-file fileb://file-path/app.zip \
--role execution-role-arn \
--handler app.handler \
--runtime python3.6 \
--vpc-config SubnetIds=comma-separated-subnet-ids,SecurityGroupIds=default-vpc-security-
group-id \
--profile adminuser
```

或者，您也可以将 `.zip` 文件上传到同一 AWS 区域中的 Amazon S3 存储桶，然后在之前的命令中指定该存储桶和对象名称。您需要将 `--zip-file` 参数替换为 `--code` 参数，如下所示：

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

Note

您还可以使用 AWS Lambda 控制台创建 Lambda 函数（使用在上一 CLI 命令中所示的参数值）。

下一步

[步骤 3：测试 Lambda 函数（手动调用）\(p. 138\)](#)

步骤 3：测试 Lambda 函数（手动调用）

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 AWS Lambda `invoke` 命令调用 Lambda 函数 (`ReadMySqlTable`)。

```
$ aws lambda invoke \
```

```
--function-name CreateTableAddRecordsAndRead \
--region us-east-1 \
--profile adminuser \
output.txt
```

- 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。

AWS Lambda 执行模型

AWS Lambda 代表您执行您的 Lambda 函数时，它负责预配置和管理运行 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。

Note

这一部分的内容仅用于信息目的。AWS Lambda 管理执行上下文创建和删除操作，没有供您管理执行上下文的 AWS Lambda API。

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：

- 您的 Lambda 函数代码（ handler 代码的外部，请参阅 [编程模型 \(p. 18\)](#) ）中的任何声明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。
- 每个执行上下文提供 500 MB 的额外磁盘空间在 /tmp 目录中。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 \(p. 369\)](#)。
- 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（使用 Node.js 时）在代码退出前已完成。

Note

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）。如前所述，向 Lambda 函数代码添加逻辑以检查是否存在执行上下文。

调用 Lambda 函数

在 AWS Lambda 上构建应用程序 (包括无服务器应用程序) 时，核心组件是 Lambda 函数和事件源。事件源是发布事件的 AWS 服务或自定义应用程序，Lambda 函数 是处理事件的自定义代码。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。

每个这些事件源为事件数据使用特定格式。有关更多信息，请参阅 [事件源发布的示例事件 \(p. 154\)](#)。调用 Lambda 函数时，它接收事件作为 Lambda 函数的参数。

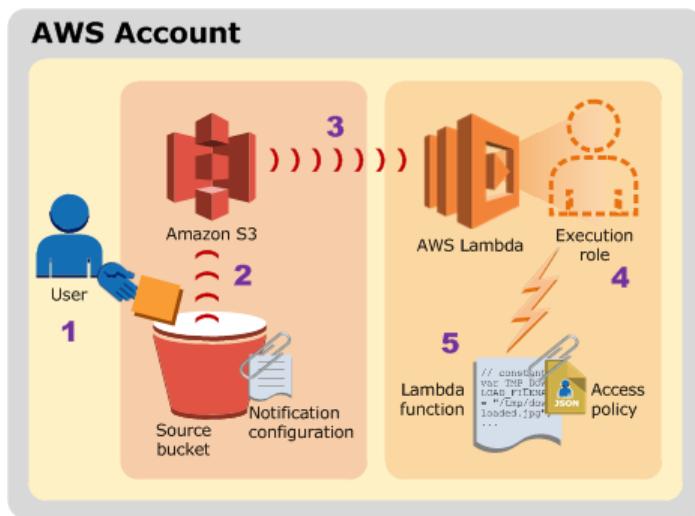
AWS Lambda 支持将多种 AWS 服务作为事件源。有关更多信息，请参阅 [支持的事件源 \(p. 148\)](#)。当您配置了这些事件源触发 Lambda 函数时，Lambda 函数在出现事件时自动调用。您可以定义事件源映射，这是您如何确定要跟踪的事件以及要调用的 Lambda 函数。

除了支持的 AWS 服务之外，用户应用程序还可以生成事件，您可以构建自己的自定义事件源。自定义事件源使用 AWS Lambda [Invoke \(p. 425\)](#) 操作调用 Lambda 函数。用户应用程序（如客户端、手机或 Web 应用程序）可以发布事件，并使用 AWS 开发工具包或 AWS 移动开发工具包（如适用于 Android 的 AWS 移动软件开发工具包）按需调用 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 使用由执行角色提供的权限来调用 Lambda 函数。有关执行角色的更多信息，请参阅 [AWS Lambda 的身份验证和访问控制 \(p. 319\)](#)。Amazon S3 根据存储在存储桶通知配置中的事件源映射明确应该调用哪个 Lambda 函数。
4. AWS Lambda 执行 Lambda 函数，指定事件作为参数。

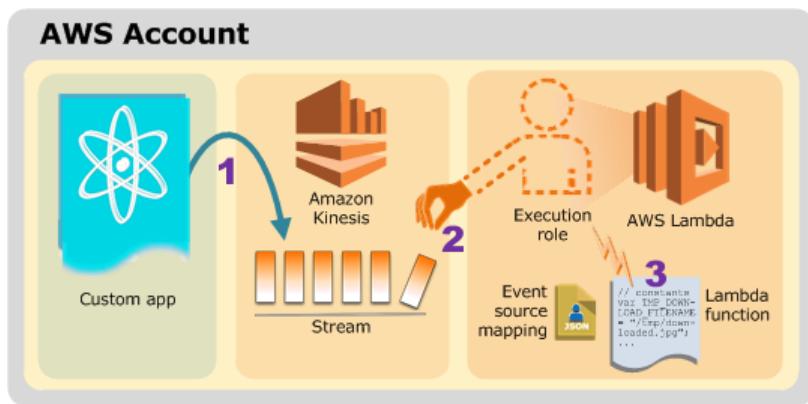
请注意以下几点：

- 事件源映射在事件源服务中维护，在此情景下为 Amazon S3。对于所有支持的 AWS 事件源，除了基于流的源（Kinesis 和 DynamoDB 流）之外，均是如此。下一个示例说明了基于流的事件源。
- 事件源（Amazon S3）调用 Lambda 函数（称为推模型）。再次强调，对于所有支持的 AWS 服务，除了基于流的事件源之外，均是如此。
- 要让事件源（Amazon S3）调用您的 Lambda 函数，您必须使用附加到 Lambda 函数的权限策略来授予权限。

示例 2：AWS Lambda 从 Kinesis 流中拉取事件并调用 Lambda 函数

对于基于流的事件源，AWS Lambda 轮询流，在流上检测到记录时调用 Lambda 函数。这些流源的特殊之处在于，事件源映射信息存储在 Lambda 中。AWS Lambda 为您提供用于创建和管理这些事件源映射的 API。

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。

请注意以下几点：

- 在处理基于流的事件源时，会发生以下情况：
 - 您在 AWS Lambda 中创建事件源映射。
 - AWS Lambda 调用 Lambda 函数（称为拉模型）。
- AWS Lambda 无需权限即可调用您的 Lambda 函数，因此您不需要添加任何权限到附加到您 Lambda 函数的权限策略。
- 您的 Lambda 角色需要从流读取的权限。

调用类型

AWS Lambda 支持同步和异步调用 Lambda 函数。只有在您调用 Lambda 函数时，您才能控制调用类型（称为按需调用）。以下示例说明了按需调用：

- 您的自定义应用程序调用 Lambda 函数。
- 您手动调用 Lambda 函数（例如，使用 AWS CLI）用于测试用途。

在这两种情况下，您使用 [Invoke \(p. 425\)](#) 操作调用 Lambda 函数，并且可以指定同步或异步调用类型。

但是，在您使用 AWS 服务作为事件源时，每个这些服务的调用类型是预定义的。对于这些事件源在调用您的 Lambda 函数时使用的调用类型，您没有任何控制方法。例如，Amazon S3 始终异步调用 Lambda 函数，Amazon Cognito 始终同步调用 Lambda 函数。对基于流的 AWS 服务（Amazon Kinesis Streams 和 Amazon DynamoDB Streams），AWS Lambda 轮询流并同步调用您的 Lambda 函数。

事件源映射

在 AWS Lambda 中，Lambda 函数和事件源是 AWS Lambda 中的核心组件。事件源是发布事件的实体，Lambda 函数是处理事件的自定义代码。支持的事件源是指那些经过预配置可与 AWS Lambda 一起

使用的 AWS 服务。配置称为事件源映射，将事件源映射到 Lambda 函数。它允许在事件发生时自动调用 Lambda 函数。

每个事件源映射标识要发布的事件类型，以及发生事件时要调用的 Lambda 函数。特定 Lambda 函数随后接收事件信息作为参数，接下来您的 Lambda 函数代码可处理事件。

请注意以下有关事件源的信息。这些事件源可以是以下任意之一：

- **AWS 服务** - 这些是经过预配置可与 AWS Lambda 一起使用的支持的 AWS 服务。您可以将这些服务分组为常规 AWS 服务或基于流的服务。Amazon Kinesis Data Streams 和 Amazon DynamoDB Streams 是基于流的事件源，所有其他 AWS 服务不使用基于流的事件源。您维护事件源映射的位置以及 Lambda 函数的调用方法取决于您是否在使用基于流的事件源。
- **自定义应用程序** - 您可以让自定义应用程序发布事件和调用 Lambda 函数。

您可能会问，我在什么位置保存事件映射信息？我将它保存在事件源中还是保存在 AWS Lambda 中？以下部分说明了每个这些事件源类别的事件源映射。这些部分还说明了如何调用 Lambda 函数以及如何管理权限来允许调用您的 Lambda 函数。

主题

- [AWS 服务的事件源映射 \(p. 143\)](#)
- [AWS 基于流的服务的事件源映射 \(p. 144\)](#)
- [自定义应用程序的事件源映射 \(p. 145\)](#)

AWS 服务的事件源映射

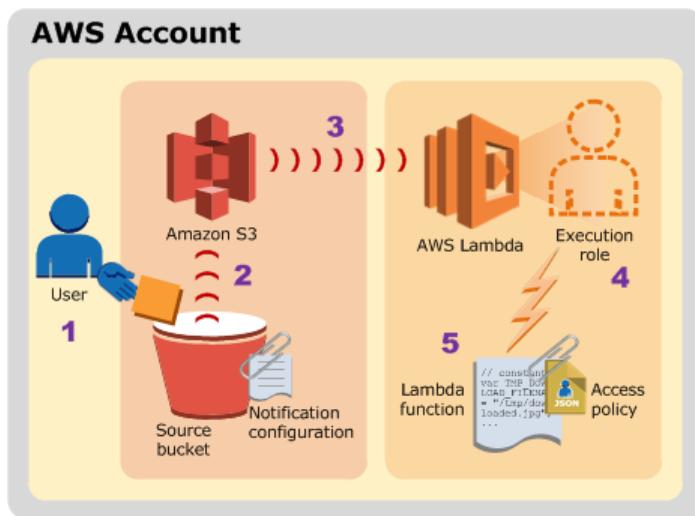
除了基于流的 AWS 服务（Amazon Kinesis Data Streams 和 DynamoDB 流）之外，其他支持的 AWS 服务可发布事件，并且也可以调用您的 Lambda 函数（称为推模型）。在推模型中，请注意以下事项：

- 事件源映射在事件源中维护。事件源中的相关 API 支持使您可以创建和管理事件源映射。例如，Amazon S3 提供存储桶通知配置 API。使用此 API，您可以配置事件源映射，标识存储桶事件以发布和调用 Lambda 函数。
- 由于事件源调用您的 Lambda 函数，您需要使用基于资源的策略授予事件源必需的权限（称为 Lambda 函数策略）。有关更多信息，请参阅 [AWS Lambda 权限模型 \(p. 339\)](#)。

以下示例说明了此模型的工作原理。

Example - Amazon S3 推送事件和调用 Lambda 函数

假定您希望为每个对象创建的存储桶事件调用您的 AWS Lambda 函数。您在存储桶通知配置中添加必需的事件源映射。



下图说明了这一流程：

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 根据在存储桶通知配置中描述的事件源映射调用 Lambda 函数。
4. AWS Lambda 验证附加到 Lambda 函数的策略来确保 Amazon S3 具有所需的权限。有关权限策略的详细信息，请参阅 [AWS Lambda 的身份验证和访问控制 \(p. 319\)](#)
5. AWS Lambda 验证附加的权限策略之后，会执行 Lambda 函数。请记住，您的 Lambda 函数接收事件作为参数。

AWS 基于流的服务的事件源映射

Amazon Kinesis Data Streams 和 DynamoDB 流是基于流的服务，您可以预配置这些服务以便与 AWS Lambda 一起使用。在您执行必要的事件源映射之后，AWS Lambda 轮询流并调用 Lambda 函数（称为轮询模型）。在拉模型中，请注意以下事项：

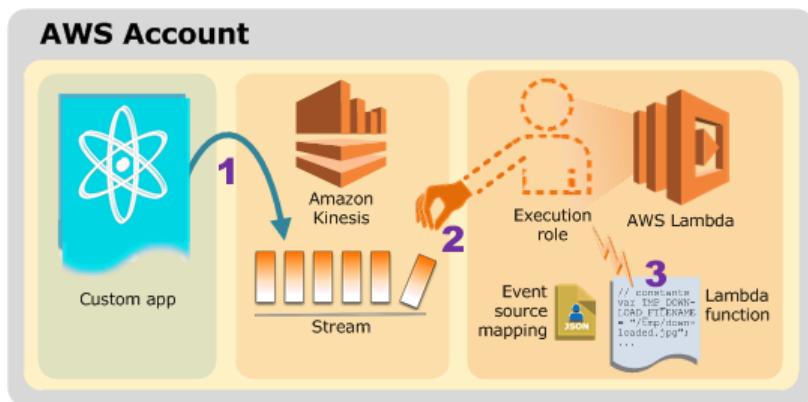
- 事件源映射在 AWS Lambda 中维护。AWS Lambda 提供相关 API 来创建和管理事件源映射。有关更多信息，请参阅 [CreateEventSourceMapping \(p. 382\)](#)。
- AWS Lambda 需要您的权限来轮询流和读取记录。您可以通过执行角色，使用您在创建 Lambda 函数时指定与角色关联的权限策略，授予这些权限。AWS Lambda 无需任何权限即可调用您的 Lambda 函数。

以下示例说明了此模型的工作原理。

Example - AWS Lambda 从 Kinesis 流中提取事件并调用 Lambda 函数

下图显示将记录写入 Kinesis 流的自定义应用程序以及 AWS Lambda 轮询流的方式。当 AWS Lambda 在流上检测到新记录时，它调用您的 Lambda 函数。

假定您有写入记录到 Kinesis 流的自定义应用程序。您希望在流上检测到新记录时调用 Lambda 函数。您可以在 AWS Lambda 中创建 Lambda 函数以及所需的事件源映射。



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 AWS 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. 假定允许 AWS Lambda 轮询流的附加权限策略通过了验证，接下来 AWS Lambda 执行 Lambda 函数。有关权限策略的详细信息，请参阅 [AWS Lambda 的身份验证和访问控制 \(p. 319\)](#)

本示例中使用 Kinesis 流，不过这种情况也适用于使用 DynamoDB 流时。

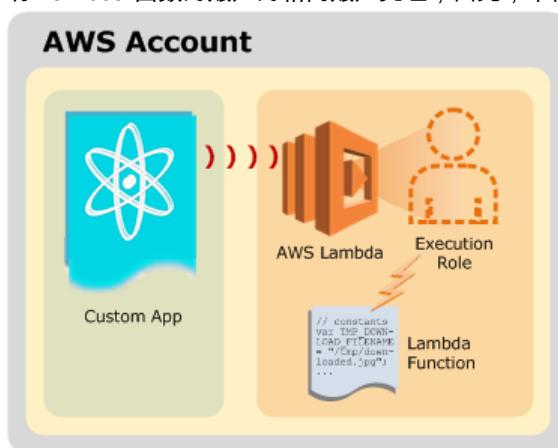
自定义应用程序的事件源映射

如果您有发布和处理事件的自定义应用程序，您可以创建 Lambda 函数来处理这些事件。在这种情况下，没有预配置要求，您无需设置事件源映射。相反，事件源使用 AWS Lambda Invoke API。如果应用程序和 Lambda 函数由不同 AWS 账户拥有，在与 Lambda 函数关联的权限策略中，拥有 Lambda 函数的 AWS 账户必须允许跨账户的权限。

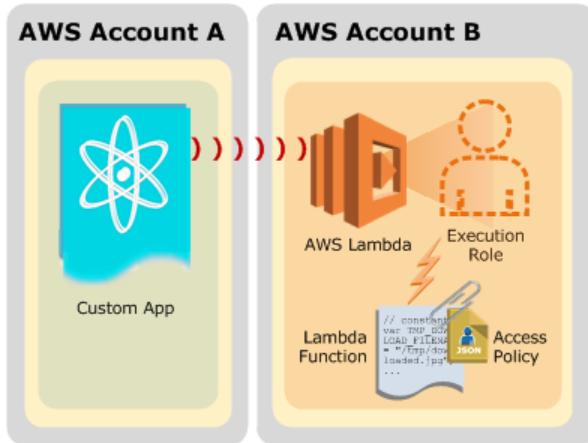
以下示例说明了它的工作原理。

Example - 自定义应用程序发布事件和调用 Lambda 函数

下图说明了您账户中的自定义应用程序如何调用 Lambda 函数的过程。在此示例中，自定义应用程序使用拥有 Lambda 函数的账户的相同账户凭证，因此，不需要其他权限即可调用函数。



在下面的示例中，用户应用程序和 Lambda 函数由不同的 AWS 账户所有。这种情况下，在与 Lambda 函数关联的权限策略中，拥有 Lambda 函数的 AWS 账户必须具有跨账户权限。有关更多信息，请参阅 [AWS Lambda 权限模型 \(p. 339\)](#)。



了解重试行为

Lambda 函数可能会由于以下任意原因而失败：

- 函数在尝试访问终端节点时超时。
- 函数无法成功解析输入数据。
- 函数遇到资源限制，例如内存不足错误或者其他超时。

如果出现任何这些错误，您的函数可能会引发异常错误。如何处理异常错误取决于 Lambda 函数的调用方式：

- 并非基于流的事件源 - 这些事件源中的一部分设置为同步调用 Lambda 函数，另一些则异步调用。相应的异常处理方式分别如下：
 - 同步调用 - 发出调用的应用程序收到 429 错误并负责重试。有关支持的事件源列表及其使用的调用类型，请参阅[支持的事件源](#)。这些事件源可能会有内置到集成中的额外重试。

如果您直接通过 AWS 开发工具包调用 Lambda 函数，则您的客户端会收到错误并可以选择重试。

- 异步调用 - 使用异步事件调用 Lambda 函数之前会对它们进行排队。如果 AWS Lambda 无法完全处理事件，它将自动重试调用两次，且在重试之间有一定的延迟。如果为函数指定了一个死信队列，那么失败的事件将被发送到指定的 Amazon SQS 队列或 Amazon SNS 主题。如果保留默认设置，不指定死信队列 (DLQ) (非必需操作)，那么系统将丢弃事件。有关更多信息，请参阅[死信队列 \(p. 361\)](#)。
- 基于流的事件源 - 对于基于流的事件源 (Amazon Kinesis Data Streams 和 DynamoDB 流)，AWS Lambda 会轮询您的流并调用您的 Lambda 函数。因此，如果 Lambda 函数失败，AWS Lambda 尝试处理错误的记录批次，直至数据过期，对于 Amazon Kinesis Data Streams 这最多可以为七天。将异常作为阻塞处理，并且 AWS Lambda 不会从流中读取任何新记录，直至失败的记录批次过期或者处理成功。这确保 AWS Lambda 按顺序处理流事件。

有关调用模式的更多信息，请参阅[事件源映射 \(p. 142\)](#)。

了解扩展行为

并发执行是指在任意指定时间对您的函数代码的执行数量。您可以估计并发执行计数，但是，根据 Lambda 函数处理的事件是否来自基于流的事件源，并发执行计数会有所不同。

- 基于流的事件源 - 对于处理 Kinesis 或 DynamoDB 流的 Lambda 函数，分区数即为并发单位。如果您的流有 100 个活动分区，则最多会有 100 个 Lambda 函数调用并发运行。这是因为 Lambda 按顺序处理每个分片的事件。
- 并非基于流的事件源 - 如果您创建的 Lambda 函数处理的不是来自基于流的事件源的事件（例如，Lambda 可处理来自 Amazon S3 或 API 网关 等其他事件源的每个事件），则每个发布的事件就是一个工作单元（采用并行方式，最多可达您的账户限制）。因此，这些事件源发布的事件数（或请求数）影响并发度。您可以使用此公式来估算并发 Lambda 函数调用数：

```
events (or requests) per second * function duration
```

例如，考虑一个处理 Amazon S3 事件的 Lambda 函数。假定 Lambda 函数平均用时 3 秒，Amazon S3 每秒发布 10 个事件。因此，您的 Lambda 函数有 30 个并发执行。

请求速率

请求速率是指调用您的 Lambda 函数的速率。对于除了基于流的服务之外的所有服务，请求速率是事件源生成事件的速率。对于基于流的服务，AWS Lambda 按下面所示计算请求速率：

```
request rate = number of concurrent executions / function duration
```

例如，如果一个流上有 5 个活动分区（即，您有 5 个并行运行的 Lambda 函数），并且您的 Lambda 函数用时大概 2 秒，则请求速率为 2.5 个请求/秒。

扩展

AWS Lambda 将根据增加的流量动态扩展容量，具体取决于您的账户的[账户级别并发执行限制](#) (p. 350)。为了处理突增流量，Lambda 将立即根据预定量增加您的并发执行函数，具体取决于在哪个区域执行（请参阅下表）。

如果默认的并发立即增加量值（如下表所示）不足以适应流量的猛增，Lambda 将继续增加并发函数执行数量（每分钟 500 次），直达到到账户的安全限制，或并发执行的函数数量足以成功处理增加的负载。

Note

由于 Lambda 依赖 Amazon EC2 针对已启用 VPC 的 Lambda 函数提供弹性网络接口，这些函数在扩展时也会受制于 Amazon EC2 的速率限制。如果您的 Amazon EC2 速率限制使得已启用 VPC 的函数无法在每分钟内添加 500 个并发调用，请根据前一部分的说明请求提高限制：请求提高并发执行数限制。

超过此速率后，您的应用程序（即利用全部并发立即增加量的应用程序）应该可以通过客户端重试和回退处理 Amazon EC2 限制（502 EC2ThrottledException）。有关更多详情，请参阅 [AWS 中的错误重试和指数回退](#)。

下表按区域列出了并发立即增加量：

区域	并发立即增加量 (函数执行)
亚太区域（东京）	1000

区域	并发立即增加量 (函数执行)
亚太区域 (首尔)	500
亚太地区 (孟买)	500
亚太区域 (新加坡)	500
亚太区域 (悉尼)	500
中国 (北京)	500
加拿大 (中部)	500
欧洲 (法兰克福)	1000
欧洲 (伦敦)	500
欧洲 (爱尔兰)	3000
AWS GovCloud (US)	500
美国东部 (俄亥俄州)	500
美国西部 (加利福尼亚北部)	500
美国西部 (俄勒冈)	3000
美国东部 (弗吉尼亚北部)	3000
南美洲 (圣保罗)	500
AWS GovCloud (US)	500

要了解如何查看和管理您的函数的并发执行数，请参阅[管理并发 \(p. 350\)](#)

支持的事件源

本主题列出了支持的 AWS 服务，您可以将这些服务配置为 AWS Lambda 函数的事件源。在预配置事件源映射后，这些事件源检测事件时将自动调用您的 Lambda 函数。有关调用模式的更多信息，请参阅[事件源映射 \(p. 142\)](#)。

对于本主题中列出的所有事件源，请注意以下几点：

- 事件源将维护事件源映射，基于流的服务（Amazon Kinesis Data Streams 和 Amazon DynamoDB Streams）除外。对于基于流的服务，AWS Lambda 将维护事件源映射。AWS Lambda 为您提供[CreateEventSourceMapping \(p. 382\)](#) 操作以便您创建和管理事件源映射。有关更多信息，请参阅[事件源映射 \(p. 142\)](#)。
- 调用 Lambda 函数时这些事件源使用的调用类型也将进行预配置。例如，Amazon S3 始终异步调用 Lambda 函数，Amazon Cognito 将同步调用 Lambda 函数。仅在您使用[Invoke \(p. 425\)](#) 操作自行调用 Lambda 函数（例如，根据需要从自定义应用程序调用 Lambda 函数）时，可以控制调用类型。
- 为了处理 AWS 事件，您的 Lambda 函数可能需要包含附加库，具体取决于用于创建该函数的编程语言。用 Node.js 或 Python 编写的函数不需要任何附加库。对于 C#，您需要包括[AWS Lambda for .NET Core](#)。对于 Java，您需要包括[aws-lambda-java-libs](#)。

Important

每个包括的程序包都不需要修改就可以使用。移除依赖项、添加冲突依赖项或有选择地包括程序包中的类可能会导致意外行为。

您也可以根据需要调用 Lambda 函数。有关详细信息，请参阅 [其他事件源：根据需要调用 Lambda 函数 \(p. 154\)](#)。

有关这些事件源发布的事件的示例，请参阅 [事件源发布的示例事件 \(p. 154\)](#)。

主题

- [Amazon S3 \(p. 149\)](#)
- [Amazon DynamoDB \(p. 149\)](#)
- [Amazon Kinesis Data Streams \(p. 150\)](#)
- [Amazon Simple Notification Service \(p. 150\)](#)
- [Amazon Simple Email Service \(p. 150\)](#)
- [Amazon Cognito \(p. 151\)](#)
- [AWS CloudFormation \(p. 151\)](#)
- [Amazon CloudWatch Logs \(p. 151\)](#)
- [Amazon CloudWatch 事件 \(p. 151\)](#)
- [AWS CodeCommit \(p. 152\)](#)
- [计划的事件 \(由 Amazon CloudWatch 事件提供支持\) \(p. 152\)](#)
- [AWS Config \(p. 152\)](#)
- [Amazon Alexa \(p. 152\)](#)
- [Amazon Lex \(p. 153\)](#)
- [Amazon API Gateway \(p. 153\)](#)
- [AWS IoT 按钮 \(p. 153\)](#)
- [Amazon CloudFront \(p. 153\)](#)
- [Amazon Kinesis Data Firehose \(p. 154\)](#)
- [其他事件源：根据需要调用 Lambda 函数 \(p. 154\)](#)
- [事件源发布的示例事件 \(p. 154\)](#)

Amazon S3

您可以编写 Lambda 函数来处理 S3 存储桶事件，例如，对象创建事件或对象删除事件。例如，当用户将一张照片上传到存储桶时，您可能希望 Amazon S3 调用 Lambda 函数，以便读取图像和创建照片缩略图。

您可以使用 Amazon S3 中的存储桶通知配置来配置事件源映射，并标识您希望 Amazon S3 发布的存储桶事件以及要调用的 Lambda 函数。

有关示例 Amazon S3 事件，请参阅 [事件消息结构](#)、[Amazon S3 放置示例事件 \(p. 159\)](#) 和 [Amazon S3 删除示例事件 \(p. 160\)](#)。有关示例使用案例，请参阅 [配合使用 AWS Lambda 和 Amazon S3 \(p. 165\)](#)。

给定事件源的错误处理取决于 Lambda 的调用方式。Amazon S3 异步调用 Lambda 函数。有关如何重试错误的更多信息，请参阅 [了解重试行为 \(p. 146\)](#)。

Amazon DynamoDB

您可以使用 Lambda 函数作为 Amazon DynamoDB 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。要创建触发器，首先要为表启用 Amazon DynamoDB Streams。AWS Lambda 将轮询流，而 Lambda 函数将处理已发布到流的任何更新。

这是一个基于流的事件源。对于基于流的服务，您可以在 AWS Lambda 中创建事件源映射，并标识要轮询的流以及要调用的 Lambda 函数。

有关示例 DynamoDB 事件，请参阅[步骤 2.3.2：测试 Lambda 函数（手动调用）\(p. 198\)](#)和[Amazon DynamoDB 更新示例事件 \(p. 157\)](#)。有关一般格式，请参阅 Amazon DynamoDB API Reference 中的[GetRecord](#)。有关示例使用案例，请参阅[配合使用 AWS Lambda 和 Amazon DynamoDB \(p. 191\)](#)。

给定事件源的错误处理取决于 Lambda 的调用方式。DynamoDB 是一个基于流的事件源。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

Amazon Kinesis Data Streams

可以将 AWS Lambda 配置为自动轮询流并处理任何新记录，例如，网站点击流、财务交易记录、社交媒体源、IT 日志和位置跟踪事件。这样，AWS Lambda 会定期（每秒一次）轮询流中的新记录。

对于基于流的服务，您可以在 AWS Lambda 中创建事件源映射，并标识要轮询的流以及要调用的 Lambda 函数。

有关示例事件，请参阅[步骤 2.3：创建 Lambda 函数并手动对其进行测试 \(p. 187\)](#)和[Amazon Kinesis Data Streams 示例事件 \(p. 158\)](#)。有关示例使用案例，请参阅[配合使用 AWS Lambda 和 Kinesis \(p. 181\)](#)。

给定事件源的错误处理取决于 Lambda 的调用方式。Amazon Kinesis Data Streams 是一个基于流的事件源。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

Amazon Simple Notification Service

您可以编写 Lambda 函数来处理 Amazon Simple Notification Service 通知。在将消息发布到 Amazon SNS 主题时，服务可以通过将消息负载作为参数传递来调用 Lambda 函数。随后，您的 Lambda 函数代码可以处理事件，例如，将消息发布到其他 Amazon SNS 主题或将消息发送到其他 AWS 服务。

这还使您能够触发 Lambda 函数以响应 Amazon CloudWatch 警报和其他使用 Amazon SNS 的 AWS 服务。

可以通过主题订阅配置在 Amazon SNS 中配置事件源映射。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](#)。

有关示例活动，请参阅[附录：消息与 JSON 格式和 Amazon SNS 示例事件 \(p. 156\)](#)。有关示例使用案例，请参阅[将 AWS Lambda 与其他账户中的 Amazon SNS 结合使用 \(p. 217\)](#)。

当用户调用 Lambda 函数订阅的主题的 SNS Publish API 时，Amazon SNS 会调用 Lambda，进而异步调用函数。随后，Lambda 将返回传输状态。如果调用 Lambda 时出错，Amazon SNS 将重新尝试调用 Lambda 函数（最多尝试 3 次）。在尝试 3 次后，如果 Amazon SNS 仍无法成功调用 Lambda 函数，则 Amazon SNS 将向 CloudWatch 发送传输状态失败消息。

给定事件源的错误处理取决于 Lambda 的调用方式。Amazon SNS 异步调用 Lambda 函数。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

Amazon Simple Email Service

Amazon Simple Email Service (Amazon SES) 是一项经济高效的电子邮件服务。利用 Amazon SES，您可以发送电子邮件以及使用服务来接收消息。有关 Amazon SES 的更多信息，请参阅[Amazon Simple Email Service](#)。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。有关示例方案，请参阅[考虑 Amazon SES 电子邮件接收的使用案例](#)。

您可使用规则配置在 Amazon SES 中配置事件源映射。Amazon Simple Email Service 开发人员指南 中的以下主题提供额外的信息：

- 有关示例事件，请参阅[Lambda 操作和 Amazon SES 通过电子邮件接收示例事件 \(p. 155\)](#)。

- 有关 Lambda 函数示例，请参阅 [Lambda 函数示例](#)。

给定事件源的错误处理取决于 Lambda 的调用方式。Amazon SES 异步调用 Lambda 函数。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

Amazon Cognito

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，您可以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](#)。

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](#)。有关其他示例事件，请参阅 [Amazon Cognito 同步触发表示例事件 \(p. 158\)](#)

给定事件源的错误处理取决于 Lambda 的调用方式。Amazon Cognito 配置为同步调用 Lambda 函数。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

AWS CloudFormation

作为部署 AWS CloudFormation 堆栈的一部分，您可以将 Lambda 函数指定为自定义资源来执行任何自定义命令。通过将 Lambda 函数与自定义资源关联，您可以在创建、更新或删除 AWS CloudFormation 堆栈时调用 Lambda 函数。

您可使用堆栈定义在 AWS CloudFormation 中配置事件源映射。有关更多信息，请参阅 AWS CloudFormation 用户指南 中的 [AWS Lambda 支持的自定义资源](#)。

有关示例事件，请参阅[AWS CloudFormation 创建请求示例事件 \(p. 155\)](#)。请注意，此事件实际上是 Amazon SNS 事件中的 AWS CloudFormation 消息。

给定事件源的错误处理取决于 Lambda 的调用方式。AWS CloudFormation 同步调用 Lambda 函数。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

Amazon CloudWatch Logs

您可以使用 AWS Lambda 函数对使用 CloudWatch Logs 订阅的 Amazon CloudWatch Logs 进行自定义分析。利用 CloudWatch Logs 订阅，可以从 CloudWatch Logs 访问日志事件的实时源并将其传输到 AWS Lambda 函数，以对其进行自定义处理、分析或将其加载到其他系统。有关 CloudWatch Logs 的更多信息，请参阅[监控日志文件](#)。

您可使用日志订阅配置在 Amazon CloudWatch Logs 中保留事件源映射。有关更多信息，请参阅 Amazon CloudWatch 用户指南 中的[使用订阅实时处理日志数据 \(示例 2 : AWS Lambda\)](#)。

有关示例事件，请参阅[Amazon CloudWatch Logs 示例事件 \(p. 156\)](#)。

给定事件源的错误处理取决于 Lambda 的调用方式。Amazon CloudWatch Logs 异步调用您的 Lambda 函数（调用 Lambda 函数不会阻止对日志执行写入操作）。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

Amazon CloudWatch 事件

[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。

有关示例事件，请参阅 Amazon CloudWatch 用户指南 中的[支持事件类型](#)。

给定事件源的错误处理取决于 Lambda 的调用方式。Amazon CloudWatch 事件异步调用您的 Lambda 函数。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

AWS CodeCommit

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。有关更多信息，请参阅[管理 AWS CodeCommit 存储库的触发器](#)。

您可以使用存储库触发器来维护 AWS CodeCommit 中的事件源映射。有关更多信息，请参阅[PutRepositoryTriggers](#) 操作。

给定事件源的错误处理取决于 Lambda 的调用方式。AWS CodeCommit 异步调用您的 Lambda 函数。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

计划的事件（由 Amazon CloudWatch 事件提供支持）

您也可以使用 Amazon CloudWatch 事件中的计划事件功能将 AWS Lambda 设置为定期调用您的代码。要设置计划，您可以指定固定速率（小时数、天数或星期数）或指定 cron 表达式（请参阅 Amazon CloudWatch 用户指南 中的[规则的计划表达式语法](#)）。

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。

有关示例使用案例，请参阅[将 AWS Lambda 用于计划的事件 \(p. 249\)](#)。

有关示例事件，请参阅[计划的事件示例事件 \(p. 156\)](#)。

给定事件源的错误处理取决于 Lambda 的调用方式。Amazon CloudWatch 事件配置为异步调用 Lambda 函数。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

AWS Config

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。

您可使用规则目标定义在 AWS Config 中保留事件源映射。有关更多信息，请参阅 AWS Config API 参考 中的[PutConfigRule](#) 操作。

有关更多信息，请参阅[使用 AWS Config 规则评估资源](#)。有关设置自定义规则的示例，请参阅[为 AWS Config 开发自定义规则](#)。有关 Lambda 函数示例，请参阅[AWS Config 规则的示例 AWS Lambda 函数 \(Node.js\)](#)。

给定事件源的错误处理取决于 Lambda 的调用方式。AWS Config 配置为异步调用 Lambda 函数。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

Amazon Alexa

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。有关更多信息，请参阅：

- Alexa Skills Kit 入门。
- alexa-skills-kit-sdk-for-nodejs
- alexa-skills-kit-java

给定事件源的错误处理取决于 Lambda 的调用方式。Amazon Echo 配置为同步调用 Lambda 函数。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

Amazon Lex

Amazon Lex 是一项可在应用程序内使用语音和文本构建对话接口的 AWS 服务。Amazon Lex 提供与 AWS Lambda 的预构建集成，支持您创建 Lambda 函数，使其作为代码钩子与 Amazon Lex 机器人配合工作。在目标配置中，您可以标识 Lambda 函数以执行初始化/验证或履行操作，或者同时执行两者。

有关更多信息，请参阅[使用 Lambda 函数](#)。有关示例使用案例，请参阅[练习 1：使用蓝图创建 Amazon Lex 机器人](#)。

给定事件源的错误处理取决于 Lambda 的调用方式。Amazon Lex 配置为同步调用 Lambda 函数。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

Amazon API Gateway

您可以通过 HTTPS 调用 Lambda 函数。您可以使用 Amazon API Gateway 定义自定义 REST API 和终端节点来做到这一点。您可以将各个 API 操作（如 GET 和 PUT）映射到特定的 Lambda 函数。当向该 API 终端节点发送 HTTPS 请求时，Amazon API Gateway 服务会调用相应的 Lambda 函数。

有关更多信息，请参阅[同步调用 Lambda 函数](#)。有关示例使用案例，请参阅[将 AWS Lambda 与 Amazon API Gateway 结合使用（按需并通过 HTTPS）\(p. 222\)](#)。

给定事件源的错误处理取决于 Lambda 的调用方式。Amazon API Gateway 配置为同步调用 Lambda 函数。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

此外，您还可以将 Lambda 函数用于将数据发布到本主题中所列的某个支持的 AWS 事件源的其他 AWS 服务。例如，您可以：

- 触发 Lambda 函数以响应 CloudTrail 更新（因为它将所有 API 访问事件记录到 Amazon S3 存储桶中）。
- 触发 Lambda 函数以响应 CloudWatch 警报（因为它将警报事件发布到 Amazon SNS 主题）。

AWS IoT 按钮

AWS IoT 按钮是一个基于 Amazon Dash Button 硬件的可编程按钮。这种简单的 Wi-Fi 设备易于配置，专为帮助开发人员开始使用 AWS Lambda 以及众多其他 AWS 服务而设计，无需编写特定于设备的代码。

您可以在云中对按钮进行逻辑编码，配置按钮单击以计数或跟踪项目、呼叫或提醒某人、启动或停止某些操作、订购服务或甚至提供反馈。例如，您可以通过单击按钮开锁或启动汽车、打开车库门、叫计程车、呼叫配偶或客户服务代表、跟踪常见家务的进行情况及药物或商品的使用情况，或远程控制家用电器。

给定事件源的错误处理取决于 Lambda 的调用方式。AWS IoT 配置为异步调用 Lambda 函数。有关如何重试错误的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。

Amazon CloudFront

利用 Lambda@Edge，您可以在 AWS 区域和 Amazon CloudFront 边缘站点运行 Lambda 函数以响应 CloudFront 事件，而无需配置或管理服务器。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)

有关更多信息，请参阅 [AWS Lambda@Edge \(p. 261\)](#)。

给定事件源的错误处理取决于 Lambda 的调用方式。CloudFront 配置为同步调用 Lambda 函数。有关如何重试错误的更多信息，请参阅 [了解重试行为 \(p. 146\)](#)。

Amazon Kinesis Data Firehose

Amazon Kinesis Data Firehose 是将流数据加载到 AWS 中的最简单方法。它可以捕获、转换流数据并将其加载到下游服务 (例如 Kinesis Data Analytics 或 Amazon S3) 中，从而使您能够利用现成的业务智能工具和仪表板执行近实时分析。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。

给定事件源的错误处理取决于 Lambda 的调用方式。Kinesis Data Firehose 配置为同步调用 Lambda 函数。有关如何重试错误的更多信息，请参阅 [了解重试行为 \(p. 146\)](#)。

其他事件源：根据需要调用 Lambda 函数

除了使用事件源调用 Lambda 函数之外，您还可以根据需要调用 Lambda 函数。在此情况下，您无需预先配置任何事件源映射。不过，请确保自定义应用程序具有调用 Lambda 函数所需的权限。

例如，用户应用程序还可以生成事件（构建您自己的自定义事件源）。用户应用程序（如客户端、手机或 Web 应用程序）可以发布事件，并使用 AWS 软件开发工具包或 AWS 移动软件开发工具包（如适用于 Android 的 AWS 移动软件开发工具包）调用 Lambda 函数。

有关更多信息，请参阅 [用于 Amazon Web Services 的工具](#)。有关示例教程，请参阅 [将 AWS Lambda 与 Amazon API Gateway 结合使用（按需并通过 HTTPS）\(p. 222\)](#)。

事件源发布的示例事件

以下是支持的 AWS 服务所发布的示例事件的列表。有关受支持的 AWS 事件源的更多信息，请参阅 [支持的事件源 \(p. 148\)](#)。

示例事件

- [AWS CloudFormation 创建请求示例事件 \(p. 155\)](#)
- [Amazon SES 通过电子邮件接收示例事件 \(p. 155\)](#)
- [计划的事件示例事件 \(p. 156\)](#)
- [Amazon CloudWatch Logs 示例事件 \(p. 156\)](#)
- [Amazon SNS 示例事件 \(p. 156\)](#)
- [Amazon DynamoDB 更新示例事件 \(p. 157\)](#)
- [Amazon Cognito 同步触发表示例事件 \(p. 158\)](#)
- [Amazon Kinesis Data Streams 示例事件 \(p. 158\)](#)
- [Amazon S3 放置示例事件 \(p. 159\)](#)
- [Amazon S3 删除示例事件 \(p. 160\)](#)
- [Amazon Lex 示例事件 \(p. 160\)](#)

- API 网关 代理请求事件 (p. 161)
- API 网关 代理响应事件 (p. 162)
- CloudFront 事件 (p. 162)
- AWS Config 事件 (p. 163)
- AWS IoT 按钮事件 (p. 163)
- Kinesis Data Firehose 事件 (p. 163)

AWS CloudFormation 创建请求示例事件

```
{
  "StackId": "arn:aws:cloudformation:us-west-2:EXAMPLE/stack-name/guid",
  "ResponseURL": "http://pre-signed-S3-url-for-response",
  "ResourceProperties": {
    "StackName": "stack-name",
    "List": [
      "1",
      "2",
      "3"
    ]
  },
  "RequestType": "Create",
  "ResourceType": "Custom::TestResource",
  "RequestId": "unique id for this create request",
  "LogicalResourceId": "MyTestResource"
}
```

Amazon SES 通过电子邮件接收示例事件

```
{
  "Records": [
    {
      "EventVersion": "1.0",
      "EventSource": "aws:sns",
      "EventSubscriptionArn": "arn:aws:sns:us-west-2:123456789000:ses_messages:26a58451-3392-4ab6-a829-d65c2968421a",
      "Sns": {
        "MessageId": "483eae4c-4fb0-57e5-a5f9-ff9b08612bef",
        "Signature": "Uy3tn/qAQg/sXARGk2DRddd31ZtyDE+B1IzRla/KA75BaerApJqN+H59q69z8H+pRx0AyUwOD1K0huBYdDRbAMVOUsMgZgdcNjj0gSfFg8uZvTuKaqtawj4E0hmzoemHENWeuswuq316xoPcAJ9fHd2yFhX+792AV++i/8P4EKv/9t4j8Ejs3OxMRN49gkWefKbv4/avyH0dSaFTnXV0rGLmPb103dtjeY4K05PTKvUlPerN+MdRTvHrjApvqDvpONEVYyBU4zFZQ6GnFcFnHtTk44c3NH/dVi6Gf9VrX8V1id5VSZICYiIG1iaUz0b676IhRh8znzjMDWaczOBwkA==",
        "Type": "Notification",
        "TopicArn": "arn:aws:sns:us-west-2:123456789000:ses_messages",
        "MessageAttributes": {},
        "SignatureVersion": "1",
        "Timestamp": "2017-07-05T20:01:21.366Z",
        "SigningCertUrl": "https://sns.us-west-2.amazonaws.com/SimpleNotificationService-b95095beb82e8f6a046b3aafc7f4149a.pem",
        "Message": "{\"notificationType\":\"Delivery\",\"mail\":{\"timestamp\":\"2017-07-05T20:01:20.773Z\"},\"source\":\"jeff@amazon.com\", \"sourceArn\":\"arn:aws:ses:us-west-2:123456789000:identity/jeff@amazon.com\", \"sourceIp\":\"205.251.233.183\", \"sendingAccountId\":\"123456789000\", \"messageId\":\"0101015d1457bd85-2ff839b3-c119-4311-b90c-5ce39eff3026-000000\", \"destination\": [\"jeff@amazon.com\"]}, \"delivery\": {\"timestamp\": \"2017-07-05T20:01:21.302Z\", \"processingTimeMillis\": 529, \"recipients\": [\"jeff@amazon.com\"], \"smtpResponse\": \"250 ok: Message 122614849 accepted\", \"remoteMtaIp\": \"207.171.188.9\", \"reportingMTA\": \"a27-42.smtp-out.us-west-2.amazonses.com\"}}",
        "UnsubscribeUrl": "https://sns.us-west-2.amazonaws.com/?Action=Unsubscribe&eif=jccgiuhijfhrcunfnglreichbrcljrnltbeked"
    }
  ]
}
```

```
SubscriptionArn=arn:aws:sns:us-
west-2:123456789000:ses_messages:26a58451-3392-4ab6-a829-d65c2968421a",
    "Subject": null
}
]
}
```

计划的事件示例事件

```
{
    "account": "123456789012",
    "region": "us-east-1",
    "detail": {},
    "detail-type": "Scheduled Event",
    "source": "aws.events",
    "time": "1970-01-01T00:00:00Z",
    "id": "cdc73f9d-aea9-11e3-9d5a-835b769c0d9c",
    "resources": [
        "arn:aws:events:us-east-1:123456789012:rule/my-schedule"
    ]
}
```

Amazon CloudWatch Logs 示例事件

```
{
    "awslogs": {
        "data": "H4sIAAAAAAAAHWPwQqCQBCGX0Xm7EftK
+smZBEUgXoLCdMhFtKV3akI8d0bLYmibvPPN3wz00CJxmOnTO41whwWORIctmEcB6sQbFC3CjW3XW8kxpOpP
+OC22diWml1qZkOGtoMsScxaczKN3plG8zlaHIta5KqWsozoTYw3/djzwhpLwivWFGHGpAFe7DL68JlBUk
+l7KSN7tCOEJ4M3/qOI49vMHj+zCKdlFqLaU2ZHV2a4Ct/an0/ivdX8oYc1UVX860fQDQiMdxRQEAAA=="
    }
}
```

Amazon SNS 示例事件

```
{
    "Records": [
        {
            "EventVersion": "1.0",
            "EventSubscriptionArn": eventsubscriptionarn,
            "EventSource": "aws:sns",
            "Sns": {
                "SignatureVersion": "1",
                "Timestamp": "1970-01-01T00:00:00.000Z",
                "Signature": "EXAMPLE",
                "SigningCertUrl": "EXAMPLE",
                "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
                "Message": "Hello from SNS!",
                "MessageAttributes": {
                    "Test": {
                        "Type": "String",
                        "Value": "TestString"
                    },
                    "TestBinary": {
                        "Type": "Binary",
                        "Value": "TestBinary"
                    }
                },
                "Type": "Notification",
                "UnsubscribeUrl": "EXAMPLE",
                "TopicArn": topicarn,
                "MessageMD5": "EXAMPLE"
            }
        }
    ]
}
```

```
        "Subject": "TestInvoke"
    }
}
]
```

Amazon DynamoDB 更新示例事件

```
{
  "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": "eventsourcesarn",
      "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"
},
{
  "eventID": "3",
  "eventVersion": "1.0",
  "dynamodb": {
    "Keys": {
      "Id": {
        "N": "101"
      }
    },
    "SizeBytes": 38,
    "SequenceNumber": "333",
    "OldImage": {
      "Message": {
        "S": "This item has changed"
      },
      "Id": {
        "N": "101"
      }
    },
    "StreamViewType": "NEW_AND_OLD_IMAGES"
  },
  "awsRegion": "us-west-2",
  "eventName": "REMOVE",
  "eventSourceARN": sourcearn,
  "eventSource": "aws:dynamodb"
}
]
```

Amazon Cognito 同步触发示例事件

```
{
  "datasetName": "datasetName",
  "eventType": "SyncTrigger",
  "region": "us-east-1",
  "identityId": "identityId",
  "datasetRecords": [
    "SampleKey2": {
      "newValue": "newValue2",
      "oldValue": "oldValue2",
      "op": "replace"
    },
    "SampleKey1": {
      "newValue": "newValue1",
      "oldValue": "oldValue1",
      "op": "replace"
    }
  ],
  "identityPoolId": "identityPoolId",
  "version": 2
}
```

Amazon Kinesis Data Streams 示例事件

```
"Records": [
  {
    "eventID": "shardId-000000000000:49545115243490985018280067714973144582180062593244200961",
    "eventVersion": "1.0",
    "kinesis": {
      "partitionKey": "partitionKey-3",
      "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0IDEyMy4=",
      "kinesisSchemaVersion": "1.0",
      "sequenceNumber": "49545115243490985018280067714973144582180062593244200961"
    },
    "invokeIdentityArn": identityarn,
    "eventName": "aws:kinesis:record",
    "eventSourceARN": eventsourcearn,
    "eventSource": "aws:kinesis",
    "awsRegion": "us-east-1"
  }
]
```

Amazon S3 放置示例事件

```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventTime": "1970-01-01T00:00:00.000Z",
      "requestParameters": {
        "sourceIPAddress": "127.0.0.1"
      },
      "s3": {
        "configurationId": "testConfigRule",
        "object": {
          "eTag": "0123456789abcdef0123456789abcdef",
          "sequencer": "0A1B2C3D4E5F678901",
          "key": "HappyFace.jpg",
          "size": 1024
        },
        "bucket": {
          "arn": bucketarn,
          "name": "sourcebucket",
          "ownerIdentity": {
            "principalId": "EXAMPLE"
          }
        },
        "s3SchemaVersion": "1.0"
      },
      "responseElements": {
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklmabaisawesome/",
        "x-amz-request-id": "EXAMPLE123456789"
      },
      "awsRegion": "us-east-1",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "eventSource": "aws:s3"
    }
  ]
}
```

Amazon S3 删除示例事件

```
{  
  "Records": [  
    {  
      "eventVersion": "2.0",  
      "eventTime": "1970-01-01T00:00:00.000Z",  
      "requestParameters": {  
        "sourceIPAddress": "127.0.0.1"  
      },  
      "s3": {  
        "configurationId": "testConfigRule",  
        "object": {  
          "sequencer": "0A1B2C3D4E5F678901",  
          "key": "HappyFace.jpg"  
        },  
        "bucket": {  
          "arn": bucketarn,  
          "name": "sourcebucket",  
          "ownerIdentity": {  
            "principalId": "EXAMPLE"  
          }  
        },  
        "s3SchemaVersion": "1.0"  
      },  
      "responseElements": {  
        "x-amz-id-2": "EXAMPLE123/5678abcdefghijklmnaaaaaaaaaaaaaaaa/  
mnopqrstuvwxyzABCDEFGHIJKLMN",  
        "x-amz-request-id": "EXAMPLE123456789"  
      },  
      "awsRegion": "us-east-1",  
      "eventName": "ObjectRemoved:Delete",  
      "userIdentity": {  
        "principalId": "EXAMPLE"  
      },  
      "eventSource": "aws:s3"  
    }  
  ]  
}
```

Amazon Lex 示例事件

```
{  
  "messageVersion": "1.0",  
  "invocationSource": "FulfillmentCodeHook or DialogCodeHook",  
  "userId": "user-id specified in the POST request to Amazon Lex.",  
  "sessionAttributes": {  
    "key1": "value1",  
    "key2": "value2",  
  },  
  "bot": {  
    "name": "bot-name",  
    "alias": "bot-alias",  
    "version": "bot-version"  
  },  
  "outputDialogMode": "Text or Voice, based on ContentType request header in runtime API request",  
  "currentIntent": {  
    "name": "intent-name",  
    "slots": {  
      "slot-name": "value",  
    }  
  }  
}
```

```

        "slot-name": "value",
        "slot-name": "value"
    },
    "confirmationStatus": "None, Confirmed, or Denied
        (intent confirmation, if configured)"
}
}

```

API 网关 代理请求事件

```

{
    "path": "/test/hello",
    "headers": {
        "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*
*q=0.8",
        "Accept-Encoding": "gzip, deflate, lzma, sdch, br",
        "Accept-Language": "en-US,en;q=0.8",
        "CloudFront-Forwarded-Proto": "https",
        "CloudFront-Is-Desktop-Viewer": "true",
        "CloudFront-Is-Mobile-Viewer": "false",
        "CloudFront-Is-SmartTV-Viewer": "false",
        "CloudFront-Is-Tablet-Viewer": "false",
        "CloudFront-Viewer-Country": "US",
        "Host": "wt6mne2s9k.execute-api.us-west-2.amazonaws.com",
        "Upgrade-Insecure-Requests": "1",
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36 OPR/39.0.2256.48",
        "Via": "1.1 fb7cca60f0ecd82ce07790c9c5eef16c.cloudfront.net (CloudFront)",
        "X-Amz-Cf-Id": "nBsWBOrSHMgnaROZJK1wGCZ9PcRcSpq_oSXZNQwQ100TzL4cimZo3g==",
        "X-Forwarded-For": "192.168.100.1, 192.168.1.1",
        "X-Forwarded-Port": "443",
        "X-Forwarded-Proto": "https"
    },
    "pathParameters": {
        "proxy": "hello"
    },
    "requestContext": {
        "accountId": "123456789012",
        "resourceId": "us4z18",
        "stage": "test",
        "requestId": "41b45ea3-70b5-11e6-b7bd-69b5aaebc7d9",
        "identity": {
            "cognitoIdentityPoolId": "",
            "accountId": "",
            "cognitoIdentityId": "",
            "caller": "",
            "apiKey": "",
            "sourceIp": "192.168.100.1",
            "cognitoAuthenticationType": "",
            "cognitoAuthenticationProvider": "",
            "userArn": "",
            "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36 OPR/39.0.2256.48",
            "user": ""
        },
        "resourcePath": "/{proxy+}",
        "httpMethod": "GET",
        "apiId": "wt6mne2s9k"
    },
    "resource": "/{proxy+}",
    "httpMethod": "GET",
    "queryStringParameters": {
        "name": "me"
    }
},

```

```
    "stageVariables": {
        "stageVarName": "stageVarValue"
    }
}
```

API 网关 代理响应事件

```
{
    "statusCode": 200,
    "headers": {
        "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8",
        "Accept-Encoding": "gzip, deflate, lzma, sdch, br",
        "Accept-Language": "en-US,en;q=0.8",
        "CloudFront-Forwarded-Proto": "https",
        "CloudFront-Is-Desktop-Viewer": "true",
        "CloudFront-Is-Mobile-Viewer": "false",
        "CloudFront-Is-SmartTV-Viewer": "false",
        "CloudFront-Is-Tablet-Viewer": "false",
        "CloudFront-Viewer-Country": "US",
        "Host": "wt6mne2s9k.execute-api.us-west-2.amazonaws.com",
        "Upgrade-Insecure-Requests": "1",
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36 OPR/39.0.2256.48",
        "Via": "1.1 fb7cca60f0ecd82ce07790c9c5eef16c.cloudfront.net (CloudFront)",
        "X-Amz-Cf-Id": "nBsWBOrSHMgnAROZJK1wGCZ9PcRcSpq_oSXZNQwQ10OTZL4cimZo3g==",
        "X-Forwarded-For": "192.168.100.1, 192.168.1.1",
        "X-Forwarded-Port": "443",
        "X-Forwarded-Proto": "https"
    },
    "body": "Hello World"
}
```

CloudFront 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"
                            }
                        ]
                    }
                }
            }
        }
    ]
}
```

```
        ]  
    }
```

AWS Config 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\"\n}
```

AWS IoT 按钮事件

```
{  
  \"serialNumber\": \"ABCDEFG12345\",  
  \"clickType\": \"SINGLE\",  
  \"batteryVoltage\": \"2000 mV\"\n}
```

Kinesis Data Firehose 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\": 1510772160000,  
      \"kinesisRecordMetadata\": {  
        \"shardId\": \"shardId-000000000001\",  
        \"partitionKey\": \"4d1ad2b9-24f8-4b9d-a088-76e9947c317b\",  
        \"approximateArrivalTimestamp\": \"2012-04-23T18:25:43.512Z\",  
        \"sequenceNumber\": \"49546986683135544286507457936321625675700192471156785155\",  
        \"subsequenceNumber\": ""  
      }  
    }  
  ]  
}
```

```
"kinesisRecordMetadata": {  
    "shardId": "shardId-000000000001",  
    "partitionKey": "4d1ad2b9-24f8-4b9d-a088-76e9947c318a",  
    "approximateArrivalTimestamp": "2012-04-23T19:25:43.511Z",  
    "sequenceNumber": "49546986683135544286507457936321625675700192471156785155",  
    "subsequenceNumber": ""  
}  
}  
]  
}
```

如何使用 AWS Lambda 的示例

AWS Lambda 的使用案例可分为以下类别：

- 将 AWS Lambda 作为事件源用于 AWS 服务 - 事件源发布导致调用 Lambda 函数的事件。这些事件源可以是 AWS 服务，如 Amazon S3。有关更多信息和教程，请参阅以下主题：

[配合使用 AWS Lambda 和 Amazon S3 \(p. 165\)](#)

[配合使用 AWS Lambda 和 Kinesis \(p. 181\)](#)

[配合使用 AWS Lambda 和 Amazon DynamoDB \(p. 191\)](#)

[配合使用 AWS Lambda 和 AWS CloudTrail \(p. 203\)](#)

[将 AWS Lambda 与其他账户中的 Amazon SNS 结合使用 \(p. 217\)](#)

- 通过 HTTPS (Amazon API Gateway) 实现的按需 Lambda 函数调用 - 除了使用事件源调用 Lambda 函数以外，您还可以通过 HTTPS 调用 Lambda 函数。您可以使用 API 网关 定义自定义 REST API 和终端节点来做到这一点。有关更多信息和教程，请参阅[将 AWS Lambda 与 Amazon API Gateway 结合使用 \(按需并通过 HTTPS\) \(p. 222\)](#)。
- 按需 Lambda 函数调用 (使用自定义应用程序构建您自己的事件源) - 用户应用程序 (如客户端应用程序、移动应用程序或 Web 应用程序) 可以发布事件并使用 AWS 软件开发工具包或 AWS 移动软件开发工具包 (如适用于 Android 的 AWS 移动软件开发工具包) 调用 Lambda 函数。有关更多信息和教程，请参阅[入门 \(p. 3\)](#)和[使用 AWS Lambda 作为移动应用程序后端 \(自定义事件源 : Android\) \(p. 237\)](#)
- 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。有关更多信息和教程，请参阅[将 AWS Lambda 用于计划的事件 \(p. 249\)](#)。

此外，您还可以使用 Lambda 状态机。有关更多信息，请参阅[使用状态机](#)。

配合使用 AWS Lambda 和 Amazon S3

Amazon S3 可将事件 (例如，在存储桶中创建对象时) 发布到 AWS Lambda 并通过将事件数据作为参数传递来调用您的 Lambda 函数。利用这种集成，您可以编写处理 Amazon S3 事件的 Lambda 函数。在 Amazon S3 中，您可添加存储桶通知配置，该配置可标识您希望 Amazon S3 发布的事件类型和您希望调用的 Lambda 函数。

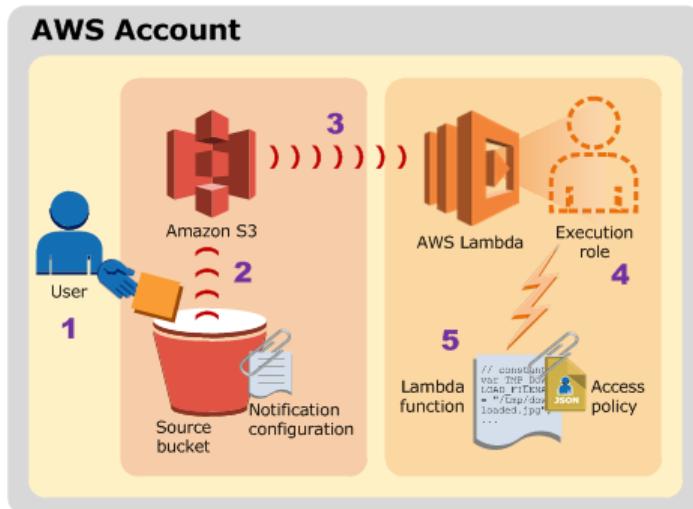
请注意有关 Amazon S3 和 AWS Lambda 集成的工作原理的以下信息：

- 非基于流 (异步) 的模型 - 这是一个模型 (请参阅[事件源映射 \(p. 142\)](#))，其中 Amazon S3 监控存储桶并通过将事件数据作为参数传递来调用 Lambda 函数。在推模型中，您将使用存储桶通知配置保留 Amazon S3 中的事件源映射。在此配置中，您应指定您希望 Amazon S3 监控的事件类型和您希望 Amazon S3 调用的 AWS Lambda 函数。有关更多信息，请参阅 Amazon Simple Storage Service 开发人员指南 中的[配置 Amazon S3 事件通知](#)。
- 异步调用 - AWS Lambda 使用 Event 调用类型 (异步调用) 来调用 Lambda 函数。有关调用类型的更多信息，请参阅[调用类型 \(p. 142\)](#)。
- 事件结构 - 您的 Lambda 函数接收的事件针对单个对象并提供了存储桶名称和对象键名称等信息。

请注意，设置端到端体验时有两种类型的权限策略可供选择：

- 针对 Lambda 函数的权限 - 无论哪个对象调用 Lambda 函数，AWS Lambda 都将通过代入您在创建 Lambda 函数时指定的 IAM 角色（执行角色）来执行该函数。利用与此角色关联的权限策略，您可以向 Lambda 函数授予其所需的权限。例如，如果 Lambda 函数需要读取某个对象，您可以在权限策略中为相关 Amazon S3 操作授予权限。有关更多信息，请参阅 [管理权限：使用 IAM 角色（执行角色）\(p. 339\)](#)。
- 供 Amazon S3 调用 Lambda 函数的权限 - Amazon S3 无法在未经您的许可的情况下调用 Lambda 函数。您通过与 Lambda 函数关联的权限策略授予此权限。

下图概述了该流程：



1. 用户将对象上传到 S3 存储桶（对象创建事件）。
2. Amazon S3 检测到对象创建事件。
3. Amazon S3 调用在存储桶通知配置中指定的 Lambda 函数。
4. 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。
5. Lambda 函数执行。

有关引导您完成示例设置的教程，请参阅[教程：将 AWS Lambda 与 Amazon S3 结合使用 \(p. 166\)](#)。

教程：将 AWS Lambda 与 Amazon S3 结合使用

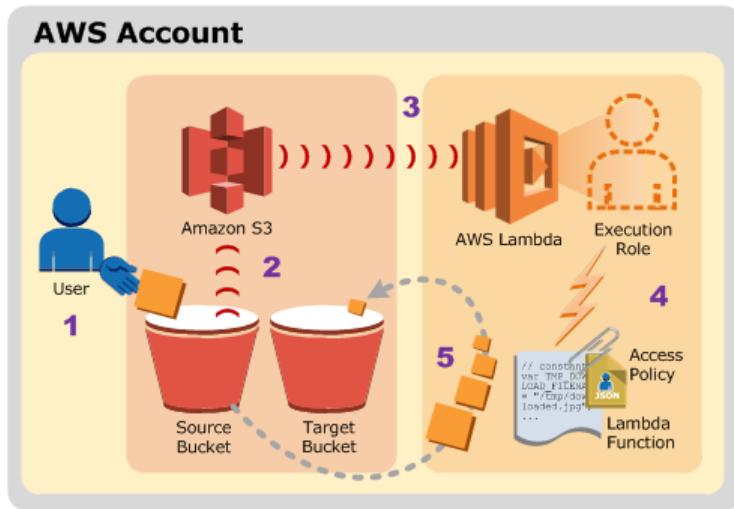
假设您要为将上传到存储桶的每个图像 (.jpg 和 .png 对象) 创建一个缩略图。您可以创建一个 Lambda 函数 (CreateThumbnail)，在创建对象后，Amazon S3 可调用该函数。然后，Lambda 函数可以从 **source** 存储桶中读取图像对象并创建缩略图目标存储桶 (在本教程中，该存储桶称为 **source resized** 存储桶)。

Important

以下教程假定您使用的是两个存储桶，一个用于源，一个用于目标。如果使用同一个存储桶作为源和目标，上传到源存储桶的每个缩略图都会触发另一个对象创建事件，该事件将再次调用 Lambda 函数，从而产生不必要的递归。但您可以在 **source** 存储桶中创建一个文件夹，以便为目标创建唯一的终端节点。如果您选择执行此操作，请更新下面的示例。

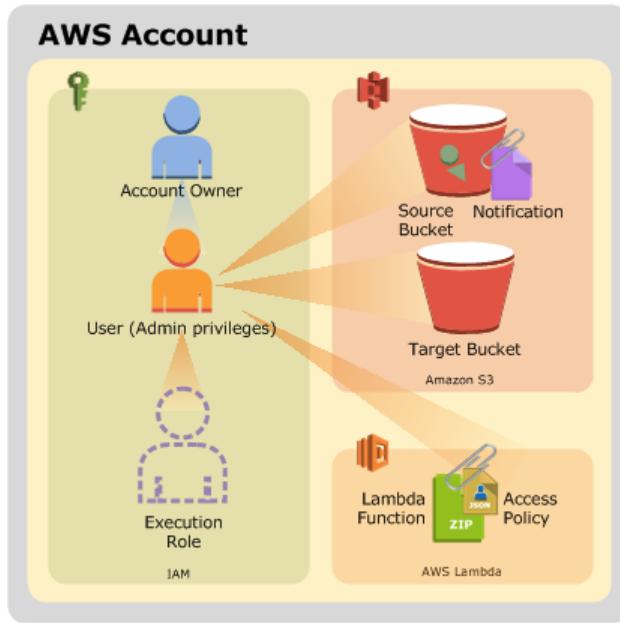
实现摘要

下图说明应用程序的流程：



1. 用户将对象上传到 Amazon S3 中的源存储桶（对象创建事件）。
2. Amazon S3 检测到对象创建事件。
3. Amazon S3 调用 Lambda 函数并将事件数据作为函数传递，由此将 s3:ObjectCreated:* 事件发布到 AWS Lambda。
4. 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。
5. Lambda 函数通过收到的事件数据获得了源存储桶名称和对象键名称。Lambda 函数读取该对象，使用图形库创建缩略图，然后将其保存到目标存储桶。

请注意，完成本教程后，您的账户中将具有以下 Amazon S3、Lambda 和 IAM 资源：



在 Lambda 中：

- Lambda 函数。
- 与 Lambda 函数关联的访问权限策略 - 使用此权限策略向 Amazon S3 授予调用 Lambda 函数的权限。此外，您还将限制该权限，以使 Amazon S3 只能针对来自特定存储桶（归特定 AWS 账户所有）的对象创建事件调用 Lambda 函数。

Note

某个 AWS 账户删除存储桶后，其他 AWS 账户可以创建使用该相同名称的存储桶。额外的条件可确保：仅当 Amazon S3 检测到来自特定存储桶（归特定 AWS 账户所有）的对象创建事件时，Amazon S3 才能调用 Lambda 函数。

在 IAM 中：

- 管理员用户 - 称作 adminuser。不建议使用 AWS 账户的根凭证。相反，请使用 adminuser 凭证执行本教程中的步骤。

Note

如果您尚未创建 adminuser 配置文件，请参阅[设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)。

- IAM 角色 (执行角色) - 通过与此角色关联的权限策略授予 Lambda 函数所需的权限。

在 Amazon S3 中：

- 两个存储桶名为 `source` 和 `source resized`。请注意，`source` 是一个占位符名称，您需要将它替换为您的实际存储桶名称。例如，如果您将名为 example 存储桶作为源，您将创建 exempleresized 作为目标存储桶。
- 源存储桶上的通知配置 - 您将在源存储桶上添加通知配置，用来标识您希望 Amazon S3 发布到 AWS Lambda 的事件的类型（对象创建事件）和要调用的 Lambda 函数。有关 Amazon S3 通知功能的更多信息，请参阅 Amazon Simple Storage Service 开发人员指南中的[设置存储桶事件的通知](#)。

您现在可以开始教程。请注意，最初的准备工作完成后，本教程分为了两个主要部分：

- 首先，完成创建 Lambda 函数的必要设置步骤，并使用 Amazon S3 示例事件数据手动调用该函数。该中间测试旨在验证函数能够正常工作。
- 其次，向源存储桶添加通知配置，以便 Amazon S3 在检测到对象创建事件时能够调用 Lambda 函数。

下一步

[步骤 1：准备 \(p. 168\)](#)

步骤 1：准备

请在此部分中执行以下操作：

- 注册 AWS 账户并设置 AWS CLI。
- 创建两个存储桶（`source` 和 `source resized` 存储桶），且源存储桶中包含示例 .jpg 对象 (`HappyFace.jpg`)。有关说明，请参阅以下过程。

步骤 1.1：注册 AWS 并设置 AWS CLI

确保您已完成以下步骤：

- 注册 AWS 账户并在该账户中创建管理员用户（名为 adminuser）。有关说明，请参阅[设置 AWS 账户 \(p. 4\)](#)。
- 安装并设置 AWS CLI。有关说明，请参阅[设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)。

步骤 1.2：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。

Important

源存储桶和 Lambda 函数必须位于同一个 AWS 区域内。此外，用于 Lambda 函数的示例代码还假定这两个存储桶位于同一个区域内。在本教程中，我们将使用 us-west-2 区域。

1. 使用 IAM 用户登录 URL，以 adminuser 身份登录 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 mybucketresized。
有关说明，请参阅 Amazon Simple Storage Service 入门指南 中的[创建存储桶](#)。
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。

下一步

[步骤 2：创建 Lambda 函数并手动调用该函数（使用示例事件数据）\(p. 169\)](#)

步骤 2：创建 Lambda 函数并手动调用该函数（使用示例事件数据）

请在此部分中执行以下操作：

- 使用提供的示例代码创建 Lambda 函数部署程序包。

Note

要查看在您的函数中使用其他 AWS 服务的更多示例，包括调用其他 Lambda 函数，请参阅 [AWS SDK for JavaScript](#)

- 创建 IAM 角色（执行角色）。在上传部署程序包时，需要指定 Lambda 可代入的代表您执行该函数的 IAM 角色（执行角色）。
- 通过上传部署程序包创建 Lambda 函数，然后使用示例 Amazon S3 事件数据手动调用该函数以便对其进行测试。

主题

- [步骤 2.1：创建部署程序包 \(p. 169\)](#)
- [步骤 2.2：创建执行角色（IAM 角色）\(p. 176\)](#)
- [步骤 2.3：创建 Lambda 函数并手动对其进行测试 \(p. 177\)](#)

步骤 2.1：创建部署程序包

从 Filter View 列表中，选择要用于 Lambda 函数的语言。适当的部分随即出现，其中包括创建部署程序包的代码和具体说明。

Node.js

部署程序包是包含 Lambda 函数代码和依赖项的 .zip 文件。

1. 创建文件夹 (`examplefolder`)，然后创建子文件夹 (`node_modules`)。
2. 安装 Node.js 平台。有关更多信息，请参阅 [Node.js 网站](#)。
3. 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
 - gm , GraphicsMagick for node.js
 - Async 实用程序模块

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，因此您只需安装其他库。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。

```
npm install async gm
```

4. 打开文本编辑器，然后复制以下代码。

```
// dependencies
var async = require('async');
var AWS = require('aws-sdk');
var gm = require('gm')
    .subClass({ imageMagick: true }); // Enable ImageMagick integration.
var util = require('util');

// constants
var MAX_WIDTH  = 100;
var MAX_HEIGHT = 100;

// get reference to S3 client
var s3 = new AWS.S3();

exports.handler = function(event, context, callback) {
    // Read options from the event.
    console.log("Reading options from event:\n", util.inspect(event, {depth: 5}));
    var srcBucket = event.Records[0].s3.bucket.name;
    // Object key may have spaces or unicode non-ASCII characters.
    var srcKey     =
        decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));
    var dstBucket = srcBucket + "resized";
    var dstKey    = "resized-" + srcKey;

    // Sanity check: validate that source and destination are different buckets.
    if (srcBucket == dstBucket) {
        callback("Source and destination buckets are the same.");
        return;
    }

    // Infer the image type.
    var typeMatch = srcKey.match(/^.([^.]*$)/);
    if (!typeMatch) {
        callback("Could not determine the image type.");
        return;
    }
    var imageType = typeMatch[1];
    if (imageType != "jpg" && imageType != "png") {
        callback('Unsupported image type: ${imageType}');
        return;
    }

    // Download the image from S3, transform, and upload to a different S3 bucket.
    async.waterfall([
        function download(next) {
```

```
// Download the image from S3 into a buffer.
s3.getObject({
    Bucket: srcBucket,
    Key: srcKey
},
next);
},
function transform(response, next) {
    gm(response.Body).size(function(err, size) {
        // Infer the scaling factor to avoid stretching the image unnaturally.
        var scalingFactor = Math.min(
            MAX_WIDTH / size.width,
            MAX_HEIGHT / size.height
        );
        var width = scalingFactor * size.width;
        var height = scalingFactor * size.height;

        // Transform the image buffer in memory.
        this.resize(width, height)
            .toBuffer(imageType, function(err, buffer) {
                if (err) {
                    next(err);
                } else {
                    next(null, response.ContentType, buffer);
                }
            });
    },
    function upload(contentType, data, next) {
        // Stream the transformed image to a different S3 bucket.
        s3.putObject({
            Bucket: dstBucket,
            Key: dstKey,
            Body: data,
            ContentType: contentType
        },
        next);
    }
], function (err) {
    if (err) {
        console.error(
            'Unable to resize ' + srcBucket + '/' + srcKey +
            ' and upload to ' + dstBucket + '/' + dstKey +
            ' due to an error: ' + err
        );
    } else {
        console.log(
            'Successfully resized ' + srcBucket + '/' + srcKey +
            ' and uploaded to ' + dstBucket + '/' + dstKey
        );
    }
    callback(null, "message");
}
);
});
```

Note

代码示例与 Node.js 运行时 v6.10 或 v4.3 兼容。有关更多信息，请参阅[编程模型 \(Node.js\) \(p. 18\)](#)

5. 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 `examplebucketresized`。
 - 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。
6. 将该文件保存到 `CreateThumbnail.js` 中，文件名为 `examplefolder`。完成此步骤后，文件夹结构如下：

```
CreateThumbnail.js
/node_modules/gm
/node_modules/async
```

7. 将 `CreateThumbnail.js` 文件和 `node_modules` 文件夹压缩为 `CreateThumbnail.zip`。

此即 Lambda 函数部署程序包。

下一步

步骤 2.2：创建执行角色（IAM 角色）(p. 176)

Java

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.S3EventProcessorCreateThumbnail`）。有关使用接口提供处理程序的更多信息，请参阅[利用预定义接口创建处理程序（Java）\(p. 39\)](#)。

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。

```
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.net.URLDecoder;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import javax.imageio.ImageIO;

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.AmazonS3Client;
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;
```

```
public class S3EventProcessorCreateThumbnail 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().getKey()
                .replace('+', ' ');
            srcKey = URLDecoder.decode(srcKey, "UTF-8");

            String dstBucket = srcBucket + "resized";
            String dstKey = "resized-" + srcKey;

            // Sanity check: validate that source and destination are different
            // buckets.
            if (srcBucket.equals(dstBucket)) {
                System.out
                    .println("Destination bucket must not match source bucket.");
                return "";
            }

            // Infer the image type.
            Matcher matcher = Pattern.compile(".*\\\\.([^\\\\.]*)").matcher(srcKey);
            if (!matcher.matches()) {
                System.out.println("Unable to infer image type for key "
                    + srcKey);
                return "";
            }
            String imageType = matcher.group(1);
            if (!(JPG_TYPE.equals(imageType)) && !(PNG_TYPE.equals(imageType))) {
                System.out.println("Skipping non-image " + srcKey);
                return "";
            }

            // Download the image from S3 into a stream
            AmazonS3 s3Client = new AmazonS3Client();
            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
        // If you want higher quality algorithms, check this link:
        // https://today.java.net/pub/a/today/2007/04/03/perils-of-image-
getscaledinstance.html
        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);
        s3Client.putObject(dstBucket, dstKey, is, meta);
        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。

使用之前的代码 (在名为 S3EventProcessorCreateThumbnail.java 的文件中) 创建一个部署程序包。确保添加以下依赖项：

- aws-lambda-java-core
- aws-lambda-java-events

这些内容可以在 [aws-lambda-java-libs](#) 中找到。

有关更多信息，请参阅 [使用 Java 编写 Lambda 函数的编程模型 \(p. 33\)](#)。

部署程序包既可以是 .zip 文件，也可以是独立的 .jar。您可以使用自己熟悉的任何构建和打包工具来创建部署程序包。有关如何使用 Maven 构建工具创建独立 .jar 的示例，请参阅[使用 Maven 但不借助任何 IDE 创建 .jar 部署程序包 \(Java\) \(p. 88\)](#)和[使用 Maven 和 Eclipse IDE 创建 .jar 部署程序包 \(Java\) \(p. 90\)](#)。有关如何使用 Gradle 构建工具创建 .zip 文件的示例，请参阅[创建 .zip 部署程序包 \(Java\) \(p. 92\)](#)。

确认已创建您的部署程序包后，请转到下一步来创建 IAM 角色（执行角色）。您将在创建 Lambda 函数时指定此角色。

[下一步](#)

[步骤 2.2：创建执行角色（IAM 角色）\(p. 176\)](#)

Python

在本节中，您将创建示例 Python 函数并安装依赖项。此代码示例与 Python 运行时版本 3.6 或 2.7 兼容。这些步骤适用于 3.6 运行时，但您也可以使用另一版本。

1. 打开文本编辑器，并复制以下代码。该代码使用相同的图像名称将调整大小后的图像上传到其他存储桶，如下所示：

```
source-bucket/image.png -> source-bucketresized/image.png
```

Note

利用 `from __future__ import print_function` 语句，可以编写与 Python 2 或 3 兼容的代码。如果您使用的是运行时版本 3.6，则不必将其包含在内。

```
from __future__ import print_function
import boto3
import os
import sys
import uuid
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 handler(event, context):
    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']
        download_path = '/tmp/{}{}'.format(uuid.uuid4(), key)
        upload_path = '/tmp/resized-{}'.format(key)

        s3_client.download_file(bucket, key, download_path)
        resize_image(download_path, upload_path)
        s3_client.upload_file(upload_path, '{}resized'.format(bucket), key)
```

2. 将该文件保存为 `CreateThumbnail.py`。
3. 如果您的源代码在本地主机上，请复制它。

```
scp -i key.pem /path/to/my_code.py ec2-user@public-ip-address:~/CreateThumbnail.py
```

4. 通过 SSH 连接到 64 位 Amazon Linux 实例。

```
ssh -i key.pem ec2-user@public-ip-address
```

5. 使用以下步骤安装 Python 3.6 和 virtualenv：

```
1. sudo yum install -y gcc zlib zlib-devel openssl openssl-devel
2. wget https://www.python.org/ftp/python/3.6.1/Python-3.6.1.tgz
3. tar -xzvf Python-3.6.1.tgz
4. cd Python-3.6.1 && ./configure && make
5. sudo make install
6. sudo /usr/local/bin/pip3 install virtualenv
6. 选择通过 pip3 安装的虚拟环境
```

```
/usr/local/bin/virtualenv ~/shrink_venv
```

```
source ~/shrink_venv/bin/activate
```

7. 在虚拟环境中安装库

```
pip install Pillow
```

```
pip install boto3
```

Note

AWS Lambda 包含适用于 Python 的 AWS 软件开发工具包 (Boto 3)，因此您无需将它包含在部署程序包中，但可以选择包含它以用于本地测试。

8. 将 lib 和 lib64 站点程序包的内容添加到 .zip 文件中。请注意，以下步骤假定您使用 Python 运行时版本 3.6。如果您使用版本 2.7，需要进行相应更新。

```
cd $VIRTUAL_ENV/lib/python3.6/site-packages
```

```
zip -r9 ~/CreateThumbnail.zip *
```

Note

要包括所有隐藏的文件，请使用以下选项：`zip -r9 ~/CreateThumbnail.zip`

9. 添加 Python 代码到 .zip 文件

```
cd ~
```

```
zip -g CreateThumbnail.zip CreateThumbnail.py
```

[下一步](#)

[步骤 2.2：创建执行角色（IAM 角色）\(p. 176\)](#)

步骤 2.2：创建执行角色（IAM 角色）

在本节中，您将使用以下预定义的角色类型和访问权限策略创建 IAM 角色：

- AWS Lambda 类型的 AWS 服务角色 - 此角色为 AWS Lambda 授予代入 IAM 角色的权限。
- 您附加到 IAM 角色的 AWSLambdaExecute 访问权限策略。
- 添加一个自定义策略，它为您分配将对象添加到 Amazon S3 存储桶的权限。有关创建 IAM 角色（执行角色）的更多信息，请参阅 IAM 用户指南 中的[创建向 AWS 服务委派权限的角色](#)。

有关 IAM 角色的更多信息，请参阅 IAM 用户指南 中的[IAM 角色](#)。使用以下程序创建 IAM 角色。

创建 IAM 角色（执行角色）

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 选择 Create role
3. 在 Select type of trusted entity 中，选择 AWS service，然后选择 Lambda。这将允许 Lambda 函数在您的账户下调用 AWS 服务。
4. 选择 Next: Permissions。
5. 在 Filter: Policy type 中输入 AWSLambdaExecute，并选择 Next: Review。
6. 在 Role name* 中，输入在 AWS 账户内唯一的角色名称（例如 lambda-s3-execution-role），然后选择 Create role。
7. 打开您刚创建的服务角色。

8. 在 Permissions 选项卡下，选择 Add inline policy。
9. 在 service 中，选择 Choose a service。
10. 在 Select a service below 中，选择 S3。
11. 在 Actions 中，选择 Select actions。
12. 在 Access level groups 下展开 Write，然后选择 PutObject。
13. 选择 Resources，然后选中 Any 复选框。
14. 选择查看策略。
15. 输入 Name*，然后选择 Create policy。请注意策略规范：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "VisualEditor0",  
            "Effect": "Allow",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::*/*"  
        }  
    ]  
}
```

16. 在您角色的 Summary 下，记录 Role ARN。在下一步中，创建 Lambda 函数时需要用到它。

下一步

步骤 2.3：创建 Lambda 函数并手动对其进行测试 (p. 177)

步骤 2.3：创建 Lambda 函数并手动对其进行测试

请在此部分中执行以下操作：

- 通过上传部署程序包来创建 Lambda 函数。
- 通过手动调用 Lambda 函数并将示例 Amazon S3 事件数据作为参数传递来对其进行测试。

步骤 2.3.1：创建 Lambda 函数（上传部署程序包）

在本步骤中，您将使用 AWS CLI 上传部署程序包。

1. 在命令提示符处，使用 adminuser 作为 --profile 来运行以下 Lambda AWS CLI create-function 命令。有关该设置的更多信息，请参阅[配置 AWS CLI](#)。您需要提供 .zip 文件路径和执行角色 ARN 来更新该命令。对于运行时参数，根据您创建部署程序包时的代码示例，在 nodejs6.10、nodejs4.3、python3.6、python2.7 或 java8 之间选择。

```
$ aws lambda create-function \  
--region region \  
--function-name CreateThumbnail \  
--zip-file file:///file-path/CreateThumbnail.zip \  
--role role-arn \  
--handler CreateThumbnail.handler \  
--runtime runtime \  
--profile adminuser \  
--timeout 10 \  
--memory-size 1024
```

或者，您也可以将 .zip 文件上传到同一 AWS 区域中的 Amazon S3 存储桶，然后在之前的命令中指定该存储桶和对象名称。您需要将 --zip-file 参数替换为 --code 参数，如下所示：

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

2. 记下函数 ARN。在下一章节向 Amazon S3 存储桶添加通知配置时，您会用到它。
3. (可选) 之前的命令指定 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 \
--region region \
--timeout timeout-in-seconds \
--profile adminuser
```

Note

您可以使用 AWS Lambda 控制台创建 Lambda 函数，在这种情况下，应记下 `create-function` AWS CLI 命令参数的值。您应在控制台 UI 中提供相同的值。

步骤 2.3.2：测试 Lambda 函数（手动调用）

在本步骤中，您将使用示例 Amazon S3 事件数据手动调用 Lambda 函数。您可以使用 AWS 管理控制台或 AWS CLI 测试该函数。

测试 Lambda 函数（控制台）

1. 在[手动调用 Lambda 函数并验证结果、日志和指标 \(p. 11\)](#)中按照“入门”中的步骤创建并调用 Lambda 函数。对于用于测试的示例事件，请在 Sample event template 中选择 S3 Put。
2. 验证是否已在目标存储桶中创建缩略图并在 AWS Lambda 控制台中监控 Lambda 函数的活动，如下所示：
 - AWS Lambda 控制台在函数的 Cloudwatch Metrics at a glance 部分中显示某些 CloudWatch 指标的图表化表示。
 - 对于每个图表，您还可以单击 logs 链接来直接查看 CloudWatch Logs。

测试 Lambda 函数 (AWS CLI)

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": "A3NL1KOZZKExample"
                },
                "arn": "arn:aws:s3:::sourcebucket"
            },
            "object": {
                "key": "HappyFace.jpg",
                "size": 1024,
                "eTag": "d41d8cd98f00b204e9800998ecf8427e",
                "versionId": "096fKKXTRTtl3on89fVO.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 \
--invocation-type Event \
--function-name CreateThumbnail \
--region region \
--payload file://file-path/inputfile.txt \
--profile adminuser \
outputfile.txt
```

Note

您可以调用该函数是因为您使用自己的凭证调用自己的函数。在下一节中，您将配置 Amazon S3 来代表您调用此函数，这需要您向与 Lambda 函数关联的访问策略添加权限，以授予 Amazon S3 调用您的函数的权限。

- 验证是否已在目标存储桶中创建缩略图并在 AWS Lambda 控制台中监控 Lambda 函数的活动，如下所示：
 - AWS Lambda 控制台在函数的 Cloudwatch Metrics at a glance 部分中显示某些 CloudWatch 指标的图表化表示。
 - 对于每个图表，您还可以单击 logs 链接来直接查看 CloudWatch Logs。

下一步

[步骤 3：添加事件源（配置 Amazon S3 以发布事件）\(p. 179\)](#)

步骤 3：添加事件源（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。

步骤 3.1：向 Lambda 函数的访问权限策略添加权限

- 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。
 - 存储桶归特定的 AWS 账户所有。如果存储桶拥有者删除了某个存储桶，则其他 AWS 账户可以创建使用该名称的存储桶。该条件确保只有特定的 AWS 账户能调用您的 Lambda 函数。

```
$ aws lambda add-permission \
--function-name CreateThumbnail \
--region region \
--statement-id some-unique-id \
--action "lambda:InvokeFunction" \
--principal s3.amazonaws.com \
--source-arn arn:aws:s3:::sourcebucket \
--source-account bucket-owner-account-id \
--profile adminuser
```

- 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 function-name \
--profile adminuser
```

步骤 3.2：在存储桶上配置通知

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。在配置中，指定以下内容：

- 事件类型 - 在本教程中，选择 `ObjectCreated (All)` Amazon S3 事件类型。
- Lambda 函数 - 这是您希望 Amazon S3 调用的 Lambda 函数。

有关向存储桶添加通知配置的说明，请参阅 Amazon Simple Storage Service 控制台用户指南 中的[启用事件通知](#)。

步骤 3.3：测试设置

全部完成！现在，`adminuser` 可以按以下方式测试设置：

- 使用 Amazon S3 控制台将 `.jpg` 或 `.png` 对象上传到源存储桶。
- 使用 `CreateThumbnail` 函数验证是否在目标存储桶中创建了缩略图。
- `adminuser` 也可以验证 CloudWatch Logs。您可以在 AWS Lambda 控制台中监控 Lambda 函数的活动。例如，在控制台中选择 `logs` 链接可查看日志，包括您的函数写入到 CloudWatch Logs 的日志。

步骤 4：使用 AWS SAM 和 AWS CloudFormation 执行部署

在上一部分，您使用 AWS Lambda API 通过一个部署程序包 ZIP 文件创建并更新了 Lambda 函数。然而，本机制可能不便用于自动执行函数部署步骤，或者协调事件源和下游资源等其他无服务器应用程序元素之间的部署和更新。

您可以使用 AWS CloudFormation 轻松指定、部署和配置无服务器应用程序。AWS CloudFormation 是一项有助于您对 Amazon Web Services 资源进行建模和设置的服务，能使您花费更少的时间来管理这些资源，

而将更多的时间用于关注在 AWS 中运行的应用程序上。您可以创建一个描述您所需的所有 AWS 资源 (如 Lambda 函数或 DynamoDB 表) 的模板，并且 AWS CloudFormation 将负责为您预配和配置这些资源。

此外，您也可以使用 AWS 无服务器应用程序模型表明包含无服务器应用程序的资源。Lambda 函数和 API 等资源类型完全受 AWS CloudFormation 支持，能让您更轻松地定义和部署无服务器应用程序。

有关更多信息，请参阅 [部署基于 Lambda 的应用程序 \(p. 264\)](#)。

Amazon S3 缩略图应用程序规范

以下内容包含该应用程序的 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。

```
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
```

部署无服务器应用程序

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 [打包和部署 \(p. 286\)](#)。

配合使用 AWS Lambda 和 Kinesis

您可以创建一个 Kinesis 流，每小时从几十万个源（例如网站点击流、财务交易记录、社交媒体源、IT 日志和位置跟踪事件）中连续捕获和存储数 TB 数据。有关更多信息，请参阅 [Kinesis](#)。

如果在流上检测到记录，您可以订阅 Lambda 函数以自动从 Kinesis 流读取消批记录并处理它们。随后，AWS Lambda 会定期（每秒一次）轮询流中的新记录。

请注意有关 Kinesis 和 AWS Lambda 集成的工作原理的以下信息：

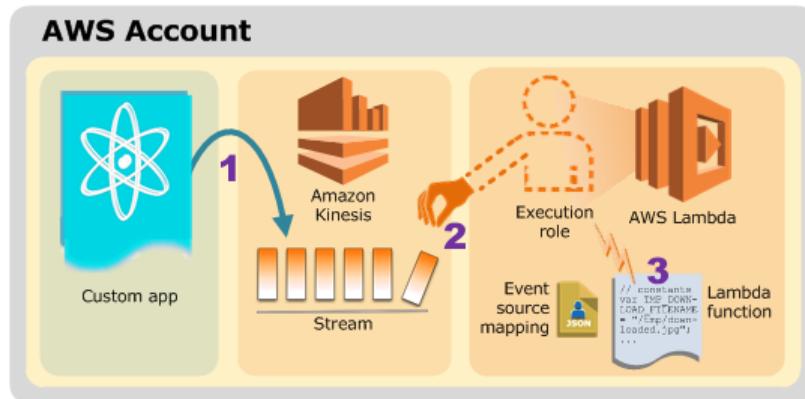
- 基于流的模型 - 这是一个模型（请参阅 [事件源映射 \(p. 142\)](#)），其中 AWS Lambda 轮询流并在检测到新记录时通过将新记录作为参数传递来调用您的 Lambda 函数。

在基于流的模型中，您在 AWS Lambda 中保留事件源映射。事件源映射描述了哪个流映射到哪个 Lambda 函数。AWS Lambda 提供了可用于创建映射的 API ([CreateEventSourceMapping \(p. 382\)](#))。您也可以使用 AWS Lambda 控制台创建事件源映射。

- 同步调用 - AWS Lambda 通过轮询 Kinesis 流来使用 RequestResponse 调用类型（同步调用）来调用 Lambda 函数。有关调用类型的更多信息，请参阅 [调用类型 \(p. 142\)](#)。
- 事件结构 - Lambda 函数接收的事件是 AWS Lambda 从流读取的记录的集合。在配置事件源映射时，您指定的批处理大小是您希望 Lambda 函数在每次调用时接收的最大记录数。

无论哪个对象调用 Lambda 函数，AWS Lambda 始终代表您执行 Lambda 函数。如果 Lambda 函数需要访问任何 AWS 资源，您需要授予访问这些资源的相关权限。您还需要向 AWS Lambda 授予轮询 Kinesis 流的权限。您应该向 AWS Lambda 可代入的 IAM 角色（执行角色）授予所有这些权限，以便轮询流并代表您执行 Lambda 函数。您应该先创建该角色，然后在创建 Lambda 函数时启用它。有关更多信息，请参阅 [管理权限：使用 IAM 角色（执行角色）\(p. 339\)](#)。

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。

有关引导您完成示例设置的教程，请参阅[教程：将 AWS Lambda 与 Kinesis 结合使用 \(p. 182\)](#)。

教程：将 AWS Lambda 与 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。

本教程分为两个主要部分：

- 首先，执行创建 Lambda 函数的必要设置，然后使用示例事件数据（不需要 Kinesis 流）手动调用该函数以便对其进行测试。
- 其次，创建一个 Kinesis 流（事件源）。在 AWS Lambda 中添加事件源映射以将该流关联到您的 Lambda 函数。AWS Lambda 开始轮询该流，您使用 Kinesis API 将测试记录添加到该流，然后验证 AWS Lambda 是否已执行 Lambda 函数。

Important

Lambda 函数和 Kinesis 流必须位于同一个 AWS 区域中。本教程假定您在 us-west-2 区域中创建了这些资源。

在本教程中，您将使用 AWS Command Line Interface 执行 AWS Lambda 操作，如创建 Lambda 函数、创建流和将记录添加到该流。在创建 Kinesis 流之前使用 AWS Lambda 控制台手动调用函数。在控制台 UI 中验证返回值和日志。

下一步

[步骤 1：准备 \(p. 182\)](#)

步骤 1：准备

确保您已完成以下步骤：

- 注册 AWS 账户并在该账户中创建管理员用户（名为 adminuser）。有关说明，请参阅[设置 AWS 账户 \(p. 4\)](#)。
- 安装并设置 AWS CLI。有关说明，请参阅[设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)。

下一步

[步骤 2：创建 Lambda 函数并手动调用该函数（使用示例事件数据）\(p. 183\)](#)

步骤 2：创建 Lambda 函数并手动调用该函数（使用示例事件数据）

请在此部分中执行以下操作：

- 使用提供的示例代码创建 Lambda 函数部署程序包。您将用于处理 Kinesis 事件的示例 Lambda 函数代码有各种语言版本。选择一种语言并按照相应说明创建部署程序包。

Note

要查看在您的函数中使用其他 AWS 服务的更多示例，包括调用其他 Lambda 函数，请参阅[AWS SDK for JavaScript](#)

- 创建 IAM 角色（执行角色）。在上传部署程序包时，需要指定 Lambda 可代入的代表您执行该函数的 IAM 角色（执行角色）。
- 通过上传部署程序包创建 Lambda 函数，然后使用示例 Kinesis 事件数据手动调用该函数以便对其进行测试。

主题

- [步骤 2.1：创建部署程序包 \(p. 183\)](#)
- [步骤 2.2：创建执行角色（IAM 角色）\(p. 187\)](#)
- [步骤 2.3：创建 Lambda 函数并手动对其进行测试 \(p. 187\)](#)

步骤 2.1：创建部署程序包

从 Filter View 列表中，选择要用于 Lambda 函数的语言。适当的部分随即出现，其中包含创建部署程序包的代码和具体说明。

Node.js

以下是将 Kinesis 事件记录作为输入接收并对其进行处理的示例 Node.js 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。

按照以下说明创建一个 AWS Lambda 函数部署程序包。

1. 打开文本编辑器，然后复制以下代码。

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
    //console.log(JSON.stringify(event, null, 2));
    event.Records.forEach(function(record) {
        // Kinesis data is base64 encoded so decode here
        var payload = new Buffer(record.kinesis.data, 'base64').toString('ascii');
        console.log('Decoded payload:', payload);
    });
    callback(null, "message");
}
```

```
};
```

Note

代码示例与 Node.js 运行时 v6.10 或 v4.3 兼容。有关更多信息，请参阅[编程模型 \(Node.js\) \(p. 18\)](#)

2. 将该文件保存为 `ProcessKinesisRecords.js`。
3. 将 `ProcessKinesisRecords.js` 文件压缩为 `ProcessKinesisRecords.zip`。

下一步

[步骤 2.2：创建执行角色（IAM 角色）\(p. 187\)](#)

Java

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。

```
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 ProcessKinesisEvents 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。

使用之前的代码（在名为 `ProcessKinesisEvents.java` 的文件中）创建一个部署程序包。确保添加以下依赖项：

- `aws-lambda-java-core`
- `aws-lambda-java-events`

有关更多信息，请参阅[使用 Java 编写 Lambda 函数的编程模型 \(p. 33\)](#)。

部署程序包既可以是 `.zip` 文件，也可以是独立的 `.jar`。您可以使用自己熟悉的任何构建和打包工具来创建部署程序包。有关如何使用 Maven 构建工具创建独立 `.jar` 的示例，请参阅[使用 Maven 但不借助任何 IDE 创建 .jar 部署程序包 \(Java\) \(p. 88\)](#)和[使用 Maven 和 Eclipse IDE 创建 .jar 部署程序包 \(Java\) \(p. 90\)](#)。有关如何使用 Gradle 构建工具创建 `.zip` 文件的示例，请参阅[创建 .zip 部署程序包 \(Java\) \(p. 92\)](#)。

确认已创建您的部署程序包后，请转到下一步来创建 IAM 角色（执行角色）。您将在创建 Lambda 函数时指定此角色。

下一步

[步骤 2.2：创建执行角色（IAM 角色）\(p. 187\)](#)

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。

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

要创建部署程序包，请按照[.NET 内核 CLI \(p. 79\)](#)中所列的步骤操作。在这种情况下，请在创建完成 .NET 项目后注意以下内容：

- 将默认的 `Program.cs` 文件重命名为您选择的文件名，如 `ProcessingKinesisEvents.cs`。
- 使用上述代码实例替换重命名的 `Program.cs` 文件中的默认内容。

确认已创建您的部署程序包后，请转到下一步来创建 IAM 角色（执行角色）。您将在创建 Lambda 函数时指定此角色。

下一步

[步骤 2.2：创建执行角色（IAM 角色）\(p. 187\)](#)

Python

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。

按照说明创建 AWS Lambda 函数部署程序包。

1. 打开文本编辑器，然后复制以下代码。

Note

利用 `from __future__ import print_function` 语句，可以编写与 Python 2 或 3 兼容的代码。如果您使用的是运行时版本 3.6，则不必将它包含在内。

```
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))
```

2. 将该文件保存为 `ProcessKinesisRecords.py`。
3. 将 `ProcessKinesisRecords.py` 文件压缩为 `ProcessKinesisRecords.zip`。

下一步

[步骤 2.2：创建执行角色（IAM 角色）\(p. 187\)](#)

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。

按照说明创建 AWS Lambda 函数部署程序包。

1. 打开文本编辑器，然后复制以下代码。

```
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 := kinesisRecordData.Data
        dataText := string(dataBytes)

        fmt.Printf("%s Data = %s \n", record.EventName, dataText)
    }
}
```

2. 将该文件保存为 `ProcessKinesisRecords.go`。
3. 将 `ProcessKinesisRecords.go` 文件压缩为 `ProcessKinesisRecords.zip`。

下一步

[步骤 2.2：创建执行角色（IAM 角色）\(p. 187\)](#)

步骤 2.2：创建执行角色（IAM 角色）

在本节中，您将使用下面的预定义角色类型和访问策略创建 IAM 角色：

- AWS Lambda 类型的 AWS 服务角色 - 此角色为 AWS Lambda 授予代入 IAM 角色的权限。
- AWSLambdaKinesisExecutionRole - 这是您附加到 IAM 角色的访问权限策略。

有关 IAM 角色的更多信息，请参阅 IAM 用户指南 中的 [IAM 角色](#)。使用以下程序创建 IAM 角色。

创建 IAM 角色（执行角色）

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 按照 IAM 用户指南 中[创建角色以向 AWS 服务委派权限](#)的步骤创建 IAM 角色（执行角色）。遵循步骤 创建角色时，请注意以下事项：
 - 在 Role Name 中，使用在 AWS 账户内唯一的名称（例如，lambda-kinesis-execution-role）。
 - 在 Select Role Type 中，选择 AWS Service Roles，然后选择 AWS Lambda。这将为 AWS Lambda 服务授予代入 IAM 角色的权限。
 - 在 Attach Policy 中，选择 AWSLambdaKinesisExecutionRole。此策略中的权限足以用于本教程中的 Lambda 函数。
3. 记下角色 ARN。在下一步中，创建 Lambda 函数时需要用到它。

下一步

[步骤 2.3：创建 Lambda 函数并手动对其进行测试 \(p. 187\)](#)

步骤 2.3：创建 Lambda 函数并手动对其进行测试

请在此部分中执行以下操作：

- 通过上传部署程序包来创建 Lambda 函数。
- 通过手动调用 Lambda 函数来对其进行测试。您应使用示例 Kinesis 事件数据，而不是创建事件源。

在下一节中，您将创建 Kinesis 流并测试端到端体验。

步骤 2.3.1：创建 Lambda 函数（上传部署程序包）

在本步骤中，您将使用 AWS CLI 上传部署程序包。

在命令提示符处，使用 adminuser profile 运行下面的 Lambda CLI `create-function` 命令。有关该设置的更多信息，请参阅[配置 AWS CLI](#)。

您需要提供 .zip 文件路径和执行角色 ARN 来更新该命令。`--runtime` 参数值可以是 `python3.6`、`python2.7`、`nodejs6.10`、`nodejs4.3` 或 `java8`，具体取决于您编写代码所用的语言。

```
$ aws lambda create-function \
--region region \
--function-name ProcessKinesisRecords \
--zip-file fileb://file-path/ProcessKinesisRecords.zip \
--role execution-role-arn \
--handler handler \
--runtime runtime-value \
```

```
--profile adminuser
```

对于 Java，`--handler` 参数值应该为 `example.ProcessKinesisRecords::recordHandler`。对于 Node.js，该参数值应为 `ProcessKinesisRecords.handler`；对于 Python，该参数值应为 `ProcessKinesisRecords.lambda_handler`。

或者，您也可以将 `.zip` 文件上传到同一 AWS 区域中的 Amazon S3 存储桶，然后在之前的命令中指定该存储桶和对象名称。您需要将 `--zip-file` 参数替换为 `--code` 参数，如下所示：

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

Note

您可以使用 AWS Lambda 控制台创建 Lambda 函数，在这种情况下，应记下 `create-function` AWS CLI 命令参数的值。您应在控制台 UI 中提供相同的值。

步骤 2.3.2：测试 Lambda 函数（手动调用）

使用示例 Kinesis 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（控制台）

1. 在[手动调用 Lambda 函数并验证结果、日志和指标 \(p. 11\)](#)中按照“入门”中的步骤创建并调用 Lambda 函数。对于用于测试的示例事件，请在 Sample event template 中选择 Kinesis。
2. 在控制台中验证结果。

测试 Lambda 函数 (AWS CLI)

1. 将以下 JSON 复制到文件中并将其保存为 `input.txt`。

```
{  
    "Records": [  
        {  
            "kinesis": {  
                "partitionKey": "partitionKey-3",  
                "kinesisSchemaVersion": "1.0",  
                "data": "SGVsbG8sIHRoaXMgXmgYSB0ZXN0IDEyMy4=",  
                "sequenceNumber":  
                    "49545115243490985018280067714973144582180062593244200961"  
            },  
            "eventSource": "aws:kinesis",  
            "eventID":  
                "shardId-000000000000:49545115243490985018280067714973144582180062593244200961",  
                "invokeIdentityArn": "arn:aws:iam::account-id:role/testLEBRole",  
                "eventVersion": "1.0",  
                "eventName": "aws:kinesis:record",  
                "eventSourceARN": "arn:aws:kinesis:us-west-2:35667example:stream/  
examplestream",  
                "awsRegion": "us-west-2"  
            }  
        ]  
    }
```

2. 执行下面的 `invoke` 命令：

```
$ aws lambda invoke \  
--invocation-type Event \  

```

```
--function-name ProcessKinesisRecords \
--region region \
--payload file:///file-path/input.txt \
--profile adminuser
outputfile.txt
```

Note

在本教程示例中，消息保存在 `outputfile.txt` 文件中。如果您请求同步执行（`RequestResponse` 作为调用类型），函数将在响应正文中返回字符串消息。对于 Node.js，该消息可能为以下形式之一（您在代码中指定的任何一个）：

```
context.succeed("message")
context.fail("message")
context.done(null, "message")
```

对于 Python 或 Java，它是返回语句中的消息：

```
return "message"
```

下一步

步骤 3：添加事件源 (创建 Kinesis 流并将其与您的 Lambda 函数关联) (p. 189)

步骤 3：添加事件源 (创建 Kinesis 流并将其与您的 Lambda 函数关联)

在本节中，您将创建 Kinesis 流，然后在 AWS Lambda 中添加事件源，以便将 Kinesis 流与您的 Lambda 函数关联。

创建事件源后，AWS Lambda 即开始轮询该流。之后，您可以通过以下方式测试设置：向流中添加事件并验证 AWS Lambda 是否代表您执行了 Lambda 函数：

步骤 3.1：创建 Kinesis 流

使用下面的 Kinesis `create-stream` CLI 命令创建流。

```
$ aws kinesis create-stream \
--stream-name examplestream \
--shard-count 1 \
--region region \
--profile adminuser
```

运行下面的 Kinesis `describe-stream` AWS CLI 命令以获取流 ARN。

```
$ aws kinesis describe-stream \
--stream-name examplestream \
--region region \
--profile adminuser
```

您需要下一步中的流 ARN 来将该流关联到您的 Lambda 函数。流的格式为：

```
arn:aws:kinesis:aws-region:account-id:stream/stream-name
```

步骤 3.2：在 AWS Lambda 中添加事件源

运行以下 AWS CLI `add-event-source` 命令。命令执行后，记下 UUID。您需要该 UUID 来在任何命令（如删除事件源时）中引用该事件源。

```
$ aws lambda create-event-source-mapping \
--region region \
--function-name ProcessKinesisRecords \
--event-source kinesis-stream-arn \
--batch-size 100 \
--starting-position TRIM_HORIZON \
--profile adminuser
```

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 \
--region region \
--function-name ProcessKinesisRecords \
--event-source kinesis-stream-arn \
--profile adminuser \
--debug
```

在该响应中，您可以验证状态值是否为 enabled。

Note

如果禁用事件源映射，AWS Lambda 将停止轮询 Kinesis 流。如果重新启用事件源映射，AWS Lambda 将从其停止轮询的序列号处恢复轮询，因此每条记录都会在您禁用映射前或启用映射后进行处理。如果序列号在 TRIM_HORIZON 之后，则当您重新启用事件源映射时，轮询将从 TRIM_HORIZON 开始。但是，如果您创建了新的事件源映射，轮询将始终从 TRIM_HORIZON、LATEST 或 AT_TIMESTAMP 开始，具体取决于您指定的开始位置。即使您删除了某个事件源映射并使用与删除的事件源映射相同的配置创建了一个新事件源映射，此规则也适用。

步骤 3.3：测试设置

全部完成！现在，adminuser 可以按以下方式测试设置：

1. 使用下面的 AWS CLI 命令，将事件记录添加到 Kinesis 流。--data 值是 "Hello, this is a test." 字符串的 base64 编码值。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 examplestream \
--data "This is a test. final" \
--partition-key shardId-000000000000 \
--region region \
--profile adminuser
```

2. AWS Lambda 轮询该流，当检测到流有更新时，它会通过传递流中的事件数据来调用您的 Lambda 函数。

AWS Lambda 担任执行角色来轮询流。您已为该角色授予执行必要的 Kinesis 操作的权限，以便让 AWS Lambda 轮询流并读取来自流的事件。

3. 您的函数将执行并将日志添加到与 Amazon CloudWatch 中的 Lambda 函数对应的日志组中。

adminuser 还可以验证 Amazon CloudWatch 控制台中报告的日志。确保您检查的是创建 Lambda 函数所在的 AWS 区域的日志。

步骤 4：使用 AWS SAM 和 AWS CloudFormation 执行部署

在上一部分，您使用 AWS Lambda API 通过一个部署程序包 ZIP 文件创建并更新了 Lambda 函数。然而，本机制可能不便用于自动执行函数部署步骤，或者协调事件源和下游资源等其他无服务器应用程序元素之间的部署和更新。

您可以使用 AWS CloudFormation 轻松指定、部署和配置无服务器应用程序。AWS CloudFormation 是一项有助于您对 Amazon Web Services 资源进行建模和设置的服务，能使您花费更少的时间来管理这些资源，而将更多的时间用于关注在 AWS 中运行的应用程序上。您可以创建一个描述您所需的所有 AWS 资源（如 Lambda 函数或 DynamoDB 表）的模板，并且 AWS CloudFormation 将负责为您预配和配置这些资源。

此外，您也可以使用 AWS 无服务器应用程序模型表明包含无服务器应用程序的资源。Lambda 函数和 API 等资源类型完全受 AWS CloudFormation 支持，能让您更轻松地定义和部署无服务器应用程序。

有关更多信息，请参阅 [部署基于 Lambda 的应用程序 \(p. 264\)](#)。

Kinesis 应用程序规范文件

以下内容包含该应用程序的 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。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  ProcessKinesisRecords:
    Type: AWS::Serverless::Function
    Properties:
      Handler: handler
      Runtime: runtime
      Policies: AWSLambdaKinesisExecutionRole
      Events:
        Stream:
          Type: Kinesis
          Properties:
            Stream: !GetAtt ExampleStream.Arn
            BatchSize: 100
            StartingPosition: TRIM_HORIZON

  ExampleStream:
    Type: AWS::Kinesis::Stream
    Properties:
      ShardCount: 1
```

部署无服务器应用程序

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 [打包和部署 \(p. 286\)](#)。

配合使用 AWS Lambda 和 Amazon DynamoDB

您可以使用 Lambda 函数作为 Amazon DynamoDB 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。要创建触发器，首先要为表启用 Amazon DynamoDB Streams。然后，编写一个 Lambda 函数来处理发布到该流的更新。

请注意有关 Amazon DynamoDB 和 AWS Lambda 集成的工作原理的以下信息：

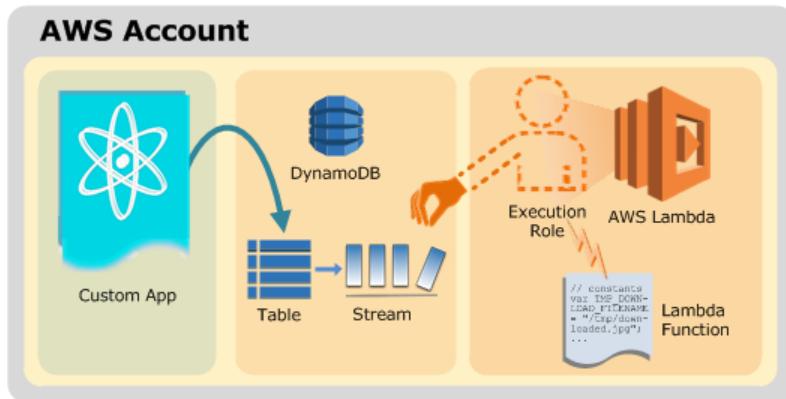
- 基于流的模型 – 这是一个模型（请参阅 [事件源映射 \(p. 142\)](#)），其中 AWS Lambda 以每秒 4 次的速率轮询流，并在检测到新记录时将更新事件作为参数传递以调用 Lambda 函数。

在基于流的模型中，您在 AWS Lambda 中保留事件源映射。事件源映射描述了哪个流映射到哪个 Lambda 函数。AWS Lambda 提供了用于创建映射的 API（[CreateEventSourceMapping \(p. 382\)](#)）。您也可以使用 AWS Lambda 控制台创建事件源映射。

- 同步调用 - AWS Lambda 使用 RequestResponse 调用类型（同步调用）来调用 Lambda 函数。有关调用类型的更多信息，请参阅 [调用类型 \(p. 142\)](#)。
- 事件结构 - Lambda 函数接收的事件是 AWS Lambda 从流读取的表更新信息。在配置事件源映射时，您指定的批处理大小是您希望 Lambda 函数在每次调用时接收的最大记录数。

无论哪个对象调用 Lambda 函数，AWS Lambda 始终代表您执行 Lambda 函数。如果 Lambda 函数需要访问任何 AWS 资源，您需要授予访问这些资源的相关权限。您还需要向 AWS Lambda 授予轮询 DynamoDB 流的权限。您应该向 AWS Lambda 可代入的 IAM 角色（执行角色）授予所有这些权限，以便轮询流并代表您执行 Lambda 函数。您应该先创建该角色，然后在创建 Lambda 函数时启用它。有关更多信息，请参阅[管理权限：使用 IAM 角色（执行角色）\(p. 339\)](#)。

下图说明应用程序的流程：



1. 自定义应用程序更新 DynamoDB 表。
2. Amazon DynamoDB 将项目更新发布到流。
3. AWS Lambda 轮询流并在检测到流中的新记录时调用 Lambda 函数。
4. 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。

有关引导您完成示例设置的教程，请参阅[教程：将 AWS Lambda 与 Amazon DynamoDB 结合使用 \(p. 192\)](#)。

教程：将 AWS Lambda 与 Amazon DynamoDB 结合使用

在本教程中，您将创建一个 Lambda 函数来处理来自 DynamoDB 流的事件。

本教程分为两个主要部分：

- 首先，执行创建 Lambda 函数的必要设置，然后使用示例事件数据手动调用该函数以便对其进行测试。
- 其次，创建启用了 DynamoDB 流的表并在 AWS Lambda 中添加事件映射，以便将该流与 Lambda 函数关联。AWS Lambda 将开始轮询该流。然后，您将测试端到端设置。在该表中创建、更新和删除项目时，Amazon DynamoDB 会向该流写入记录。AWS Lambda 在轮询该流时会检测到新记录并代表您执行您的 Lambda 函数。

Important

Lambda 函数和 DynamoDB 流必须位于同一个 AWS 区域中。本教程假定您在 us-east-1 区域中创建了这些资源。

在本教程中，您将使用 AWS Command Line Interface 执行 AWS Lambda 操作，如创建 Lambda 函数、创建流和将记录添加到该流。在创建 DynamoDB 流之前使用 AWS Lambda 控制台手动调用函数。在控制台 UI 中验证返回值和日志。

下一步

[步骤 1：准备 \(p. 193\)](#)

步骤 1：准备

确保您已完成以下步骤：

- 注册 AWS 账户并在该账户中创建管理员用户（名为 adminuser）。有关说明，请参阅 [设置 AWS 账户 \(p. 4\)](#)
- 安装并设置 AWS CLI。有关说明，请参阅 [设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)

下一步

[步骤 2：创建 Lambda 函数并手动调用该函数（使用示例事件数据）\(p. 193\)](#)

步骤 2：创建 Lambda 函数并手动调用该函数（使用示例事件数据）

请在此部分中执行以下操作：

- 使用提供的示例代码创建 Lambda 函数部署程序包。您将用于处理 DynamoDB 事件的示例 Lambda 函数代码有各种语言版本。选择一种语言并按照相应说明创建部署程序包。

Note

要查看在您的函数中使用其他 AWS 服务的更多示例，包括调用其他 Lambda 函数，请参阅 [AWS SDK for JavaScript](#)

- 创建 IAM 角色（执行角色）。在上传部署程序包时，需要指定 Lambda 可代入的代表您执行该函数的 IAM 角色（执行角色）。例如，AWS Lambda 需要 DynamoDB 操作权限，以便轮询流并读取来自流的记录。在拉模型中，您还必须授予 AWS Lambda 权限才能调用您的 Lambda 函数。示例 Lambda 函数向 CloudWatch 写入一些事件数据，因此，该函数需要必要的 CloudWatch 操作权限。
- 通过上传部署程序包创建 Lambda 函数，然后使用示例 DynamoDB 事件数据手动调用该函数以便对其进行测试。创建 Lambda 函数时，您需要提供部署程序包和 IAM 角色。此外，您还可以指定其他的配置信息，如函数名、内存大小、要使用的运行时环境、处理程序等。有关这些参数的更多信息，请参阅 [CreateFunction \(p. 387\)](#)。创建 Lambda 函数后，您将使用示例 Amazon DynamoDB 事件数据调用它。

主题

- [步骤 2.1：创建 Lambda 函数部署程序包 \(p. 193\)](#)
- [步骤 2.2：创建执行角色（IAM 角色）\(p. 197\)](#)
- [步骤 2.3：创建 Lambda 函数并手动对其进行测试 \(p. 197\)](#)

步骤 2.1：创建 Lambda 函数部署程序包

从 Filter View 列表中，选择要用于 Lambda 函数的语言。适当的部分随即出现，其中包含创建部署程序包的代码和具体说明。

Node.js

1. 打开文本编辑器，然后复制以下代码。

```
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");
};
```

Note

代码示例与 Node.js 运行时 v6.10 或 v4.3 兼容。有关更多信息，请参阅[编程模型 \(Node.js\) \(p. 18\)](#)

2. 将该文件保存为 `ProcessDynamoDBStream.js`。
3. 将 `ProcessDynamoDBStream.js` 文件压缩为 `ProcessDynamoDBStream.zip`。

下一步

[步骤 2.2：创建执行角色（IAM 角色）\(p. 197\)](#)

Java

在以下代码中，`handleRequest` 是处理程序，AWS Lambda 调用该程序并为之提供事件数据。该处理程序使用了预定义的 `DynamodbEvent` 类（在 `aws-lambda-java-events` 库中定义）。

```
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.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。

使用之前的代码（在名为 `DDBEventProcessor.java` 的文件中）创建一个部署程序包。确保添加以下依赖项：

- `aws-lambda-java-core`
- `aws-lambda-java-events`

有关更多信息，请参阅[使用 Java 编写 Lambda 函数的编程模型 \(p. 33\)](#)。

部署程序包既可以是 `.zip` 文件，也可以是独立的 `.jar`。您可以使用自己熟悉的任何构建和打包工具来创建部署程序包。有关如何使用 Maven 构建工具创建独立 `.jar` 的示例，请参阅[使用 Maven 但不借助任何 IDE 创建部署程序包](#)

[建 .jar 部署程序包 \(Java\) \(p. 88\)](#)和[使用 Maven 和 Eclipse IDE 创建 .jar 部署程序包 \(Java\) \(p. 90\)](#)。有关如何使用 Gradle 构建工具创建 .zip 文件的示例，请参阅[创建 .zip 部署程序包 \(Java\) \(p. 92\)](#)。

确认已创建您的部署程序包后，请转到下一步来创建 IAM 角色（执行角色）。您将在创建 Lambda 函数时指定此角色。

下一步

[步骤 2.2：创建执行角色（IAM 角色）\(p. 197\)](#)

C#

在以下代码中，`ProcessDynamoEvent` 是处理程序，AWS Lambda 调用该程序并为之提供事件数据。该处理程序使用了预定义的 `DynamoDbEvent` 类（在 `Amazon.Lambda.DynamoDBEvents` 库中定义）。

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

要创建部署程序包，请按照[.NET 内核 CLI \(p. 79\)](#) 中所列的步骤操作。在这种情况下，请在创建完成 .NET 项目后注意以下内容：

- 将默认的 `Program.cs` 文件重命名为您选择的文件名，如 `ProcessingDynamoDBStreams.cs`。

- 使用上述代码实例替换重命名的 Program.cs 文件中的默认内容。

确认已创建您的部署程序包后，请转到下一步来创建 IAM 角色（执行角色）。您将在创建 Lambda 函数时指定此角色。

下一步

[步骤 2.2：创建执行角色（IAM 角色）\(p. 197\)](#)

Python

- 打开文本编辑器，然后复制以下代码。

Note

利用 `from __future__ import print_function` 语句，可以编写与 Python 2 或 3 兼容的代码。如果您使用的是运行时版本 3.6，则不必将它包含在内。

```
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'])))
```

- 将该文件保存为 `ProcessDynamoDBStream.py`。
- 将 `ProcessDynamoDBStream.py` 文件压缩为 `ProcessDynamoDBStream.zip`。

下一步

[步骤 2.2：创建执行角色（IAM 角色）\(p. 197\)](#)

转到

- 打开文本编辑器，然后复制以下代码。

```
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.DynamoDB.NewImage {
            if value.DataType() == events.DataTypeString {
                fmt.Printf("Attribute name: %s, value: %s\n", name, value.String())
            }
        }
    }
}
```

- 将该文件保存为 `ProcessDynamoDBStream.go`。
- 将 `ProcessDynamoDBStream.go` 文件压缩为 `ProcessDynamoDBStream.zip`。

下一步

[步骤 2.2：创建执行角色（IAM 角色）\(p. 197\)](#)

步骤 2.2：创建执行角色（IAM 角色）

在本节中，您将使用下面的预定义角色类型和访问策略创建 IAM 角色：

- Lambda 类型的 AWS 服务角色 - 此角色授予 AWS Lambda 权限，用以调用其他 AWS 服务。
- AWSLambdaDynamoDBExecutionRole – 它包含 DynamoDB 权限策略，您可以附加该策略来增强 Lambda 的基本执行策略并允许两个服务在您的 Lambda 函数账户下交互操作。

有关 IAM 角色的更多信息，请参阅 IAM 用户指南 中的 [IAM 角色](#) 以及 IAM 用户指南 的 [创建角色以向 AWS 服务委派权限](#) 中的步骤来创建 IAM 角色（执行角色）。

要为此练习创建 IAM 角色（执行角色），请执行以下操作：

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 选择 Roles
3. 选择 Create role
4. 在 Select type of trusted entity 中，选择 AWS service，然后选择 Lambda。这将允许 Lambda 函数在您的账户下调用 AWS 服务。
5. 选择 Next: Permissions。
6. 在 Filter: Policy type 中输入 AWSLambdaDynamoDBExecutionRole，并选择 Next: Review。
7. 在 Role name* 中，输入在 AWS 账户内唯一的角色名称（例如 lambda-dynamodb-execution-role），然后选择 Create role。
8. 在您角色的 Summary 下，记录 Role ARN (Amazon 资源名称)。在下一步中创建 Lambda 函数时，您将会提供该值。

下一步

[步骤 2.3：创建 Lambda 函数并手动对其进行测试 \(p. 197\)](#)

步骤 2.3：创建 Lambda 函数并手动对其进行测试

请在此部分中执行以下操作：

- 通过上传部署程序包来创建 Lambda 函数。
- 通过手动调用 Lambda 函数来对其进行测试。您应使用示例 DynamoDB 事件数据，而不是创建事件源。

在下一节中，您将创建 DynamoDB 流并测试端到端体验。

[步骤 2.3.1：创建 Lambda 函数（上传部署程序包）](#)

在本步骤中，您将使用 AWS CLI 上传部署程序包。

在命令提示符处，使用 adminuser profile 运行下面的 Lambda CLI `create-function` 命令。如果您已创建此配置文件，请参阅 [设置 AWS 账户 \(p. 4\)](#)。

您需要提供 .zip 文件路径和执行角色 ARN 来更新该命令。`--runtime` 参数值可以是 `python3.6`、`python2.7`、`nodejs6.10`、`nodejs4.3` 或 `java8`，具体取决于您编写代码所用的语言。

```
$ aws lambda create-function \
--region us-east-1 \
--function-name ProcessDynamoDBStream \
--zip-file fileb://file-path/ProcessDynamoDBStream.zip \
```

```
--role role-arn \
--handler ProcessDynamoDBStream.lambda_handler \
--runtime runtime-value \
--profile adminuser
```

Note

如果您选择 Java 8 作为运行时，则处理程序值必须为 `packageName::methodName`。

有关更多信息，请参阅 [CreateFunction \(p. 387\)](#)。AWS Lambda 将创建该函数并返回函数配置信息。

或者，您也可以将 .zip 文件上传到同一 AWS 区域中的 Amazon S3 存储桶，然后在之前的命令中指定该存储桶和对象名称。您需要将 `--zip-file` 参数替换为 `--code` 参数，如下所示：

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

步骤 2.3.2：测试 Lambda 函数（手动调用）

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。

1. 将以下 JSON 复制到文件中并将其保存为 `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"  
}  
]  
}
```

2. 执行下面的 invoke 命令。

```
$ aws lambda invoke \  
--invocation-type RequestResponse \  
--function-name ProcessDynamoDBStream \  
--region us-east-1 \  
--payload file://file-path/input.txt \  
--profile adminuser \  
outputfile.txt
```

请注意，invoke 命令指定 RequestResponse 作为调用类型，该类型请求同步执行。有关更多信息，请参阅 [Invoke \(p. 425\)](#)。函数在响应正文中返回字符串消息（代码中 context.succeed() 中的消息）。

3. 在 `outputfile.txt` 文件中验证输出。

您可以在 AWS Lambda 控制台中监控 Lambda 函数的活动。

- AWS Lambda 控制台在针对您的函数的 Cloudwatch Metrics at a glance 部分显示某些 CloudWatch 指标的图表化表示。在 <https://console.aws.amazon.com/> 处登录 AWS 管理控制台。
- 对于每个图表，您还可以单击 `logs` 链接来直接查看 CloudWatch 日志。

下一步

[步骤 3：添加事件源（创建 DynamoDB 流并将其与您的 Lambda 函数关联）\(p. 200\)](#)

步骤 3：添加事件源（创建 DynamoDB 流并将其与您的 Lambda 函数关联）

请在此部分中执行以下操作：

- 创建一个启用了 Amazon DynamoDB 表。
- 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。
- 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。

Note

以下示例假定您拥有具有管理员权限的用户 (adminuser)。在按照流程进行操作时，请创建名为“adminuser”的用户。

为您自己创建一个 IAM 用户并将该用户添加到管理员组

1. 使用 AWS 账户电子邮件地址和密码，以 [AWS 账户根用户](#) 身份登录到 IAM 控制台 (<https://console.aws.amazon.com/iam/>)。

Note

我们强烈建议您遵守以下使用###用户的最佳实践，妥善保存根用户凭证。只在执行少数[账户和服务管理任务](#)时才作为根用户登录。

2. 在控制台的导航窗格中，选择 `Users`，然后选择 `Add user`。
3. 对于 `User name`，键入 `Administrator`。
4. 选中 AWS 管理控制台 `access` 旁边的复选框，选择 `Custom password`，然后在文本框中键入新用户的密码。您可以选择 `Require password reset` 以强制用户在下次登录时选择新密码。
5. 选择 `Next: Permissions`。
6. 在 `Set permissions for user` 页面上，选择 `Add user to group`。
7. 选择 `Create group`。
8. 在 `Create group` 对话框中，键入 `Administrators`。
9. 对于 `Filter`，选择 `Job function`。
10. 在策略列表中，选中 `AdministratorAccess` 的复选框。然后选择 `Create group`。
11. 返回到组列表中，选中您的新组所对应的复选框。如有必要，选择 `Refresh` 以便在列表中查看该组。
12. 选择 `Next: Review` 以查看要添加到新用户的组成员资格的列表。如果您已准备好继续，请选择 `Create user`。

您可使用此相同的流程创建更多的组和用户，并允许您的用户访问 AWS 账户资源。要了解有关使用策略限制用户对特定 AWS 资源的权限的信息，请参阅[访问管理和示例策略](#)。

步骤 3.1：创建启用了流的 DynamoDB 表

按照以下流程创建带有流的表：

1. 登录 AWS 管理控制台 并通过以下网址打开 DynamoDB 控制台：<https://console.aws.amazon.com/dynamodb/>。
2. 在 DynamoDB 控制台中，创建一个启用了流的表。有关启用流的更多信息，请参阅[使用 DynamoDB 流捕获表活动](#)。

Important

您必须在创建了 Lambda 函数的相同区域中创建 DynamoDB 表。本教程假定为 美国东部（弗吉尼亚北部）区域。此外，表和 Lambda 函数必须隶属同一个 AWS 账户。

3. 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。

步骤 3.2：在 AWS Lambda 中添加事件源

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 \
--region us-east-1 \
--function-name ProcessDynamoDBStream \
--event-source DynamoDB-stream-arn \
--batch-size 100 \
--starting-position TRIM_HORIZON \
--profile adminuser
```

Note

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 \
--region us-east-1 \
--function-name ProcessDynamoDBStream \
--event-source DynamoDB-stream-arn \
--profile adminuser
```

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) 等值表示未出现问题。如果出现问题，您将收到一条错误消息。

步骤 3.3：测试设置

全部完成！现在，adminuser 可以按以下方式测试设置：

1. 在 DynamoDB 控制台中，添加、更新、删除表中的项目。DynamoDB 会将这些操作记录写入流。
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。adminuser 还可以验证 Amazon CloudWatch 控制台中报告的日志。

步骤 4：使用 AWS SAM 和 AWS CloudFormation 执行部署

在上一部分，您使用 AWS Lambda API 通过一个部署程序包 ZIP 文件创建并更新了 Lambda 函数。然而，本机制可能不便用于自动执行函数部署步骤，或者协调事件源和下游资源等其他无服务器应用程序元素之间的部署和更新。

您可以使用 AWS CloudFormation 轻松指定、部署和配置无服务器应用程序。AWS CloudFormation 是一项有助于您对 Amazon Web Services 资源进行建模和设置的服务，能使您花费更少的时间来管理这些资源，而将更多的时间用于关注在 AWS 中运行的应用程序上。您可以创建一个描述您所需的所有 AWS 资源（如 Lambda 函数或 DynamoDB 表）的模板，并且 AWS CloudFormation 将负责为您预配和配置这些资源。

此外，您也可以使用 AWS 无服务器应用程序模型表明包含无服务器应用程序的资源。Lambda 函数和 API 等资源类型完全受 AWS CloudFormation 支持，能让您更轻松地定义和部署无服务器应用程序。

有关更多信息，请参阅 [部署基于 Lambda 的应用程序 \(p. 264\)](#)。

DynamoDB 应用程序规范

以下内容包含该应用程序的 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。

```
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
```

部署无服务器应用程序

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 [打包和部署 \(p. 286\)](#)。

配合使用 AWS Lambda 和 AWS CloudTrail

您可以在 AWS 账户中启用 CloudTrail 以获取您账户中的 API 调用日志和相关事件历史记录。CloudTrail 将所有 API 访问事件记录为您在启用 CloudTrail 时指定的 Amazon S3 存储桶中的对象。

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。

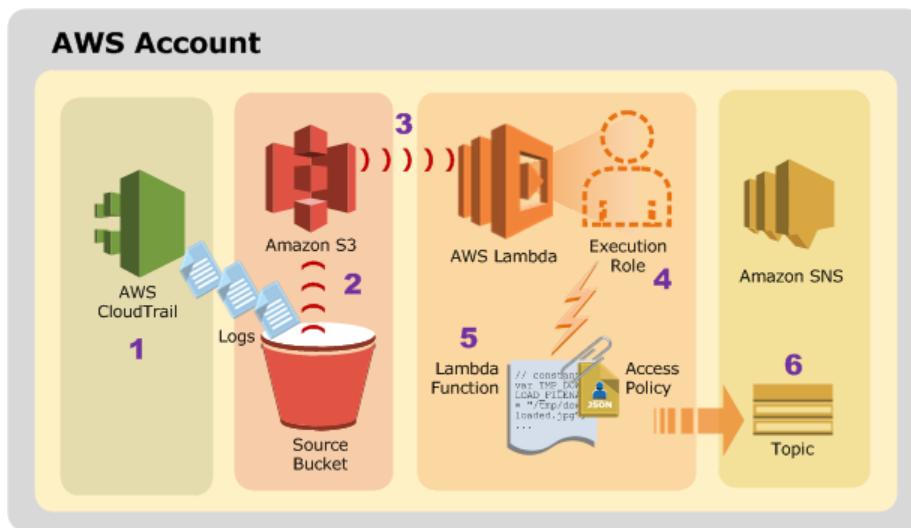
在这种情况下，您可启用 CloudTrail 以便让它将访问日志写入 S3 存储桶。对于 AWS Lambda，Amazon S3 是事件源，因此 Amazon S3 会将事件发布到 AWS Lambda 并调用 Lambda 函数。

Note

Amazon S3 只能支持一个事件目标。

有关如何将 Amazon S3 配置为事件源的详细信息，请参阅[配合使用 AWS Lambda 和 Amazon S3 \(p. 165\)](#)。

下图概述了该流程：



1. AWS CloudTrail 将日志保存到 S3 存储桶（对象创建事件）。
2. Amazon S3 检测到对象创建事件。
3. 按照存储桶通知配置的规定，Amazon S3 通过调用 Lambda 函数将 `s3:ObjectCreated:*` 事件发布到 AWS Lambda。由于 Lambda 函数的访问权限策略包括 Amazon S3 调用该函数的权限，因此 Amazon S3 可调用该函数。
4. 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。
5. Lambda 函数读取其作为参数接收的 Amazon S3 事件、确定 CloudTrail 对象的位置、读取 CloudTrail 对象，然后处理 CloudTrail 对象中的日志记录。
6. 如果日志包含带有特定 `eventType` 和 `eventSource` 值的记录，即将该事件发布到您的 Amazon SNS 主题。在[教程：将 AWS Lambda 与 AWS CloudTrail 结合使用 \(p. 204\)](#)中，您使用电子邮件协议订阅该 SNS 主题，因此您会收到电子邮件通知。

有关引导您完成示例方案的教程，请参阅[教程：将 AWS Lambda 与 AWS CloudTrail 结合使用 \(p. 204\)](#)。

教程：将 AWS Lambda 与 AWS CloudTrail 结合使用

假定您已对自己的 AWS 账户启用 AWS CloudTrail 以保留对您的账户进行的 AWS API 调用的记录（日志），并希望在每次发生用于创建 SNS 主题的 API 调用时收到通知。由于 API 是在您的账户中调用的，CloudTrail 会将日志写入到您配置的 Amazon S3 存储桶中。在这种情况下，您希望 Amazon S3 在 CloudTrail 创建日志对象时将对象创建事件发布到 AWS Lambda 并调用您的 Lambda 函数。

当 Amazon S3 调用 Lambda 函数时，它会传递标识存储桶名称和 CloudTrail 创建的对象的键名称等的 S3 事件。Lambda 函数可读取该日志对象，并了解日志中报告的 API 调用。

CloudTrail 在 S3 存储桶中创建的每个对象都是包含一个或多个事件记录的 JSON 对象。此外，每条记录提供的内容都包括 `eventSource` 和 `eventName`。

```
{  
    "Records": [  
        {  
            "eventVersion": "1.02",  
            "userIdentity": {  
                ...  
            },  
            "eventTime": "2014-12-16T19:17:43Z",  
            "eventSource": "sns.amazonaws.com",  
            "eventName": "CreateTopic",  
            "awsRegion": "us-west-2",  
            "sourceIPAddress": "72.21.198.64",  
            ...  
        },  
        {  
            ...  
        },  
        ...  
    ]  
}
```

为了展示这个过程，如果日志中报告了用于创建 Amazon SNS 主题的 API 调用，Lambda 函数会通过电子邮件向您发送通知。也就是说，Lambda 函数在分析日志时会查找包含以下内容的记录：

- `eventSource = "sns.amazonaws.com"`
- `eventName = "CreateTopic"`

如果找到，该函数会将该事件发布到 Amazon SNS 主题（您配置此主题以通过电子邮件向您发送通知）。

实现摘要

完成本教程后，您的账户中会有 Amazon S3、AWS Lambda、Amazon SNS 和 AWS Identity and Access Management (IAM) 资源：

Note

本教程假定您在 us-west-2 区域中创建了这些资源。

在 Lambda 中：

- Lambda 函数。
- 与 Lambda 函数关联的访问策略 - 使用此权限策略向 Amazon S3 授予调用 Lambda 函数的权限。此外，您还将限制该权限，以使 Amazon S3 只能针对来自特定存储桶（归特定 AWS 账户所有）的对象创建事件调用 Lambda 函数。

Note

某个 AWS 账户删除存储桶后，其他 AWS 账户可以创建使用该相同名称的存储桶。额外的条件可确保：仅当 Amazon S3 检测到来自特定存储桶（归特定 AWS 账户所有）的对象创建事件时，Amazon S3 才能调用 Lambda 函数。

在 IAM 中：

- IAM 角色（执行角色）- 通过与此角色关联的权限策略授予 Lambda 函数所需的权限。

在 Amazon S3 中：

- 存储桶 - 在本教程中，存储桶名称为 `examplebucket`。在 CloudTrail 控制台中启用跟踪时，为 CloudTrail 指定该存储桶以保存日志。
- `examplebucket` 上的通知配置 - 在该配置中，您通过调用 Lambda 函数指示 Amazon S3 将对象创建事件发布到 Lambda。有关 Amazon S3 通知功能的更多信息，请参阅 Amazon Simple Storage Service 开发人员指南中的[设置存储桶事件的通知](#)。
- `ExampleCloudTrailLog.jsonexamplebucket ##### CloudTrail ##### ()` - 在本练习的前半部分，您将使用示例 S3 事件手动调用 Lambda 函数，以便创建和测试该函数。此示例事件将 `examplebucket` 标识为存储桶名称和此示例对象键名称。Lambda 函数随后读取该对象并使用 SNS 主题向您发送电子邮件通知。

在 Amazon SNS 中：

- SNS 主题 - 订阅该主题（协议指定为电子邮件）。

您现在可以开始教程。

下一步

[步骤 1：准备 \(p. 205\)](#)

步骤 1：准备

请在此部分中执行以下操作：

- 注册 AWS 账户并设置 AWS CLI。
- 在您的账户中启用 CloudTrail。
- 创建一个 SNS 主题并订阅此主题。

按照以下章节中的步骤完成设置过程。

Note

在本教程中，我们假定您在 us-west-2 区域中设置了资源。

步骤 1.1：注册 AWS 并设置 AWS CLI

确保您已完成以下步骤：

- 注册 AWS 账户并在该账户中创建管理员用户（名为 adminuser）。有关说明，请参阅[设置 AWS 账户 \(p. 4\)](#)。

- 安装并设置 AWS CLI。有关说明，请参阅[设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)。

步骤 1.2：启用 CloudTrail

在 AWS CloudTrail 控制台中，为 CloudTrail 指定 us-west-2 区域中的 `examplebucket` 来保存日志，从而在您的账户中启用跟踪。配置跟踪时，不要启用 SNS 通知。

有关说明，请参阅 AWS CloudTrail User Guide 中的[创建和更新您的跟踪](#)。

Note

尽管您现在启用了 CloudTrail，但在本练习的前半部分，您没有为 Lambda 函数执行任何额外的配置来处理实际 CloudTrail 日志。相反，您将使用示例 CloudTrail 日志对象（您将上传的）和示例 S3 事件来手动调用和测试 Lambda 函数。在本教程的后半部分，您将执行支持 Lambda 函数处理 CloudTrail 日志的额外配置步骤。

步骤 1.3：创建 SNS 主题并订阅该主题

按照流程进行操作，在 us-west-2 区域中创建 SNS 主题，并通过提供电子邮件地址作为终端节点的方法订阅它。

创建和订阅主题

1. 创建 SNS 主题。

有关说明，请参阅 Amazon Simple Notification Service 开发人员指南 中的[创建主题](#)。

2. 通过提供电子邮件地址作为终端节点的方法订阅主题。

有关说明，请参阅 Amazon Simple Notification Service 开发人员指南 中的[订阅主题](#)。

3. 记下主题 ARN。在下面几节中，您会用到该值。

下一步

[步骤 2：创建 Lambda 函数并手动调用该函数（使用示例事件数据）\(p. 206\)](#)

步骤 2：创建 Lambda 函数并手动调用该函数（使用示例事件数据）

请在此部分中执行以下操作：

- 使用提供的示例代码创建 Lambda 函数部署程序包。您将用于处理 Amazon S3 事件的示例 Lambda 函数代码有各种语言版本。选择一种语言并按照相应说明创建部署程序包。

Note

Lambda 函数将使用一个 S3 事件，该事件提供了存储桶名称和 CloudTrail 创建的对象的键名称。Lambda 函数随后读取该对象以处理 CloudTrail 记录。

- 创建 IAM 角色（执行角色）。在上传部署程序包时，需要指定 Lambda 可代入的代表您执行该函数的 IAM 角色（执行角色）。
- 通过上传部署程序包创建 Lambda 函数，然后使用示例 CloudTrail 事件数据手动调用该函数以便对其进行测试。

主题

- [步骤 2.1：创建部署程序包 \(p. 207\)](#)
- [步骤 2.2：创建执行角色 \(IAM 角色\) \(p. 208\)](#)

- 步骤 2.3：创建 Lambda 函数并手动对其进行测试 (p. 209)

步骤 2.1：创建部署程序包

部署程序包是包含 Lambda 函数代码的 .zip 文件。在本教程中，您需要安装 `async` 库。要执行此操作，请打开命令窗口，导航至您希望将代码文件存储到的目录，您将在以下步骤中复制并保存该文件。使用 `npm` 安装异步库，如下所示：

```
npm install async
```

Node.js

1. 打开文本编辑器，然后复制以下代码。

```
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);
                });
    ],
    function done(err, result) {
        if (err) {
            return callback(err);
        }
        callback(null, result);
    }
});
```

```
        console.log('Publishing ' + matchingRecords.length + ' notification(s) in parallel...');
        await 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");
    });
};
```

Note

代码示例与 Node.js 运行时 v6.10 或 v4.3 兼容。有关更多信息，请参阅[编程模型 \(Node.js\) \(p. 18\)](#)

2. 将该文件保存为 CloudTrailEventProcessing.js。
3. 将 CloudTrailEventProcessing.js 文件压缩为 CloudTrailEventProcessing.zip。

Note

我们在本教程示例中使用的是 Node.js，但您也可以用 Java 或 Python 编写 Lambda 函数。

下一步

[步骤 2.2：创建执行角色（IAM 角色）\(p. 208\)](#)

[步骤 2.2：创建执行角色（IAM 角色）](#)

现在，您将创建在创建 Lambda 函数时指定的 IAM 角色（执行角色）。此角色具有授予 Lambda 函数所需的必要权限（如写入 CloudWatch 日志的权限、从 S3 存储桶读取 CloudTrail 日志对象的权限、在 Lambda 函数在 CloudTrail 记录中找到特定 API 调用时将事件发布到 SNS 主题的权限）的权限策略。

有关执行角色的更多信息，请参阅[管理权限：使用 IAM 角色（执行角色）\(p. 339\)](#)。

创建 IAM 角色（执行角色）

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 创建托管策略并将其附加到 IAM 角色。在本步骤中，您将修改现有 AWS 托管策略，使用其他名称保存该策略，然后将该权限策略附加到您创建的 IAM 角色。
 - a. 在 IAM 控制台的导航窗格中，选择 Policies，然后选择 Create Policy。
 - b. 在 Copy an AWS Managed Policy 旁边，选择 Select。
 - c. 在 AWSLambdaExecute 旁边，选择 Select。

- d. 将下面的策略复制到 Policy Document 中替换现有策略，然后使用您创建的 Amazon SNS 主题的 ARN 更新该策略。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:*"  
            ],  
            "Resource": "arn:aws:logs:*:*::*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Resource": "arn:aws:s3:::examplebucket/*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sns:Publish"  
            ],  
            "Resource": "your sns topic ARN"  
        }  
    ]  
}
```

3. 记下权限策略名称（在下一步中会用到）。
4. 按照 IAM 用户指南 中[创建角色以向 AWS 服务委派权限](#)的步骤创建 IAM 角色，然后将您刚刚创建的权限策略附加到该角色。遵循步骤创建角色时，请注意以下事项：
 - 在 Role Name 中，使用在 AWS 账户内唯一的名称（例如，lambda-cloudtrail-execution-role）。
 - 在 Select Role Type 中，选择 AWS Service Roles，然后选择 AWS Lambda。
 - 在 Attach Policy 中，选择在上一步中创建的策略。

下一步

[步骤 2.3：创建 Lambda 函数并手动对其进行测试 \(p. 209\)](#)

步骤 2.3：创建 Lambda 函数并手动对其进行测试

请在此部分中执行以下操作：

- 通过上传部署程序包来创建 Lambda 函数。
- 通过手动调用 Lambda 函数来对其进行测试。

在本步骤中，您使用标识存储桶名称和示例对象（即示例 CloudTrail 日志）的示例 S3 事件。在下一节中，您将 S3 存储桶通知配置为发布对象创建事件并测试端到端体验。

步骤 2.3.1：创建 Lambda 函数（上传部署程序包）

在本步骤中，您使用 AWS CLI 上传部署程序包并在使用 `adminuser profile` 创建 Lambda 函数时提供配置信息。

有关设置 `admin` 配置文件和使用 AWS CLI 的更多信息，请参阅[设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)。

Note

您需要通过提供 .zip 文件路径 (`//file-path/CloudTrailEventProcessing.zip`) 和执行角色 ARN (`execution-role-arn`) 来更新该命令。如果您使用了本教程的前面部分提供的示例代码，请将 `--runtime` 参数值设置为 `nodejs6.10` 或 `nodejs4.3`。您也可以用 Java 或 Python 编写 Lambda 函数。如果您使用了其他语言，请根据需要将 `--runtime` 参数值更改为 `java8`、`python3.6` 或 `python2.7`。

```
$ aws lambda create-function \
--region region \
--function-name CloudTrailEventProcessing \
--zip-file fileb://file-path/CloudTrailEventProcessing.zip \
--role execution-role-arn \
--handler CloudTrailEventProcessing.handler \
--runtime nodejs6.10 \
--profile adminuser \
--timeout 10 \
--memory-size 1024
```

或者，您也可以将 .zip 文件上传到同一 AWS 区域中的 Amazon S3 存储桶，然后在之前的命令中指定该存储桶和对象名称。您需要将 `--zip-file` 参数替换为 `--code` 参数，如下所示：

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

Note

您可以使用 AWS Lambda 控制台创建 Lambda 函数，在这种情况下，应记下 `create-function` AWS CLI 命令参数的值。您应在控制台中提供相同的值。

步骤 2.3.2：测试 Lambda 函数（手动调用）

在本节中，您将使用示例 Amazon S3 事件数据手动调用 Lambda 函数。Lambda 函数执行时，会从 S3 事件数据标识的存储桶中读取 S3 对象（示例 CloudTrail 日志），如果示例 CloudTrail 日志报告使用了特定的 API，则会向您的 SNS 主题发布事件。对于本教程，API 是用于创建主题的 SNS API。即：CloudTrail 日志报告将 `sns.amazonaws.com` 标识为 `eventSource`、将 `CreateTopic` 标识为 `eventName` 的记录。

1. 将下面的示例 CloudTrail 日志保存到文件 (`ExampleCloudTrailLog.json`) 中。

Note

注意：该日志中某个事件的 `sns.amazonaws.com` 为 `eventSource`，`CreateTopic` 为 `eventName`。您的 Lambda 函数将读取这些日志，如果它找到此类型的事件，则向您创建的 Amazon SNS 主题发布该事件，然后当您手动调用 Lambda 函数时，您会收到一封电子邮件。

```
{
  "Records": [
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "Root",
        "principalId": "account-id",
        "arn": "arn:aws:iam::account-id:root",
        "accountId": "account-id",
        "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-west-2",
        "sourceIPAddress": "205.251.233.176",
        "userAgent": "console.amazonaws.com",
        "requestParameters": {
            "name": "dropmeplease"
        },
        "responseElements": {
            "topicArn": "arn:aws:sns:us-west-2:account-id:exampletopic"
        },
        "requestID": "3fdb7834-9079-557e-8ef2-350abc03536b",
        "eventID": "17b46459-dada-4278-b8e2-5a4ca9ff1a9c",
        "eventType": "AwsApiCall",
        "recipientAccountId": "account-id"
    },
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "Root",
            "principalId": "account-id",
            "arn": "arn:aws:iam::account-id:root",
            "accountId": "account-id",
            "accessKeyId": "access-key-id",
            "sessionContext": {
                "attributes": {
                    "mfaAuthenticated": "false",
                    "creationDate": "2015-01-24T22:41:54Z"
                }
            }
        },
        "eventTime": "2015-01-24T23:27:02Z",
        "eventSource": "sns.amazonaws.com",
        "eventName": "GetTopicAttributes",
        "awsRegion": "us-west-2",
        "sourceIPAddress": "205.251.233.176",
        "userAgent": "console.amazonaws.com",
        "requestParameters": {
            "topicArn": "arn:aws:sns:us-west-2:account-id:exampletopic"
        },
        "responseElements": null,
        "requestID": "4a0388f7-a0af-5df9-9587-c5c98c29cbec",
        "eventID": "ec5bb073-8fa1-4d45-b03c-f07b9fc9ea18",
        "eventType": "AwsApiCall",
        "recipientAccountId": "account-id"
    }
]
}
```

- 运行 gzip 命令以从之前的源文件创建 .gz 文件。

```
$ gzip ExampleCloudTrailLog.json
```

这会创建 ExampleCloudTrailLog.json.gz 文件。

- 将 ExampleCloudTrailLog.json.gz 文件上传到您在 CloudTrail 配置中指定的 *examplebucket*。该对象在我们用于手动调用 Lambda 函数的示例 Amazon S3 事件数据中指定。
- 将下面的 JSON (示例 S3 事件) 保存到文件 input.txt 中。记下存储桶名称和对象键名称值。

在调用 Lambda 函数时，您将提供该示例事件。有关 S3 事件结构的更多信息，请参阅 Amazon Simple Storage Service 开发人员指南 中的 [事件消息结构](#)。

```
{  
    "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": "your bucket name",  
                    "ownerIdentity": {  
                        "principalId": "A3NL1KOZZKExample"  
                    },  
                    "arn": "arn:aws:s3:::mybucket"  
                },  
                "object": {  
                    "key": "ExampleCloudTrailLog.json.gz",  
                    "size": 1024,  
                    "eTag": "d41d8cd98f00b204e9800998ecf8427e",  
                    "versionId": "096fKKXTRTtl3on89fVO.nfljtsv6qko"  
                }  
            }  
        }  
    ]  
}
```

5. 在 AWS 管理控制台中，使用示例 Amazon S3 事件数据手动调用函数。在控制台中，使用以下示例 Amazon S3 事件数据。

Note

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）。

```
{  
    "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": "your bucket name",  
                    "ownerIdentity": {  
                        "principalId": "A3NL1KOZZKExample"  
                    },  
                    "arn": "arn:aws:s3:::mybucket"  
                },  
                "object": {  
                    "key": "ExampleCloudTrailLog.json.gz",  
                    "size": 1024,  
                    "eTag": "d41d8cd98f00b204e9800998ecf8427e",  
                    "versionId": "096fKKXTRTtl3on89fVO.nfljtsv6qko"  
                }  
            }  
        }  
    ]  
}
```

```
        "responseElements":{  
            "x-amz-request-id":"C3D13FE58DE4C810",  
            "x-amz-id-2":"FMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/  
JRWeUWerMUE5JgHvANOjpD"  
        },  
        "s3":{  
            "s3SchemaVersion":"1.0",  
            "configurationId":"testConfigRule",  
            "bucket":{  
                "name": "your bucket name",  
                "ownerIdentity":{  
                    "principalId": "A3NL1KOZZKExample"  
                },  
                "arn": "arn:aws:s3:::mybucket"  
            },  
            "object":{  
                "key": "ExampleCloudTrailLog.json.gz",  
                "size": 1024,  
                "eTag": "d41d8cd98f00b204e9800998ecf8427e",  
                "versionId": "096fKKXTRTtl3on89fVO.nfljtsv6qko"  
            }  
        }  
    }  
}
```

6. 执行下面的 AWS CLI 命令，以使用 adminuser 配置文件手动调用该函数。

Note

如果您尚未创建此配置文件，请参阅[设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)。

```
$ aws lambda invoke-async \  
--function-name CloudTrailEventProcessing \  
--region region \  
--invoke-args /filepath/input.txt \  
--debug \  
--profile adminuser
```

由于示例日志对象中有一条事件记录，其中显示了要调用的用于创建主题的 SNS API，因此，Lambda 函数会将该事件发布到您的 SNS 主题，而您应该会收到一封电子邮件通知。

您可以借助 CloudWatch 指标和日志监控 Lambda 函数的活动。有关 CloudWatch 监控的更多信息，请参阅[使用 Amazon CloudWatch \(p. 296\)](#)。

7. (可选) 使用 AWS CLI 手动调用 Lambda 函数，如下所示：

- 将本过程前面部分的步骤 2 中的 JSON 保存到名为 `input.txt` 的文件中。
- 执行下面的 `invoke` 命令：

```
$ aws lambda invoke \  
--invocation-type Event \  
--function-name CloudTrailEventProcessing \  
--region region \  
--payload file://file-path/input.txt \  
--profile adminuser  
outputfile.txt
```

Note

在本教程示例中，消息保存在 `outputfile.txt` 文件中。如果您请求同步执行（`RequestResponse` 作为调用类型），函数将在响应正文中返回字符串消息。

对于 Node.js，该消息可能为以下形式之一（您在代码中指定的任何一个）：

```
context.succeed("message")
context.fail("message")
context.done(null, "message")
对于 Python 或 Java，它是返回语句中的消息：
return "message"
```

下一步

[步骤 3：添加事件源（配置 Amazon S3 以发布事件）\(p. 214\)](#)

步骤 3：添加事件源（配置 Amazon S3 以发布事件）

在本节中，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。您将执行以下操作：

- 向 Lambda 函数的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。对于本教程，您将指定 s3:ObjectCreated:* 事件类型。
 - 要调用的 Lambda 函数。

步骤 3.1：向 Lambda 函数的访问权限策略添加权限

1. 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。
 - 存储桶归特定的 AWS 账户所有。如果存储桶拥有者删除了某个存储桶，则其他 AWS 账户可以创建使用该名称的存储桶。该条件确保只有特定的 AWS 账户能调用您的 Lambda 函数。

Note

如果您尚未创建 adminuser 配置文件，请参阅[设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)。

```
$ aws lambda add-permission \
--function-name CloudTrailEventProcessing \
--region region \
--statement-id Id-1 \
--action "lambda:InvokeFunction" \
--principal s3.amazonaws.com \
--source-arn arn:aws:s3:::examplebucket \
--source-account examplebucket-owner-account-id \
--profile adminuser
```

2. 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 function-name \
--profile adminuser
```

步骤 3.2：在存储桶上配置通知

在 `examplebucket` 上添加通知配置，以请求 Amazon S3 向 Lambda 发布对象创建事件。在配置中，指定以下内容：

- 事件类型 - 对于本教程，可以是创建对象的任意事件类型。
- Lambda 函数 ARN - 这是您希望 Amazon S3 调用的 Lambda 函数。ARN 的格式如下：

```
arn:aws:lambda:aws-region:account-id:function:function-name
```

例如，在 us-west-2 区域中创建的函数 CloudTrailEventProcessing 的 ARN 为：

```
arn:aws:lambda:us-west-2:account-id:function:CloudTrailEventProcessing
```

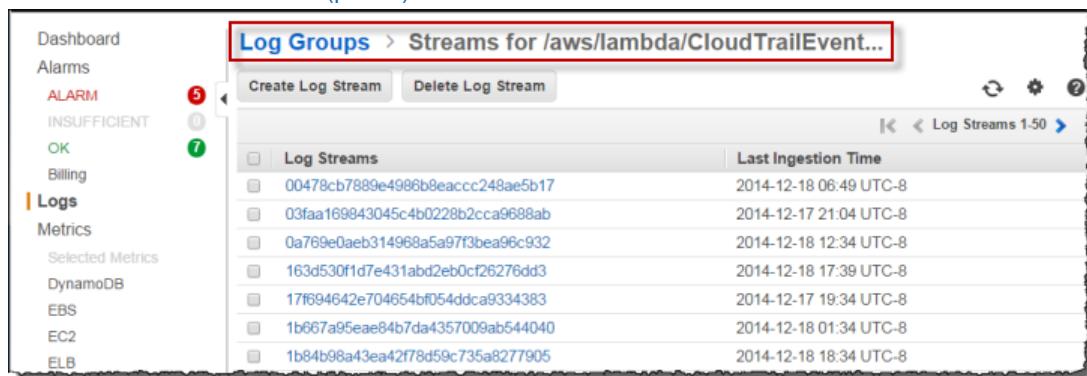
有关向存储桶添加通知配置的说明，请参阅 Amazon Simple Storage Service 控制台用户指南 中的[启用事件通知](#)。

步骤 3.3：测试设置

全部完成！现在，可以按以下方式测试设置：

1. 在您的 AWS 账户上执行一些操作。例如，在 Amazon SNS 控制台中添加另一个主题。
2. 您会收到关于该事件的电子邮件通知。
3. AWS CloudTrail 在存储桶中创建日志对象。
4. 如果您打开该日志对象 (.gz 文件)，日志会显示 CreateTopic SNS 事件。
5. 对于 AWS CloudTrail 创建的每个对象，Amazon S3 会通过将该日志对象作为事件数据传递来调用 Lambda 函数。
6. Lambda 执行您的函数。该函数分析日志，发现 CreateTopic SNS 事件，随后您会收到电子邮件通知。

您可以借助 CloudWatch 指标和日志监控 Lambda 函数的活动。有关 CloudWatch 监控的更多信息，请参阅[使用 Amazon CloudWatch \(p. 296\)](#)。



步骤 4：使用 AWS SAM 和 AWS CloudFormation 执行部署

在上一部分，您使用 AWS Lambda API 通过一个部署程序包 ZIP 文件创建并更新了 Lambda 函数。然而，本机制可能不便用于自动执行函数部署步骤，或者协调事件源和下游资源等其他无服务器应用程序元素之间的部署和更新。

您可以使用 AWS CloudFormation 轻松指定、部署和配置无服务器应用程序。AWS CloudFormation 是一项有助于您对 Amazon Web Services 资源进行建模和设置的服务，能使您花费更少的时间来管理这些资源，

而将更多的时间用于关注在 AWS 中运行的应用程序上。您可以创建一个描述您所需的所有 AWS 资源 (如 Lambda 函数或 DynamoDB 表) 的模板，并且 AWS CloudFormation 将负责为您预配和配置这些资源。

此外，您也可以使用 AWS 无服务器应用程序模型表明包含无服务器应用程序的资源。Lambda 函数和 API 等资源类型完全受 AWS CloudFormation 支持，能让您更轻松地定义和部署无服务器应用程序。

有关更多信息，请参阅 [部署基于 Lambda 的应用程序 \(p. 264\)](#)。

Amazon API Gateway 应用程序规范文件

以下内容包含该应用程序的 SAM 模板。将以下文本复制到 .yaml 文件中，并将其保存到您在上一部分创建的 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。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Parameters:
  NotificationEmail:
    Type: String
Resources:
  CloudTrailEventProcessing:
    Type: AWS::Serverless::Function
    Properties:
      Handler: handler
      Runtime: runtime
      Timeout: 10
      MemorySize: 1024
      Policies:
        Statement:
          - Effect: Allow
            Action: s3:GetObject
            Resource: !Sub 'arn:aws:s3:::${Bucket}/*'
          - Effect: Allow
            Action: sns:Publish
            Resource: !Ref Topic
      Events:
        PhotoUpload:
          Type: S3
          Properties:
            Bucket: !Ref Bucket
            Events: s3:ObjectCreated:*
  Environment:
    Variables:
      SNS_TOPIC_ARN: !Ref Topic
  Bucket:
    Type: AWS::S3::Bucket
  Trail:
    Type: AWS::CloudTrail::Trail
    Properties:
      IsLogging: true
      S3BucketName: !Ref Bucket
  Topic:
    Type: AWS::SNS::Topic
    Properties:
      Subscription:
        - Protocol: email
          Endpoint: !Ref NotificationEmail
```

部署无服务器应用程序

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 [打包和部署 \(p. 286\)](#)。

将 AWS Lambda 与其他账户中的 Amazon SNS 结合使用

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。

有关引导您完成示例设置的教程，请参阅[教程：将 AWS Lambda 与 Amazon SNS 结合使用 \(p. 217\)](#)。

教程：将 AWS Lambda 与 Amazon SNS 结合使用

在本教程中，您在一个 AWS 账户中创建 Lambda 函数，订阅单独的 AWS 账户中的 Amazon SNS 主题。

本教程分为三个主要部分：

- 首先，您执行创建 Lambda 函数的必要设置。
- 接下来，您在单独的 AWS 账户中创建一个 Amazon SNS 主题。
- 紧接着，您授予每个账户的权限以便 Lambda 函数订阅 Amazon SNS 主题。然后，您将测试端到端设置。

Important

本教程假定您在 us-east-1 区域中创建了这些资源。

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。

下一步

[步骤 1：准备 \(p. 217\)](#)

步骤 1：准备

- 注册 AWS 账户并在该账户中创建管理员用户（名为 adminuser）。
- 安装并设置 AWS CLI。

有关说明，请参阅[设置 AWS 账户 \(p. 4\)](#)。

下一步

[步骤 2：创建 Lambda 函数 \(p. 217\)](#)

步骤 2：创建 Lambda 函数

请在此部分中执行以下操作：

- 使用提供的示例代码创建 Lambda 函数部署程序包。您将用于订阅 Amazon SNS 主题的示例 Lambda 函数代码有各种语言版本。选择一种语言并按照相应说明创建部署程序包。
- 创建 IAM 角色（执行角色）。在上传部署程序包时，需要指定 Lambda 可代入的代表您执行该函数的 IAM 角色（执行角色）。

主题

- 步骤 2.1：创建 Lambda 函数部署程序包 (p. 218)
- 步骤 2.2：创建执行角色 (IAM 角色) (p. 220)

步骤 2.1：创建 Lambda 函数部署程序包

从 Filter View 列表中，选择要用于 Lambda 函数的语言。适当的部分随即出现，其中包含创建部署程序包的代码和具体说明。

Node.js

1. 打开文本编辑器，然后复制以下代码。

```
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");
};
```

Note

代码示例与 Node.js 运行时 v6.10 或 v4.3 兼容。有关更多信息，请参阅[编程模型 \(Node.js\) \(p. 18\)](#)

2. 将该文件保存为 `index.js`。
3. 将 `index.js` 文件压缩为 `LambdaWithSNS.zip`。

下一步

步骤 2.2：创建执行角色 (IAM 角色) (p. 220)

Java

- 打开文本编辑器，然后复制以下代码。

```
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;
}
```

使用之前的代码 (在名为 `LambdaWithSNS.java` 的文件中) 创建一个部署程序包。确保添加以下依赖项：

- `aws-lambda-java-core`
- `aws-lambda-java-events`

有关更多信息，请参阅 [使用 Java 编写 Lambda 函数的编程模型 \(p. 33\)](#)。

部署程序包既可以是 `.zip` 文件，也可以是独立的 `.jar`。您可以使用自己熟悉的任何构建和打包工具来创建部署程序包。有关如何使用 Maven 构建工具创建独立 `.jar` 的示例，请参阅[使用 Maven 但不借助任何 IDE 创建 .jar 部署程序包 \(Java\) \(p. 88\)](#)和[使用 Maven 和 Eclipse IDE 创建 .jar 部署程序包 \(Java\) \(p. 90\)](#)。有关如何使用 Gradle 构建工具创建 `.zip` 文件的示例，请参阅[创建 .zip 部署程序包 \(Java\) \(p. 92\)](#)。

确认已创建您的部署程序包后，请转到下一步来创建 IAM 角色（执行角色）。您将在创建 Lambda 函数时指定此角色。

下一步

[步骤 2.2：创建执行角色 \(IAM 角色\) \(p. 220\)](#)

转到

1. 打开文本编辑器，然后复制以下代码。

```
import (
    "strings"
    "github.com/aws/aws-lambda-go/events"
)

func handler(ctx context.Context, events.SNSEvent snsEvent) {
    for _, record := range snsEvent.Records {
        snsRecord := record.SNS

        fmt.Printf("[%s %s] Message = %s \n", record.EventSource, snsRecord.Timestamp,
snsRecord.Message)
    }
}
```

2. 将该文件保存为 `lambda_handler.go`。
3. 将 `lambda_handler.go` 文件压缩为 `LambdaWithSNS.zip`。

下一步

[步骤 2.2：创建执行角色 \(IAM 角色\) \(p. 220\)](#)

Python

1. 打开文本编辑器，然后复制以下代码。

Note

利用 `from __future__` 语句，可以编写与 Python 2 或 3 兼容的代码。如果您使用的是运行时版本 3.6，则不必将它包含在内。

```
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
```

2. 将该文件保存为 `lambda_handler.py`。
3. 将 `lambda_handler.py` 文件压缩为 `LambdaWithSNS.zip`。

下一步

步骤 2.2：创建执行角色（IAM 角色）(p. 220)

步骤 2.2：创建执行角色（IAM 角色）

在本节中，您将使用下面的预定义角色类型和访问策略创建 IAM 角色：

- AWS Lambda 类型的 AWS 服务角色 - 此角色为 AWS Lambda 授予代入 IAM 角色的权限。
- AWSLambdaBasicExecutionRole - 这是您附加到 IAM 角色的访问权限策略。

有关 IAM 角色的更多信息，请参阅 IAM 用户指南 中的 [IAM 角色](#)。使用以下程序创建 IAM 角色。

创建 IAM 角色（执行角色）

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 按照 IAM 用户指南 中[创建角色以向 AWS 服务委派权限](#)的步骤创建 IAM 角色（执行角色）。遵循步骤创建角色时，请注意以下事项：
 - 在 Role Name 中，使用在 AWS 账户内唯一的名称（例如，`lambda-sns-execution-role`）。
 - 在 Select Role Type 中，选择 AWS Service Roles，然后选择 AWS Lambda。这将为 AWS Lambda 服务授予代入 IAM 角色的权限。
 - 在 Attach Policy 中，选择 `AWSLambdaBasicExecutionRole`。此策略中的权限足以用于本教程中的 Lambda 函数。
3. 记下角色 ARN。在下一步中，创建 Lambda 函数时需要用到它。

步骤 3：设置跨账户权限

在此部分中，您使用 CLI 命令跨 Lambda 函数账户和 Amazon SNS 主题账户设置权限，然后测试订阅。

1. 从账户 A 创建 Amazon SNS 主题：

```
aws sns create-topic \
--name lambda-x-account
```

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。

2. 从账户 B 创建 Lambda 函数。对于运行时参数，根据您在创建部署程序包时选择的代码示例来选择 `nodejs6.10`、`nodejs4.3`、`python3.6`、`python2.7` 或 `java8`。

```
aws lambda create-function \
--function-name SNS-X-Account \
```

```
--runtime runtime language \
--role role arn \
--handler handler-name \
--description "SNS X Account Test Function" \
--timeout 60 \
--memory-size 128 \
--zip-file fileb://path/LambdaWithSNS.zip
```

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。

3. 从账户 A 中，将权限添加到账户 B 以订阅主题：

```
aws sns add-permission \
--region us-east-1 \
--topic-arn Amazon SNS topic arn \
--label lambda-access \
--aws-account-id B \
--action-name Subscribe ListSubscriptionsByTopic Receive
```

4. 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 \
--statement-id sns-x-account \
--action "lambda:InvokeFunction" \
--principal sns.amazonaws.com \
--source-arn Amazon SNS topic arn
```

作为响应，Lambda 将返回以下 JSON 代码。Statement 值是已添加到 Lambda 函数策略的语句的 JSON 字符串版本：

```
{
  "Statement": "{\"Condition\": {\"ArnLike\": {\"AWS:SourceArn\": \"arn:aws:lambda:us-east-1:B:function:SNS-X-Account\"}}, \"Action\": [\"lambda:InvokeFunction\"], \"Resource\": \"arn:aws:lambda:us-east-1:A:function:SNS-X-Account\", \"Effect\": \"Allow\", \"Principal\": {\"Service\": \"sns.amazonaws.com\"}, \"Sid\": \"sns-x-account1\"}"
}
```

Note

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。由于源账户包含在源 ARN 中，因此这不会影响安全性。

5. 从账户 B 中，将 Lambda 函数订阅到主题：

```
aws sns subscribe \
--topic-arn Amazon SNS topic arn \
--protocol lambda \
--notification-endpoint arn:aws:lambda:us-east-1:B:function:SNS-X-Account
```

您应该可以看到类似于如下所示的 JSON 输出内容：

```
{
  "SubscriptionArn": "arn:aws:sns:us-east-1:A:lambda-x-account:5d906xxxx-7c8x-45dx-a9dx-0484e31c98xx"
}
```

6. 从账户 A 中，您现在可以测试订阅。在文本文件中键入“Hello World”并将它保存为 message.txt。然后运行以下命令：

```
aws sns publish \
--topic-arn arn:aws:sns:us-east-1:A:lambda-x-account \
--message file://message.txt \
--subject Test
```

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。

Note

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。

有关 Amazon SNS 的更多信息，请参阅[什么是 Amazon Simple Notification Service？](#)

将 AWS Lambda 与 Amazon API Gateway 结合使用（按需并通过 HTTPS）

您可以通过 HTTPS 调用 AWS Lambda 函数。您可以通过以下方式执行此操作：使用 [Amazon API Gateway 定义自定义 REST API 和终端节点](#)，然后将各个方法（如 `GET` 和 `PUT`）映射到特定的 Lambda 函数。或者，您可以添加名为 `ANY` 的特殊方法，将支持的所有方法（`GET`、`POST`、`PATCH`、`DELETE`）映射到 Lambda 函数。当向该 API 终端节点发送 HTTPS 请求时，Amazon API Gateway 服务会调用相应的 Lambda 函数。有关 `ANY` 方法的更多信息，请参阅[步骤 3：使用 Lambda 和 API 网关 创建简单的微服务 \(p. 235\)](#)。

Amazon API Gateway 还在您的应用程序用户和实现以下功能的应用程序逻辑之间添加了一层：

- 限制单个用户或请求。
- 防止分布式拒绝服务攻击。
- 提供缓存层以缓存来自 Lambda 函数的响应。

请注意有关 Amazon API Gateway 和 AWS Lambda 集成的工作原理的以下信息：

- 推事件模型 - 这是一个模型（请参阅[事件源映射 \(p. 142\)](#)），其中 Amazon API Gateway 通过将请求正文中的数据作为参数传递到 Lambda 函数来调用 Lambda 函数。
- 同步调用 - Amazon API Gateway 可通过将 `RequestResponse` 指定为调用类型来调用 Lambda 函数并实时获取响应。有关调用类型的信息，请参阅[调用类型 \(p. 142\)](#)。
- 事件结构 - 您的 Lambda 函数接收的事件是来自 Amazon API Gateway 接收的 HTTPS 请求的正文，您的 Lambda 函数是为处理特定事件类型而写入的自定义代码。

请注意，设置端到端体验时有两种类型的权限策略可供选择：

- 针对 Lambda 函数的权限 - 无论哪个对象调用 Lambda 函数，AWS Lambda 都将通过代入您在创建 Lambda 函数时指定的 IAM 角色（执行角色）来执行该函数。利用与此角色关联的权限策略，您可以向 Lambda 函数授予其所需的权限。例如，如果 Lambda 函数需要读取某个对象，您可以在权限策略中为相关 Amazon S3 操作授予权限。有关更多信息，请参阅[管理权限：使用 IAM 角色（执行角色）\(p. 339\)](#)。
- 供 Amazon API Gateway 调用 Lambda 函数的权限 - Amazon API Gateway 无法在未经您的许可的情况下调用 Lambda 函数。您通过与 Lambda 函数关联的权限策略授予此权限。

有关引导您完成示例设置的教程，请参阅[将 AWS Lambda 与 Amazon API Gateway 结合使用 \(按需并通过 HTTPS \) \(p. 223 \)](#)。

将 AWS Lambda 与 Amazon API Gateway 结合使用 (按需并通过 HTTPS)

在本示例中，您将使用 Amazon API Gateway 创建一个简单 API (DynamoDBOperations)。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 (LambdaFunctionForAPIGateway) 支持。也就是说，当您通过 HTTPS 终端节点调用方法时，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 放置项目操作的示例请求负载：

```
{  
    "operation": "create",  
    "tableName": "LambdaTable",  
    "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": "LambdaTable",  
    "payload": {  
        "Key": {  
            "Id": "1"  
        }  
    }  
}
```

- 下面是 echo 操作的示例请求负载。您随后将使用请求正文中的以下数据将 HTTPS PUT 请求发送到终端节点。

```
{  
    "operation": "echo",  
    "payload": {  
        "somekey1": "somevalue1",  
        "somekey2": "somevalue2"  
    }  
}
```

您也可以从 AWS Lambda 控制台创建和管理 API 终端节点。例如，在蓝图中搜索 microservice。本教程未使用控制台，而是使用了 AWS CLI 来为您提供 API 工作原理的更多详细信息。

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](#)。

下一步

[步骤 1：准备 \(p. 224\)](#)

步骤 1：准备

确保您已完成以下步骤：

- 注册 AWS 账户并在该账户中创建管理员用户（名为 adminuser）。有关说明，请参阅[设置 AWS 账户 \(p. 4\)](#)。
- 安装并设置 AWS CLI。有关说明，请参阅[设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)。

下一步

[步骤 2：创建 Lambda 函数并手动对其进行测试 \(p. 224\)](#)

步骤 2：创建 Lambda 函数并手动对其进行测试

请在此部分中执行以下操作：

- 使用提供的示例代码创建 Lambda 函数部署程序包。
- 创建 IAM 角色（执行角色）。在上传部署程序包时，需要指定 Lambda 可代入的代表您执行该函数的 IAM 角色（执行角色）。
- 创建 Lambda 函数，然后手动对其进行测试。

主题

- [步骤 2.1：创建部署程序包 \(p. 224\)](#)
- [步骤 2.2：创建执行角色（IAM 角色） \(p. 227\)](#)
- [步骤 2.3：创建 Lambda 函数并手动对其进行测试 \(p. 228\)](#)

步骤 2.1：创建部署程序包

从 Filter View 列表中，选择要用于 Lambda 函数的语言。适当的部分随即出现，其中包含创建部署程序包的代码和具体说明。

Node.js

按照以下说明创建一个 AWS Lambda 函数部署程序包。

1. 打开文本编辑器，然后复制以下代码。

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

Note

代码示例与 Node.js 运行时 v6.10 或 v4.3 兼容。有关更多信息，请参阅[编程模型 \(Node.js\) \(p. 18\)](#)

2. 将该文件保存为 `LambdaFunctionOverHttps.js`。
3. 将 `LambdaFunctionOverHttps.js` 文件压缩为 `LambdaFunctionOverHttps.zip`。

下一步

[步骤 2.2：创建执行角色 \(IAM 角色 \) \(p. 227\)](#)

Python

按照以下说明创建一个 AWS Lambda 函数部署程序包。

1. 打开文本编辑器，然后复制以下代码。

Note

利用 `from __future__ import print_function` 语句，可以编写与 Python 2 或 3 兼容的代码。如果您使用的是运行时版本 3.6，则不必将它包含在内。

```
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)')
```

2. 将该文件保存为 `LambdaFunctionOverHttps.py`。
3. 将 `LambdaFunctionOverHttps.py` 文件压缩为 `LambdaFunctionOverHttps.zip`。

下一步

[步骤 2.2：创建执行角色 \(IAM 角色 \) \(p. 227\)](#)

[转到](#)

按照以下说明创建一个 AWS Lambda 函数部署程序包。

1. 打开文本编辑器，然后复制以下代码。

```
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
}
```

2. 将该文件保存为 LambdaFunctionOverHttps.go。
3. 将 LambdaFunctionOverHttps.go 文件压缩为 LambdaFunctionOverHttps.zip。

下一步

[步骤 2.2：创建执行角色（IAM 角色）\(p. 227\)](#)

步骤 2.2：创建执行角色（IAM 角色）

在本节中，您将使用下面的预定义角色类型创建 IAM 角色：

- AWS Lambda 类型的 AWS 服务角色 - 此角色为 AWS Lambda 授予代入 IAM 角色的权限。

有关 IAM 角色的更多信息，请参阅 IAM 用户指南 中的 [IAM 角色](#)。使用以下程序创建 IAM 角色。

创建 IAM 角色（执行角色）

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 按照 IAM 用户指南 中[创建角色以向 AWS 服务委派权限](#)的步骤创建 IAM 角色（执行角色）。遵循步骤 创建角色时，请注意以下事项：
 - 在 Role Name 中，使用在 AWS 账户内唯一的名称（例如，lambda-gateway-execution-role）。
 - 在 Select Role Type 中，选择 AWS Service Roles，然后选择 AWS Lambda。这将为 AWS Lambda 服务授予代入 IAM 角色的权限。
 - 在控制台中创建 IAM 角色，无需附加权限策略。创建 IAM 角色后，更新该角色，然后将以下权限策略附加到该角色。

```
{
    "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"
}
]
```

- 写下角色 ARN (Amazon 资源名称)。在下一步中创建 Lambda 函数时，您将需要此类信息。

下一步

步骤 2.3：创建 Lambda 函数并手动对其进行测试 (p. 228)

步骤 2.3：创建 Lambda 函数并手动对其进行测试

请在此部分中执行以下操作：

- 通过上传部署程序包来创建 Lambda 函数。
- 通过手动调用 Lambda 函数并传递示例事件数据来对其进行测试。

步骤 2.3.1：创建 Lambda 函数（上传部署程序包）

在本步骤中，您将使用 AWS CLI 上传部署程序包。

在命令提示符处，使用 adminuser profile 运行下面的 Lambda CLI `create-function` 命令。

您需要提供 .zip 文件路径和执行角色 ARN 来更新该命令。`--runtime` 参数值可以是 `python3.6`、`python2.7`、`nodejs6.10`、`nodejs4.3` 或 `java8`，具体取决于您编写代码所用的语言。

```
$ aws lambda create-function \
--region region \
--function-name LambdaFunctionOverHttps \
--zip-file file:///file-path/LambdaFunctionOverHttps.zip \
--role execution-role-arn \
--handler LambdaFunctionOverHttps.handler \
--runtime runtime-value \
--profile adminuser
```

或者，您也可以将 .zip 文件上传到同一 AWS 区域中的 Amazon S3 存储桶，然后在之前的命令中指定该存储桶和对象名称。您需要将 `--zip-file` 参数替换为 `--code` 参数，如下所示：

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

Note

您可以使用 AWS Lambda 控制台创建 Lambda 函数，在这种情况下，应记下 `create-function` AWS CLI 命令参数的值。您应在控制台 UI 中提供相同的值。

步骤 2.3.2：测试 Lambda 函数 (手动调用)

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 (AWS 管理控制台)

1. 在[手动调用 Lambda 函数并验证结果、日志和指标 \(p. 11\)](#)中按照入门练习中的步骤创建并调用 Lambda 函数。对于用于测试的示例事件，请在 Sample event template 中选择 Hello World，然后使用以下内容替换该数据：

```
{  
    "operation": "echo",  
    "payload": {  
        "somekey1": "somevalue1",  
        "somekey2": "somevalue2"  
    }  
}
```

2. 要测试 dynamo 操作之一（例如 read），请将输入数据更改为以下内容：

```
{  
    "operation": "read",  
    "tableName": "the name of your stream table",  
    "payload": {  
        "Key": {  
            "the primary key of the table": "the value of the key"  
        }  
    }  
}
```

3. 在控制台中验证结果。

测试 Lambda 函数 (AWS CLI)

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 \  
--invocation-type Event \  
--function-name LambdaFunctionOverHttps \  
--region region \  
--payload file://file-path/input.txt \  
--profile adminuser  
outputfile.txt
```

Note

在本教程示例中，如果您请求同步执行（`RequestResponse` 作为执行类型），则消息将保存在 `outputfile.txt` 文件中。函数在响应正文中返回字符串消息。如果您使用 `Event` 调用类型，则任何消息均不会返回到输出文件。在任一情况下，都需要 `outfile.txt` 参数。

下一步

[步骤 3：使用 Amazon API Gateway 创建 API 并对其进行测试 \(p. 230\)](#)

步骤 3：使用 Amazon API Gateway 创建 API 并对其进行测试

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，将 HTTPS 请求发送到 API 方法后，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。

步骤 3.1：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 \
--region region \
--profile profile
```

以下为响应示例：

```
{  
    "name": "DynamoDBOperations",  
    "id": "api-id",  
    "createdDate": 1447724091  
}
```

记下 API ID。

您还需要 API 根资源的 ID。要获取该 ID，请运行 `get-resources` 命令。

```
$ aws apigateway get-resources \
--rest-api-id api-id
```

以下是示例响应（此时，您仅具有根资源，但您将在下一步中添加更多资源）：

```
{  
    "items": [  
        {  
            "path": "/",  
            "id": "root-id"  
        }  
    ]  
}
```

步骤 3.2：在 API 中创建资源 (`DynamoDBManager`)

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-id \
--parent-id root-id \
--path-part DynamoDBManager
```

以下为响应示例：

```
{  
    "path": "/DynamoDBManager",  
    "pathPart": "DynamoDBManager",  
    "id": "resource-id",  
    "parentId": "root-id"  
}
```

记下响应中的 ID。这是您创建的资源 (DynamoDBManager) 的 ID。

步骤 3.3：在资源上创建方法 (POST)

运行以下 put-method 命令以在您的 API (DynamoDBOperations) 中的资源 (DynamoDBManager) 上创建方法 (POST)。

```
$ aws apigateway put-method \
--rest-api-id api-id \
--resource-id resource-id \
--http-method POST \
--authorization-type 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。

以下为响应示例：

```
{  
    "apiKeyRequired": false,  
    "httpMethod": "POST",  
    "authorizationType": "NONE"  
}
```

步骤 3.4：将 Lambda 函数设置为 POST 方法的目标

运行以下命令来将 Lambda 函数设置为 POST 方法（这是在您发出对 POST 方法终端节点的请求时 Amazon API Gateway 调用的方法）的集成点。

```
$ aws apigateway put-integration \
--rest-api-id api-id \
--resource-id resource-id \
--http-method POST \
--type AWS \
--integration-http-method POST \
--uri arn:aws:apigateway:aws-region:lambda:path/2015-03-31/functions/arn:aws:lambda:aws-region:aws-acct-id:function:your-lambda-function-name/invocations
```

Note

- --rest-api-id 是您在 Amazon API Gateway 中创建的 API (DynamoDBOperations) 的 ID。
- --resource-id 是您在 API 中创建的资源 (DynamoDBManager) 的资源 ID
- --http-method 是 API 网关方法，而 --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。

以下为响应示例：

```
{  
    "httpMethod": "POST",  
    "type": "AWS",  
    "uri": "arn:aws:apigateway:region:lambda:path/2015-03-31/functions/  
arn:aws:lambda:region:aws-acct-id:function:LambdaFunctionForAPIGateway/invocations",  
    "cacheNamespace": "resource-id"  
}
```

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-id \  
--resource-id resource-id \  
--http-method POST \  
--status-code 200 \  
--response-models "{\"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-id \  
--resource-id resource-id \  
--http-method POST \  
--status-code 200 \  
--response-templates "{\"application/json\": \"\"}"
```

步骤 3.5：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-id \  
--stage-name prod
```

以下为响应示例：

```
{  
    "id": "deployment-id",  
    "createdDate": 1447726017  
}
```

步骤 3.6：授予允许 Amazon API Gateway 调用 Lambda 函数的权限

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，以便让 Amazon API Gateway 在您将 HTTPS 请求发送到 POST 方法时调用 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 (`LambdaFunctionForAPIGateway`) 的权限。

```
$ aws lambda add-permission \  
--function-name LambdaFunctionForAPIGateway
```

```
--function-name LambdaFunctionOverHttps \
--statement-id apigateway-test-2 \
--action lambda:InvokeFunction \
--principal apigateway.amazonaws.com \
--source-arn "arn:aws:execute-api:region:aws-acct-id:api-id/*/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。

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:aws-acct-id:api-id/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。

步骤 3.7：测试发送 HTTPS 请求

在本步骤中，您已准备好向 POST 方法终端节点发送 HTTPS 请求。您可使用 Curl 或 Amazon API Gateway 提供的方法 (`test-invoke-method`)。

如果您要测试 Lambda 函数在 DynamoDB 表上支持的操作，则首先需要在 Amazon DynamoDB `LambdaTable` (`Id`) 中创建一个表，其中 `ID` 是字符串类型的哈希键。

如果您测试的是 Lambda 函数支持的 `echo` 和 `ping` 操作，则无需创建 DynamoDB 表。

您可使用 Amazon API Gateway CLI 命令向资源 (`DynamoDBManager`) 终端节点发送 HTTPS 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：

```
{
  "operation": "create",
  "tableName": "LambdaTable",
  "payload": {
    "Item": {
      "Id": "foo",
      "number": 5
    }
  }
}
```

运行 `test-invoke-method` Amazon API Gateway 命令以使用请求正文中的 JSON 代码向资源 (`DynamoDBManager`) 终端节点发送 HTTPS POST 方法请求。

```
$ aws apigateway test-invoke-method \
--rest-api-id api-id \
--resource-id resource-id \
--http-method POST \
--path-with-query-string "" \
--body "{\"operation\":\"create\",\"tableName\":\"LambdaTable\",\"payload\":{\"Item\":{\"Id\":\"1\",\"name\":\"Bob\"}}}"
```

或者，您也可以使用以下 Curl 命令：

```
curl -X POST -d "{\"operation\":\"create\",\"tableName\":\"LambdaTable\",\"payload\":[{\"Item\":{\"Id\":1,\"name\":\"Bob\"}}]"} https://api-id.execute-api.aws-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：

```
{  
  "operation": "echo",  
  "payload": {  
    "somekey1": "somevalue1",  
    "somekey2": "somevalue2"  
  }  
}
```

运行 test-invoke-method Amazon API Gateway CLI 命令以使用请求正文中的 JSON 代码向资源 (DynamoDBManager) 终端节点发送 HTTPS POST 方法请求。

```
$ aws apigateway test-invoke-method \  
--rest-api-id api-id \  
--resource-id resource-id \  
--http-method POST \  
--path-with-query-string "" \  
--body "{\"operation\":\"echo\",\"payload\":{\"somekey1\":\"somevalue1\",\"somekey2\":\"somevalue2\"}}"
```

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-id.execute-api.region.amazonaws.com/prod/  
DynamoDBManager
```

步骤 4：使用 AWS SAM 和 AWS CloudFormation 执行部署

在上一部分，您使用 AWS Lambda API 通过一个部署程序包 ZIP 文件创建并更新了 Lambda 函数。然而，本机制可能不便用于自动执行函数部署步骤，或者协调事件源和下游资源等其他无服务器应用程序元素之间的部署和更新。

您可以使用 AWS CloudFormation 轻松指定、部署和配置无服务器应用程序。AWS CloudFormation 是一项有助于您对 Amazon Web Services 资源进行建模和设置的服务，能使您花费更少的时间来管理这些资源，而将更多的时间用于关注在 AWS 中运行的应用程序上。您可以创建一个描述您所需的所有 AWS 资源 (如 Lambda 函数或 DynamoDB 表) 的模板，并且 AWS CloudFormation 将负责为您预配和配置这些资源。

此外，您也可以使用 AWS 无服务器应用程序模型表明包含无服务器应用程序的资源。Lambda 函数和 API 等资源类型完全受 AWS CloudFormation 支持，能让您更轻松地定义和部署无服务器应用程序。

有关更多信息，请参阅 [部署基于 Lambda 的应用程序 \(p. 264\)](#)。

Amazon API Gateway 应用程序规范文件

以下内容包含该应用程序的 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。

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

```
Resources:  
  LambdaFunctionOverHttps:  
    Type: AWS::Serverless::Function  
    Properties:  
      Handler: handler  
      Runtime: runtime  
      Policies: AmazonDynamoDBFullAccess  
      Events:  
        HttpPost:  
          Type: Api  
          Properties:  
            Path: '/DynamoDBOperations/DynamoDBManager'  
            Method: post
```

部署无服务器应用程序

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[打包和部署 \(p. 286\)](#)。

步骤 3：使用 Lambda 和 API 网关 创建简单的微服务

在本练习中，您将使用 Lambda 控制台创建 Lambda 函数 (MyLambdaMicroservice)，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

下一步

[步骤 3.1：使用 Amazon API Gateway 创建 API \(p. 235\)](#)

步骤 3.1：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：

1. 登录 AWS 管理控制台，然后打开 AWS Lambda 控制台。
2. 选择 Create Lambda function。
3. 在 Select blueprint 页中，选择 microservice-http-endpoint 蓝图。您可以使用 Filter 查找它。
4. Configure triggers 页中将填充 API 网关 触发器。将创建的默认 API 名称为 LambdaMicroservice (您可以根据需要通过 API Name 字段更改此名称)。

Note

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 MyLambdaMicroservice (您的函数名称) 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。

复查您的触发器后，选择 Next。

5. 在 Configure function 页面上，执行以下操作：

a. 查看预配置的 Lambda 函数配置信息，包括：

- Runtime 为 Node.js 6.10
- 提供了用 JavaScript 编写的代码。代码根据调用的方法和提供的负载执行 DynamoDB 操作。
- Handler 显示 index.handler。格式为：filename.handler-function

b. 在 MyLambdaMicroserviceName 中输入函数名称。

c. 在 Role 中，输入将创建的新角色的角色名称。

Note

microservice-http-endpoint 蓝图在 Policy templates 字段中预填充在创建时添加到新角色的“简单微服务”权限策略模板。这会自动将附加到该策略的必要权限添加到您的新角色。有关更多信息，请参阅 [策略模板 \(p. 343\)](#)。

6. 选择 Create function。

下一步

[步骤 3.2：测试发送 HTTPS 请求 \(p. 236\)](#)

步骤 3.2：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 [步骤 3.1：创建启用了流的 DynamoDB 表 \(p. 201\)](#)。

1. 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。
2. 使用以下内容替换现有文本：

```
{  
  "httpMethod": "GET",  
  "queryStringParameters": {  
    "TableName": "MyTable"  
  }  
}
```

3. 输入上面的文本后，选择 Save and test。

下一步

[步骤 3.3：\(可选\) 尝试其他蓝图 \(p. 236\)](#)

步骤 3.3：(可选) 尝试其他蓝图

您可以选择尝试以下练习：

- 您在本入门练习中使用了 hello-world-python 蓝图。此蓝图提供了用 Python 编写的示例代码。还有一个 hello-world 蓝图，该蓝图提供了用 Node.js 编写的类似的 Lambda 函数代码。
- hello-world-python 和 hello-world 蓝图均处理自定义事件。在本入门练习中，您使用了手动创建的示例事件数据。您可编写 Lambda 函数来处理由事件源（如 Amazon S3 和 DynamoDB）发布的事件。这需要控制台中的事件源配置。

例如，您可以编写 Lambda 函数来处理 Amazon S3 事件。然后，将 Amazon S3 配置为事件源以将对象创建事件发布到 AWS Lambda。在将对象上传到您的存储桶时，Amazon S3 会检测该事件并调

用您的 Lambda 函数。您的 Lambda 函数将事件数据作为参数接收。您可以通过在 Lambda 控制台或 CloudWatch 控制台中查看 CloudWatch 日志来验证是否已执行您的 Lambda 函数。

Lambda 控制台提供了蓝图来设置用于处理 Amazon S3 事件的示例 Lambda 函数。在控制台中创建 Lambda 函数时，在 Select blueprint 页面上的 Filter 框中输入 **s3** 以搜索可用蓝图的列表。

有关使用不同的事件源的更多信息，请参阅[使用案例 \(p. 165\)](#)。

下一步

[接下来做什么？\(p. 237\)](#)

接下来做什么？

本入门练习为您提供了使用 AWS Lambda 控制台的方法的概述。

AWS Lambda 函数还可以自动调用以响应其他 AWS 服务（如 Amazon S3 和 DynamoDB）中的事件。Lambda 函数还可以按需通过 HTTPS 进行调用。您还可以根据需要构建自己的自定义事件源并调用 Lambda 函数。

根据您的集成方案，无论您的应用程序需要事件驱动型 Lambda 函数调用还是按需调用，都请参阅以下章节：

- 配合使用 AWS Lambda 和 Amazon S3 ([p. 165](#))
- 配合使用 AWS Lambda 和 Kinesis ([p. 181](#))
- 配合使用 AWS Lambda 和 Amazon DynamoDB ([p. 191](#))
- 配合使用 AWS Lambda 和 AWS CloudTrail ([p. 203](#))
- 将 AWS Lambda 与 Amazon API Gateway 结合使用（按需并通过 HTTPS）([p. 222](#))
- 使用 AWS Lambda 作为移动应用程序后端（自定义事件源：Android）([p. 237](#))

控制台为您提供了多个蓝图，让您快速设置可处理来自这些事件源的事件的示例 Lambda 函数。您可能希望了解控制台中的其他蓝图以开始使用由这些事件源触发的 Lambda 函数。

使用 AWS Lambda 作为移动应用程序后端（自定义事件源：Android）

您可以使用 AWS Lambda 托管移动应用程序的后端逻辑。也就是说，您的某些移动应用程序代码可作为 Lambda 函数运行。这样一来，您可以将最小逻辑放入移动应用程序本身中，从而实现轻松扩展和更新（例如，您只需将代码更新应用于 Lambda 函数，而不必在应用程序客户端中部署代码更新）。

在创建 Lambda 函数后，您可以使用 AWS 移动软件开发工具包（如适用于 Android 的 AWS 软件开发工具包）从移动应用程序调用该函数。有关更多信息，请参阅[用于 Amazon Web Services 的工具](#)。

Note

您也可以使用 Amazon API Gateway 通过 HTTP 调用 Lambda 函数（而不必使用任何 AWS 软件开发工具包）。Amazon API Gateway 在移动用户和实现以下功能的应用程序逻辑之间额外添加了一层：

- 限制单个用户或请求。
- 防止分布式拒绝服务攻击。
- 提供缓存层以缓存来自 Lambda 函数的响应。

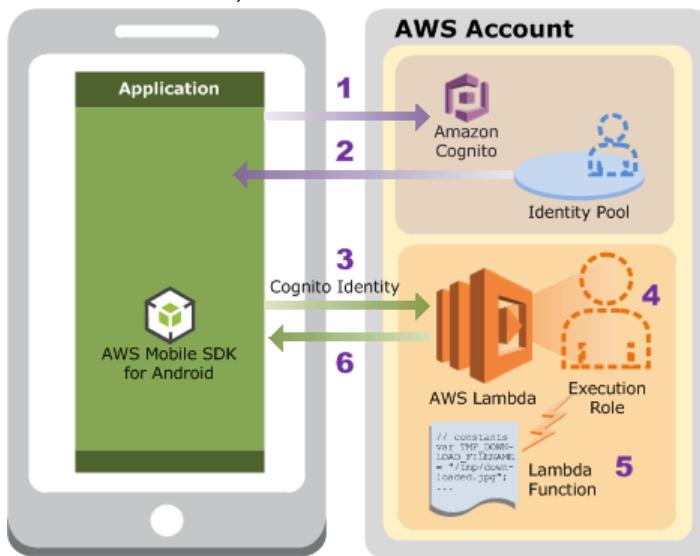
请注意有关移动应用程序和 AWS Lambda 集成的工作原理的以下信息：

- 推事件模型 - 这是一个模型（请参阅[事件源映射 \(p. 142\)](#)），其中应用程序通过将事件数据作为参数传递来调用 Lambda 函数。
- 同步或异步调用 - 该应用程序可通过将 RequestResponse 指定为调用类型（或使用用于异步调用的 Event 调用类型）来调用 Lambda 函数并实时获取响应。有关调用类型的信息，请参阅[管理权限：使用 Lambda 函数策略 \(p. 340\)](#)。
- 事件结构 - 您的 Lambda 函数接收的事件由应用程序定义，您的 Lambda 函数是为处理特定事件类型而写入的自定义代码。

请注意，设置端到端体验时有两种类型的权限策略可供选择：

- 针对 Lambda 函数的权限 - 无论哪个对象调用 Lambda 函数，AWS Lambda 都将通过代入您在创建 Lambda 函数时指定的 IAM 角色（执行角色）来执行该函数。利用与此角色关联的权限策略，您可以向 Lambda 函数授予其所需的权限。例如，如果 Lambda 函数需要读取某个对象，您可以在权限策略中为相关 Amazon S3 操作授予权限。有关更多信息，请参阅[管理权限：使用 IAM 角色（执行角色）\(p. 339\)](#)。
- 供移动应用程序调用 Lambda 函数的权限 - 应用程序必须具有有效安全凭证和权限才能调用 Lambda 函数。对于移动应用程序，您可以使用 Amazon Cognito 服务管理用户身份、身份验证和权限。

下图说明了应用程序的流程（此说明假设了一个使用适用于 Android 的 AWS 移动软件开发工具包进行 API 调用的移动应用程序）：



1. 移动应用程序向 Amazon Cognito 发送了一条请求并在其中附带一个身份池 ID（您在设置的过程中创建身份池）。
2. Amazon Cognito 将临时安全凭证返回到该应用程序。

Amazon Cognito 代入与身份池关联的角色来生成临时凭证。应用程序使用该临时凭证可以执行的操作受限于权限策略（与在获取该凭证时使用的角色 Amazon Cognito 关联）中定义的权限。

Note

AWS 软件开发工具包可以缓存临时凭证，以便应用程序不必在每次需要调用 Lambda 函数时都向 Amazon Cognito 发送请求。

3. 移动应用程序使用临时凭证（Cognito 身份）调用 Lambda 函数。
4. AWS Lambda 代入执行角色以代表您执行您的 Lambda 函数。

5. Lambda 函数执行。
6. AWS Lambda 向移动应用程序返回结果（假定该应用程序使用 RequestResponse 调用类型（同步调用）调用了 Lambda 函数）。

有关引导您完成示例设置的教程，请参阅[教程：使用 AWS Lambda 作为移动应用程序后端 \(p. 239\)](#)。

教程：使用 AWS Lambda 作为移动应用程序后端

在本教程中，您将创建一个简单的 Android 移动应用程序。本教程的主要目的是向您介绍如何挂载各种组件，以使 Android 移动应用程序能够调用 Lambda 函数和处理响应。应用程序本身很简单，我们将做出以下假设：

- 示例移动应用程序将采用以下格式生成包含姓名（名字和姓氏）的事件数据：

```
{ firstName: 'value1', lastName: 'value2' }
```

- 您使用 Lambda 函数处理事件。也就是说，应用程序（使用适用于 Android 的 AWS 移动软件开发工具包）通过将事件数据传递给 Lambda 函数（ExampleAndroidEventProcessor）来调用该函数。本教程中的 Lambda 函数将执行以下操作：
 - 将传入的事件数据记录到 Amazon CloudWatch Logs。
 - 成功执行后，在响应正文返回简单的字符串。您的移动应用程序将使用 Android Toast 类来显示消息。

Note

该移动应用程序调用 Lambda 函数的方式（本教程所示）展示了 AWS Lambda 的请求-响应模型，即：应用程序调用 Lambda 函数，然后实时接收响应。有关更多信息，请参阅[编程模型 \(p. 18\)](#)。

实现摘要

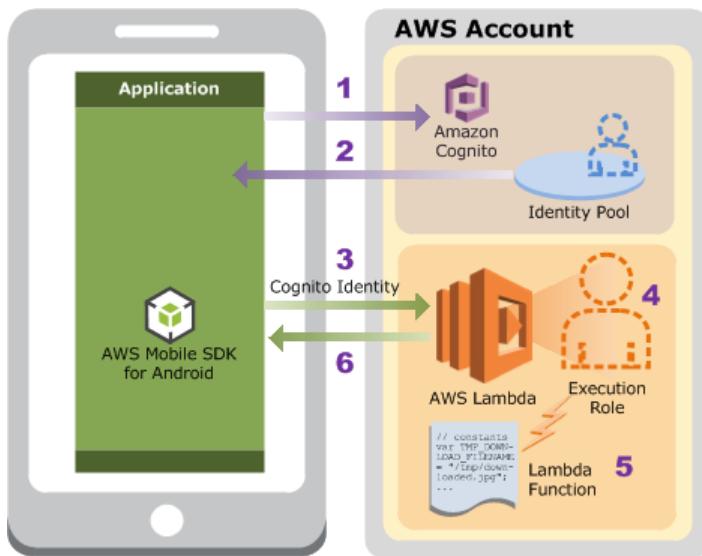
本教程分为两个主要部分：

- 首先，执行创建 Lambda 函数的必要设置，然后使用示例事件数据（不需要移动应用程序来测试 Lambda 函数）手动调用该函数以便对其进行测试。
- 其次，创建 Amazon Cognito 身份池以管理身份验证和权限，并创建该示例 Android 应用程序。然后，运行该应用程序，它将调用 Lambda 函数。您随后可以验证端到端体验。在本教程示例中：
 - 您将使用 Amazon Cognito 服务来管理用户身份、身份验证和权限。移动应用程序必须具有有效的安全凭证和调用 Lambda 函数的权限。在设置应用程序的过程中，您创建一个 Amazon Cognito 身份池来存储用户身份并定义权限。有关更多信息，请参阅[Amazon Cognito](#)
 - 本移动应用程序不需要其用户登录。移动应用程序可以要求用户使用公共身份提供商（如 Amazon 和 Facebook）登录。本教程的范围有限并假定移动应用程序用户未经身份验证。因此，在配置 Amazon Cognito 身份池时，您将执行以下操作：
 - 启用未经身份验证的身份的访问权限。

Amazon Cognito 为这些用户提供了用于调用 Lambda 函数的唯一标识符和临时 AWS 凭证。

- 在与未经身份验证的用户的 IAM 角色关联的访问权限策略中，添加调用 Lambda 函数的权限。一个身份池有两个关联的 IAM 角色，一个用于经过身份验证的应用程序用户，另一个用于未经身份验证的应用程序用户。在本例中，Amazon Cognito 使用用于未经身份验证的用户的角色来获取临时凭证。当应用程序使用这些临时凭证调用 Lambda 函数时，仅当应用程序具有必要权限时才能执行此操作（也就是说，凭证可能有效，但您还需要权限）。您可以通过更新 Amazon Cognito 来获取临时凭证的权限策略来执行此操作。

下图说明应用程序的流程：



您现在可以开始教程。

下一步

[步骤 1：准备 \(p. 240\)](#)

步骤 1：准备

确保您已完成以下步骤：

- 注册 AWS 账户并在该账户中创建管理员用户（名为 adminuser）。有关说明，请参阅 [设置 AWS 账户 \(p. 4\)](#)
- 安装并设置 AWS CLI。有关说明，请参阅 [设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)

Note

本教程在 us-east-1 区域中创建了一个 Lambda 函数和一个 Amazon Cognito 身份池。如果您要使用不同的 AWS 区域，则必须在同一区域中创建这些资源。还需要通过提供要使用的特定区域来更新示例移动应用程序代码。

下一步

[步骤 2：创建 Lambda 函数并手动调用该函数（使用示例事件数据）\(p. 240\)](#)

步骤 2：创建 Lambda 函数并手动调用该函数（使用示例事件数据）

请在此部分中执行以下操作：

- 使用提供的示例代码创建 Lambda 函数部署程序包。用于处理移动应用程序事件的示例 Lambda 函数代码有各种语言版本。选择一种语言并按照相应说明创建部署程序包。

Note

要查看在您的函数中使用其他 AWS 服务的更多示例，包括调用其他 Lambda 函数，请参阅 [AWS SDK for JavaScript](#)

- 创建 IAM 角色（执行角色）。在上传部署程序包时，需要指定 IAM 角色（执行角色）。这是 AWS Lambda 为代表您调用 Lambda 函数而代入的角色。
- 通过上传部署程序包创建 Lambda 函数，然后使用示例事件数据手动调用该函数以便对其进行测试。

主题

- [步骤 2.1：创建部署程序包 \(p. 241\)](#)
- [步骤 2.2：创建执行角色（IAM 角色）\(p. 243\)](#)
- [步骤 2.3：创建 Lambda 函数并手动调用该函数（使用示例事件数据）\(p. 243\)](#)

步骤 2.1：创建部署程序包

从 Filter View 列表中，选择要用于 Lambda 函数的语言。适当的部分随即出现，其中包含创建部署程序包的代码和具体说明。

Node.js

按照以下说明创建一个 AWS Lambda 函数部署程序包。

1. 打开文本编辑器，然后复制以下代码。

```
exports.handler = function(event, context, callback) {
    console.log("Received event: ", event);
    var data = {
        "greetings": "Hello, " + event.firstName + " " + event.lastName + "."
    };
    callback(null, data);
}
```

Note

代码示例与 Node.js 运行时 v6.10 或 v4.3 兼容。有关更多信息，请参阅[编程模型 \(Node.js\) \(p. 18\)](#)

2. 将该文件保存为 `AndroidBackendLambdaFunction.js`。
3. 将 `AndroidBackendLambdaFunction.js` 文件压缩为 `AndroidBackendLambdaFunction.zip`。

下一步

[步骤 2.2：创建执行角色（IAM 角色）\(p. 243\)](#)

Java

使用下面的 Java 代码创建 Lambda 函数 (`AndroidBackendLambdaFunction`)。该代码将接收 Android 应用程序事件数据以作为处理程序的第一个参数。然后，该代码会处理事件数据（为了展示这个过程，该代码会将一些事件数据写入 CloudWatch Logs 并在响应中返回一个字符串）。

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。

Important

在下一节中创建示例移动应用程序时，您将使用相同的类 (POJO) 来处理输入和输出数据。

```
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);
    }
}
```

将上面的代码保存在文件 (HelloPojo.java) 中。您现在可以创建部署程序包。您需要包含以下依赖项：

- aws-lambda-java-core

部署程序包既可以是 .zip 文件，也可以是独立的 .jar。您可以使用自己熟悉的任何构建和打包工具来创建部署程序包。有关如何使用 Maven 构建工具创建独立 .jar 的示例，请参阅[使用 Maven 但不借助任何 IDE 创建 .jar 部署程序包 \(Java\) \(p. 88\)](#)和[使用 Maven 和 Eclipse IDE 创建 .jar 部署程序包 \(Java\) \(p. 90\)](#)。有关如何使用 Gradle 构建工具创建 .zip 文件的示例，请参阅[创建 .zip 部署程序包 \(Java\) \(p. 92\)](#)。

在确认创建完部署程序包 (`lambda-java-example-1.0-SNAPSHOT.jar`) 后，请转至下一节：创建 IAM 角色（执行角色）。您将在创建 Lambda 函数时指定该角色。

下一步

[步骤 2.2：创建执行角色（IAM 角色）\(p. 243\)](#)

步骤 2.2：创建执行角色（IAM 角色）

在本节中，您将使用下面的预定义角色类型和访问策略创建 IAM 角色：

- AWS Lambda 类型的 AWS 服务角色 - 此角色为 AWS Lambda 授予代入 IAM 角色的权限。
- AWSLambdaBasicExecute - 这是您附加到 IAM 角色的访问权限策略。此 Lambda 函数只会将日志写入到 CloudWatch Logs。因此它只需要针对特定 CloudWatch 操作的权限。此策略提供了这些权限。

有关 IAM 角色的更多信息，请参阅 IAM 用户指南 中的 [IAM 角色](#)。使用以下程序创建 IAM 角色。

创建 IAM 角色（执行角色）

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 按照 IAM 用户指南 中[创建角色以向 AWS 服务委派权限](#)的步骤创建 IAM 角色（执行角色）。遵循步骤创建角色时，请注意以下事项：
 - 在 Role Name 中，使用在 AWS 账户内唯一的名称（例如，`lambda-android-execution-role`）。
 - 在 Select Role Type 中，选择 AWS Service Roles，然后选择 AWS Lambda。这将为 AWS Lambda 服务授予代入 IAM 角色的权限。
 - 在 Attach Policy 中，选择 AWSLambdaBasicExecute。此策略中的权限足以用于本教程中的 Lambda 函数。
3. 记下角色 ARN。在下一步中，创建 Lambda 函数时需要用到它。

下一步

[步骤 2.3：创建 Lambda 函数并手动调用该函数（使用示例事件数据）\(p. 243\)](#)

步骤 2.3：创建 Lambda 函数并手动调用该函数（使用示例事件数据）

请在此部分中执行以下操作：

- 通过上传部署程序包来创建 Lambda 函数。
- 通过手动调用 Lambda 函数来对其进行测试。您应使用示例事件数据，而不是创建事件源。在下一节中，您将创建 Android 移动应用程序并测试端到端体验。

步骤 2.3.1：创建 Lambda 函数（上传部署程序包）

在本步骤中，您将使用 AWS CLI 上传部署程序包。

在命令提示符处，使用 `adminuser profile` 运行下面的 Lambda CLI `create-function` 命令。

您需要提供 .zip 文件路径和执行角色 ARN 来更新该命令。`--runtime` 参数值可以是 `nodejs6.10`、`nodejs4.3` 或 `java8`，具体取决于您编写代码所选的语言。

```
$ aws lambda create-function \
--region us-east-1 \
--function-name AndroidBackendLambdaFunction \
--zip-file fileb://file-path-to-jar-or-zip-deployment-package \
--role execution-role-arn \
--handler handler-name \
--runtime runtime-value \
--profile adminuser
```

或者，您也可以将 .zip 文件上传到同一 AWS 区域中的 Amazon S3 存储桶，然后在之前的命令中指定该存储桶和对象名称。您需要将 --zip-file 参数替换为 --code 参数，如下所示：

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

Note

您可以使用 AWS Lambda 控制台创建 Lambda 函数，在这种情况下，应记下 create-function AWS CLI 命令参数的值。您应在控制台 UI 中提供相同的值。

步骤 2.3.2：测试 Lambda 函数（手动调用）

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（AWS 管理控制台）

1. 在[手动调用 Lambda 函数并验证结果、日志和指标 \(p. 11\)](#)中按照入门练习中的步骤创建并调用 Lambda 函数。在选择 Lambda 函数后，从 Actions 菜单中选择 Configure test event 以指定以下示例事件数据：

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

2. 在控制台中验证结果。

- Execution result 应为具有以下返回值的 Succeeded：

```
{ "greetings": "Hello first-name, last-name." }
```

- 查看 Summary 和 Log output 部分。

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 \
--invocation-type Event \
--function-name AndroidBackendLambdaFunction \
--region us-east-1 \
--payload file://file-path/input.txt \
--profile adminuser
outputfile.txt
```

Note

在本教程示例中，消息保存在 `outputfile.txt` 文件中。如果您请求同步执行（`RequestResponse` 作为调用类型），函数将在响应正文中返回字符串消息。对于 Node.js，该消息可能为以下形式之一（您在代码中指定的任何一个）：

```
context.succeed("message")
context.fail("message")
context.done(null, "message")
```

对于 Java，它是返回语句中的消息：

```
return "message"
```

下一步

[步骤 3：创建 Amazon Cognito 身份池 \(p. 245\)](#)

步骤 3：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 角色的更多信息，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. 使用 IAM 用户登录 URL，以 `adminuser` 身份登录 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（应用程序用户是未经身份验证的）。因此，请确保选择 **Enable access to unauthenticated identities** 选项。
 - 未经身份验证的应用程序用户需要调用 Lambda 函数的权限。为此，您应将以下语句添加到与未经身份验证的身份（该语句授予的权限允许对特定 Lambda 函数执行 `lambda:InvokeFunction` 操作）关联的权限策略（您必须通过提供账户 ID 来更新资源 ARN）。

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

Note

您可以在创建身份池时更新策略。您也可以在创建身份池后更新该策略，在这种情况下，请确保在 Amazon Cognito 控制台中记下未经身份验证的用户的 IAM 角色名。然后，转至 IAM 控制台，搜索特定角色并编辑访问权限策略。

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 向导的指示进行操作。

3. 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。

下一步

[步骤 4：创建适用于 Android 的移动应用程序 \(p. 246\)](#)

步骤 4：创建适用于 Android 的移动应用程序

现在，您可以创建一个简单的 Android 移动应用程序，使其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。

1. 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。
2. 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.+'
```

3. 构建项目，以便根据需要下载必需的依赖项。
4. 在 Android 应用程序清单 (`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" />
```

5. 在 `MainActivity` 中，添加以下导入：

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

6. 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。

Note

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。有关此注释的更多信息，请参阅适用于 Android 的 AWS 移动软件开发工具包 Developer Guide 中的 [AWS Lambda](#)。

```
package com.example....lambdaeventgenerator;
import com.amazonaws.mobileconnectors.lambdainvoker.LambdaFunction;
public interface MyInterface {

    /**
     * Invoke the Lambda function "AndroidBackendLambdaFunction".
     * The function name is the method name.
     */
    @LambdaFunction
    ResponseClass AndroidBackendLambdaFunction(RequestClass request);

}
```

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。

将 AWS Lambda 用于计划的事件

您可以创建一个 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 \(p. 252\)](#)。

此功能在您使用 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 Events 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 \  
  --statement-id 'statement_id' \  
  --action 'lambda:InvokeFunction' \  
  --principal events.amazonaws.com \  
  --source-arn arn:aws:events:region:account-id:rule/rule_name \  
  --function-name:function_name \  
  --region region
```

Note

每个 AWS 账户可以有 CloudWatch Events - Schedule 源类型的最多 100 个唯一事件源。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 - Schedule 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。

有关引导您完成示例设置的教程，请参阅[教程：将 AWS Lambda 用于计划的事件 \(p. 249\)](#)。

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。

下一步

步骤 1：创建 Lambda 函数 (p. 250)

步骤 1：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，然后选择 lambda-canary 蓝图。
4. 在基本信息中，为您的函数输入 Name*。
5. 在角色*中，选择选择现有角色。
6. 在现有角色*中，选择lambda_basic_execution。
7. 在 cloudWatch Events 中，选择 Rule 列表，然后选择 Create a new rule。
 - 在 Rule Name 中，键入名称（例如，`CheckWebsiteScheduledEvent`）。
 - 在 Rule description 中，键入说明（例如，`CheckWebsiteScheduledEvent trigger`）。
 - 选择 Schedule expression，然后指定 `rate(1 minute)`。请注意，您可以将值指定为 rate 或用 cron 表达式格式指定值。所有计划都使用 UTC 时区，计划的最小精度为一分钟。

Note

设置 rate 表达式时，会立即进行第一次执行，后续执行根据频率计划发生。在上述示例中，后续执行频率为每分钟执行一次。

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. 252)。

- 在 Enable trigger 中，我们建议您在测试触发器之前将触发器保持在禁用状态。
- 请注意 Lambda function code 部分。这是您创建函数后可以配置的示例代码。此外，控制台还允许您选择 Lambda 支持的运行时并添加自定义代码。

Important

如前所述，在您创建该函数后，便可编辑蓝图中提供的代码。但还请注意，它使用 SITE 和 EXPECTED 变量作为您可以设置的环境变量的占位符，如下所述。

- 在 Environment variables 部分，您可不必更新 Lambda 函数代码而配置应用于函数的设置。在这种情况下，您可以为 site 密钥提供一个 URL 值，并且提供在 expected 密钥中提供一个从该站点返回的预期值。虽然我们强烈建议填充这些值，但如果选择不对此函数使用环境变量，则需要在创建函数之前清除 site 和 expected 字段的 <enter value here> 字段。您还需要更新示例函数代码以将 SITE 和 EXPECTED 变量替换成您选择的文本值。
- 选择 Create function。

Note

一旦创建了 Lambda 函数，您还可以根据您的函数要求添加或更新环境变量部分。有关更多信息，请参阅 环境变量 (p. 354)。

下一步

步骤 2：测试 Lambda 函数（使用示例测试事件）(p. 250)

步骤 2：测试 Lambda 函数（使用示例测试事件）

1. 选择您在上一步中创建的函数，然后选择 Test。

2. 在 Input sample event 页面上，选择 Sample event 列表中的 Scheduled Event。

记下示例事件的事件时间。当 AWS Lambda 按照计划的间隔调用函数时，此值将不同。示例 Lambda 函数代码将此时间记录到 CloudWatch Logs。

3. 选择 Save and test，并确保 Execution result 部分显示成功。

下一步

[步骤 3：创建并订阅 Amazon SNS 主题 \(p. 251\)](#)

步骤 3：创建并订阅 Amazon SNS 主题

1. 使用 Amazon SNS 控制台创建 SNS 主题。有关说明，请参阅 Amazon Simple Notification Service 开发人员指南 中的[创建主题](#)。
2. 订阅至主题。对于此练习，请使用电子邮件作为通信协议。有关说明，请参阅 Amazon Simple Notification Service 开发人员指南 中的[订阅主题](#)。

在下一步中，您将在配置 CloudWatch 警报时使用此 Amazon SNS 主题，以便在 AWS Lambda 发出错误时，该警报将通知发布到此主题。

下一步

[步骤 4：配置 CloudWatch 警报 \(p. 251\)](#)

步骤 4：配置 CloudWatch 警报

要配置 CloudWatch 警报，请按照 Amazon CloudWatch 用户指南 中的[创建警报](#)的说明进行操作。在执行这些步骤时，请注意以下几点：

- 在 Create Alarm (1. Select Metric 步骤) 中，选择 Lambda Metrics，然后为您创建的 Lambda 函数选择 Errors (Metric Name 为 Errors)。另外，在统计数据下拉列表中，将设置从 Average 更改为 Sum 统计数据。
- 在 Create Alarm (2. Define Metric 步骤) 中，将警报阈值设置为 Whenever: Errors is ≥ 1 ，然后从 Send notification to: 列表中选择您的 Amazon SNS 主题。

下一步

[步骤 5：重新测试 Lambda 函数 \(p. 251\)](#)

步骤 5：重新测试 Lambda 函数

现在重新测试 Lambda 函数。这一次，通过指定不存在的网页 URL 或文本字符串来更新代码。这将导致此函数返回一个 AWS Lambda 将发送给 CloudWatch 错误指标的错误。CloudWatch 会将此消息发布到 Amazon SNS 主题，并且您将收到电子邮件通知。

(可选)：使用 AWS SAM 和 AWS CloudFormation 执行部署；

在上一部分，您使用 AWS Lambda API 通过一个部署程序包 ZIP 文件创建并更新了 Lambda 函数。然而，本机制可能不便用于自动执行函数部署步骤，或者协调事件源和下游资源等其他无服务器应用程序元素之间的部署和更新。

您可以使用 AWS CloudFormation 轻松指定、部署和配置无服务器应用程序。AWS CloudFormation 是一项有助于您对 Amazon Web Services 资源进行建模和设置的服务，能使您花费更少的时间来管理这些资源，而将更多的时间用于关注在 AWS 中运行的应用程序上。您可以创建一个描述您所需的所有 AWS 资源（如 Lambda 函数或 DynamoDB 表）的模板，并且 AWS CloudFormation 将负责为您预配和配置这些资源。

此外，您也可以使用 AWS 无服务器应用程序模型表明包含无服务器应用程序的资源。Lambda 函数和 API 等资源类型完全受 AWS CloudFormation 支持，能让您更轻松地定义和部署无服务器应用程序。

有关更多信息，请参阅 [部署基于 Lambda 的应用程序 \(p. 264\)](#)。

预定事件应用程序规范

以下内容包含该应用程序的 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。请确保 Runtime: 参数值与您在上一部分中选择的值匹配。

```
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: String
      MetricName: Errors
      Namespace: AWS/Lambda
      Period: '60'
      Statistic: Sum
      Threshold: '1'
```

部署无服务器应用程序

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 [打包和部署 \(p. 286\)](#)。

使用 Rate 或 Cron 来计划表达式

Rate 表达式

```
rate(Value Unit)
```

其中：

#可以是正整数。

##可以是分钟、小时或天。

例如：

示例	Cron 表达式
每 5 分钟调用一次 Lambda 函数	rate(5 minutes)
每小时调用一次 Lambda 函数	rate(1 hour)
每七天调用一次 Lambda 函数	rate(7 days)

请注意以下几点：

- 不支持少于一分钟的速率频率。
- 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)`）。

Cron 表达式

`cron(Minutes Hours Day-of-month Month Day-of-week Year)`

所有字段都是必填的，并且时区仅为 UTC。下表描述了这些字段。

字段	值	通配符
分钟	0-59	, - * /
小时	0-23	, - * /
日期	1-31	, - * ? / L W
月	1-12 或 JAN-DEC	, - * /
星期几	1-7 或 SUN-SAT	, - * ? / L #
年代	1970-2199	, - * /

下表列出了通配符。

字符	定义	示例
/	指定增量	分钟字段中的 0/15 指示每 15 分钟执行一次。
L	指定“最后一天”	如果用在日期字段中，则指定月中的最后一天。如果用在星期几字段中，则指定一周的最后一天（星期六）。
W	指定工作日	与日期使用时，如 5/w，则指定最近的工作日到月中的第 5 天。如果第 5 天为星球六，则在星期五执行。如果第 5 天为星期日，则在星期一执行。

字符	定义	示例
#	指定月中的第几天	指定 3#2 意味着月中第二个星期二（星期二为一星期的第三天）。
*	指定所有值	如果用在日期字段中，则表示月中的所有日期。
?	无指定值	与另一指定值结合使用。例如，如果指定了一个具体日期，而您并不在意那一天是星期几。
-	指定范围	10-12 表示 10、11 和 12
,	指定其他值	SUN, MON, TUE 表示星期日、星期一和星期二
/	指定增量	5/10 表示 5、15、25、35 等。

下表列出了 cron 表达式的常见示例。

示例	Cron 表达式
每天上午 10:00 (UTC) 调用 Lambda 函数	<code>cron(0 10 * * ? *)</code>
每天中午 12:15 (UTC) 调用 Lambda 函数	<code>cron(15 12 * * ? *)</code>
周一至周五每天下午 6:00 (UTC) 调用 Lambda 函数	<code>cron(0 18 ? * MON-FRI *)</code>
每个月第一天的上午 8:00 (UTC) 调用 Lambda 函数	<code>cron(0 8 1 * ? *)</code>
周一至周五每隔 10 分钟调用一次 Lambda 函数	<code>cron(0/10 * ? * MON-FRI *)</code>
周一至周五的上午 8:00 至下午 5:55 (UTC) 之间每隔 5 分钟调用一次 Lambda 函数	<code>cron(0/5 8-17 ? * MON-FRI *)</code>
每个月的第一个星期一上午 9:00 (UTC) 调用 Lambda 函数	<code>cron(0 9 ? * 2#1 *)</code>

请注意以下几点：

- 不支持产生的速率快于一分钟的 Cron 表达式。
- 日期值或星期几值之一必须是问号 (?)。

将 AWS Lambda 用于自定义用户应用程序

AWS Lambda 的一个使用案例是处理用户应用程序生成的事件。由于只是演示，您不必编写调用 Lambda 函数的用户应用程序。相反，本节中包含的教程将提供了一些示例事件数据，您可以使用这些数据，然后通过它们手动调用 Lambda 函数。

当用户应用程序调用 Lambda 函数时，这就是一个 AWS Lambda 请求-响应 模型的示例，即：应用程序调用 Lambda 函数并实时接收到响应。

有关引导您完成示例设置的教程，请参阅[教程：将 AWS Lambda 用于自定义用户应用程序 \(p. 255\)](#)。

教程：将 AWS Lambda 用于自定义用户应用程序

在本教程中，您将使用 AWS CLI 创建并调用一个 Lambda 函数并探索其他 AWS Lambda API。

您将执行以下操作：

- 创建一个 Lambda 函数来处理它作为参数接收的活动。您将使用以下示例 Node.js 代码来创建 Lambda 函数。

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
    console.log('value1 =', event.key1);
    console.log('value2 =', event.key2);
    console.log('value3 =', event.key3);
    callback(null, "Success");
};
```

Note

代码示例与 Node.js 运行时 v4.3 兼容。有关更多信息，请参阅[编程模型 \(Node.js\) \(p. 18\)](#)

该函数很简单。它对传入的事件数据进行日志记录处理（这些日志在 Amazon CloudWatch 中可用），在请求-响应模型中，您可请求在响应中返回日志数据。

- 模拟一个用户应用程序，该应用程序通过使用以下示例事件数据手动调用 Lambda 函数来向您的 Lambda 函数发送一个事件。

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

Note

本示例类似于入门练习（请参阅[入门 \(p. 3\)](#)）。它们的区别在于入门练习提供了基于控制台的体验。控制台可以为您执行很多操作，从而简化体验。在使用 AWS CLI 时，您可以获得执行 API 调用的体验，这可帮助您更好地了解 AWS Lambda 操作。除创建和调用 Lambda 函数外，您还可探索其他 Lambda API。

下一步

[步骤 1：准备 \(p. 255\)](#)

步骤 1：准备

确保您已完成以下步骤：

- 注册 AWS 账户并在该账户中创建管理员用户（名为 adminuser）。有关说明，请参阅[设置 AWS 账户 \(p. 4\)](#)。
- 安装并设置 AWS CLI。有关说明，请参阅[设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)。

下一步

[步骤 2：创建 Lambda 函数并手动调用该函数 \(p. 256\)](#)

步骤 2：创建 Lambda 函数并手动调用该函数

请在此部分中执行以下操作：

- 创建部署程序包。部署程序包是包含代码和任何依赖项的 .zip 文件。在本教程中，没有依赖项，只有简单的示例代码。
- 创建 IAM 角色（执行角色）。在上传部署程序包时，需要指定 Lambda 可代入的代表您执行该函数的 IAM 角色（执行角色）。

您还为此角色授予您的 Lambda 函数所需的权限。本教程中的代码将日志写入 Amazon CloudWatch Logs。因此，您需要授予执行 CloudWatch 操作的权限。有关更多信息，请参阅 [AWS Lambda Watch 日志](#)。

- 使用 `create-function` CLI 命令创建 Lambda 函数 (`HelloWorld`)。有关底层 API 和相关参数的更多信息，请参阅 [CreateFunction \(p. 387\)](#)。

主题

- [步骤 2.1：创建 Lambda 函数部署程序包 \(p. 256\)](#)
- [步骤 2.2：创建执行角色（IAM 角色）\(p. 257\)](#)
- [步骤 2.3：创建 Lambda 函数 \(p. 257\)](#)
- [下一步 \(p. 258\)](#)

步骤 2.1：创建 Lambda 函数部署程序包

按照以下说明创建一个 AWS Lambda 函数部署程序包。

1. 打开文本编辑器，然后复制以下代码。

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
    console.log('value1 =', event.key1);
    console.log('value2 =', event.key2);
    console.log('value3 =', event.key3);
    callback(null, "Success");
};
```

Note

代码示例与 Node.js 运行时 v6.10 或 v4.3 兼容。有关更多信息，请参阅 [编程模型 \(Node.js\) \(p. 18\)](#)

2. 将该文件保存为 `helloworld.js`。
3. 将 `helloworld.js` 文件压缩为 `helloworld.zip`。

Note

要查看在您的函数中使用其他 AWS 服务的更多示例，包括调用其他 Lambda 函数，请参阅 [AWS SDK for JavaScript](#)

步骤 2.2：创建执行角色 (IAM 角色)

当本教程中的 Lambda 函数执行时，它需要向 Amazon CloudWatch 写入日志的权限。您可以通过创建 IAM 角色（执行角色）来授予这些权限。AWS Lambda 在代表您执行 Lambda 函数时将代入此角色。在本节中，您将使用下面的预定义角色类型和访问策略创建 IAM 角色：

- “AWS Lambda”类型的 AWS 服务角色。此角色授予 AWS Lambda 担任此角色的权限。
- 您附加到角色的“AWSLambdaBasicExecutionRole”访问策略。此现有策略将授予一些权限，包括您的 Lambda 函数需要的、针对 Amazon CloudWatch 操作的权限。

有关 IAM 角色的更多信息，请参阅 IAM 用户指南 中的 [IAM 角色](#)。

在本节中，您将使用以下预定义的角色类型和访问权限策略创建 IAM 角色：

- AWS Lambda 类型的 AWS 服务角色 - 此角色为 AWS Lambda 授予代入 IAM 角色的权限。
- 您附加到 IAM 角色的 AWSLambdaBasicExecutionRole 访问权限策略。

有关 IAM 角色的更多信息，请参阅 IAM 用户指南 中的 [IAM 角色](#)。使用以下程序创建 IAM 角色。

创建 IAM 角色（执行角色）

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 按照 IAM 用户指南 中[创建角色以向 AWS 服务委派权限](#)的步骤创建 IAM 角色（执行角色）。遵循步骤创建角色时，请注意以下事项：
 - 在 Role Name 中，使用在 AWS 账户内唯一的名称（例如，lambda-custom-app-execution-role）。
 - 在 Select Role Type 中，选择 AWS Service Roles，然后选择 AWS Lambda。这将为 AWS Lambda 服务授予代入 IAM 角色的权限。
 - 在 Attach Policy 中，选择 AWSLambdaBasicExecutionRole。
3. 记下角色 ARN。在下一步中，创建 Lambda 函数时需要用到它。

步骤 2.3：创建 Lambda 函数

执行下面的 Lambda CLI `create-function` 命令以创建一个 Lambda 函数。您提供部署程序包和 IAM 角色 ARN 作为参数。请注意，`Runtime` 参数使用 `nodejs6.10`，但您还可以指定 `nodejs4.3`。

```
$ aws lambda create-function \
--region region \
--function-name helloworld \
--zip-file fileb://file-path/helloworld.zip \
--role role-arn \
--handler helloworld.handler \
--runtime nodejs6.10 \
--profile adminuser
```

或者，您也可以将 .zip 文件上传到同一 AWS 区域中的 Amazon S3 存储桶，然后在之前的命令中指定该存储桶和对象名称。您需要将 `--zip-file` 参数替换为 `--code` 参数，如下所示：

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

有关更多信息，请参阅 [CreateFunction \(p. 387\)](#)。AWS Lambda 创建该函数并返回函数配置信息，如下面的示例所示：

```
{
```

```
"FunctionName": "helloworld",
"CodeSize": 351,
"MemorySize": 128,
"FunctionArn": "function-arn",
"Handler": "helloworld.handler",
"Role": "arn:aws:iam::account-id:role/LambdaExecRole",
"Timeout": 3,
"LastModified": "2015-04-07T22:02:58.854+0000",
"Runtime": "nodejs6.10",
"Description": ""
}
```

下一步

[步骤 3：调用 Lambda 函数 \(AWS CLI\) \(p. 258\)](#)

步骤 3：调用 Lambda 函数 (AWS CLI)

在本节中，您将使用调用 AWS CLI 命令手动调用 Lambda 函数。

```
$ aws lambda invoke \
--invocation-type RequestResponse \
--function-name helloworld \
--region region \
--log-type Tail \
--payload '{"key1":"value1", "key2":"value2", "key3":"value3"}' \
--profile adminuser \
outputfile.txt
```

如果您希望您可以将负载保持到一个文件（例如 `input.txt`），则以参数的形式提供文件名。

```
--payload file://input.txt \
```

之前的 `invoke` 命令指定 `RequestResponse` 作为调用类型，这将针对函数执行立即返回一个响应。或者，您也可以指定 `Event` 作为调用类型来异步调用该函数。

通过指定 `--log-type` 参数，该命令还请求由该函数产生的日志的末尾。响应中的日志数据采用 base64 编码，如以下示例响应中所示：

```
{
    "LogResult": "base64-encoded-log",
    "StatusCode": 200
}
```

在 Linux 和 Mac 上，可以使用 `base64` 命令对日志进行解码。

```
$ echo base64-encoded-log | base64 --decode
```

以下是示例日志的已解码的版本。

```
START RequestId: 16d25499-d89f-11e4-9e64-5d70fce44801
2015-04-01T18:44:12.323Z 16d25499-d89f-11e4-9e64-5d70fce44801 value1 = value1
2015-04-01T18:44:12.323Z 16d25499-d89f-11e4-9e64-5d70fce44801 value2 = value2
2015-04-01T18:44:12.323Z 16d25499-d89f-11e4-9e64-5d70fce44801 value3 = value3
2015-04-01T18:44:12.323Z 16d25499-d89f-11e4-9e64-5d70fce44801 result: "value1"
END RequestId: 16d25499-d89f-11e4-9e64-5d70fce44801
REPORT RequestId: 16d25499-d89f-11e4-9e64-5d70fce44801
Duration: 13.35 ms      Billed Duration: 100 ms   Memory Size: 128 MB
```

Max Memory Used: 9 MB

有关更多信息，请参阅 [Invoke \(p. 425\)](#)。

由于使用了 RequestResponse 调用类型来调用该函数，因此该函数在调用时将实时执行并返回您传递给 context.succeed() 的对象。在本教程中，您将看到写入到您在 CLI 命令中指定的 `outputfile.txt` 的以下文本：

"value1"

Note

由于您使用同一个 AWS 账户创建和调用 Lambda 函数，因此您可以执行该函数。但是，如果您要向另一个 AWS 账户授予跨账户的权限或者为另一个 AWS 服务授予执行该函数的权限，则必须向与该函数关联的访问权限策略添加权限。使用 Amazon S3 作为事件源（请参阅[教程：将 AWS Lambda 与 Amazon S3 结合使用 \(p. 166\)](#)）的 Amazon S3 教程向 Amazon S3 授予此类权限来调用该函数。

您可以在 AWS Lambda 控制台中监控 Lambda 函数的活动。

- 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/>。

AWS Lambda 控制台在函数的 Cloudwatch Metrics at a glance 部分中显示某些 CloudWatch 指标的图表化表示。

- 对于每个图表，您还可以选择 logs 链接来直接查看 CloudWatch 日志。

下一步

[步骤 4：尝试更多 CLI 命令 \(AWS CLI\) \(p. 259\)](#)

步骤 4：尝试更多 CLI 命令 (AWS CLI)

步骤 4.1：列出您的账户中的 Lambda 函数

在本部分中，您将尝试 AWS Lambda 列表函数操作。请执行以下 AWS CLI `list-functions` 命令来检索您上传的函数的列表。

```
$ aws lambda list-functions \
--max-items 10 \
--profile adminuser
```

为了演示分页功能的使用，该命令指定可选的 `--max-items` 参数来限制在响应中返回的函数的数量。有关更多信息，请参阅 [ListFunctions \(p. 439\)](#)。以下是一个示例响应。

```
{
  "Functions": [
    {
      "FunctionName": "helloworld",
      "MemorySize": 128,
      "CodeSize": 412,
      "FunctionArn": "arn:aws:lambda:us-east-1:account-id:function:ProcessKinesisRecords",
      "Handler": "ProcessKinesisRecords.handler",
      "Role": "arn:aws:iam::account-id:role/LambdaExecRole",
      "Timeout": 3,
      "LastModified": "2015-02-22T21:03:01.172+0000",
```

```
        "Runtime": "nodejs6.10",
        "Description": ""
    },
    {
        "FunctionName": "ProcessKinesisRecords",
        "MemorySize": 128,
        "CodeSize": 412,
        "FunctionArn": "arn:aws:lambda:us-east-1:account-id:function:ProcessKinesisRecords",
        "Handler": "ProcessKinesisRecords.handler",
        "Role": "arn:aws:iam::account-id:role/lambda-execute-test-kinesis",
        "Timeout": 3,
        "LastModified": "2015-02-22T21:03:01.172+0000",
        "Runtime": "nodejs6.10",
        "Description": ""
    },
    ...
],
    "NextMarker": null
}
```

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，`NextMarker` 将提供一个您可以在下一个 `list-functions` 请求中使用的标记；否则，该值为空。以下 `list-functions` AWS CLI 命令是一个演示 `--next-marker` 参数的示例。

```
$ aws lambda list-functions \
--max-items 10 \
--marker value-of-NextMarker-from-previous-response \
--profile adminuser
```

步骤 4.2：获取元数据和 URL 以下载之前上传的 Lambda 函数部署程序包

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 .zip 文件（部署程序包）的预签名 URL。该 .zip 文件是您上传的用于创建函数的压缩文件。有关更多信息，请参阅 [GetFunction \(p. 413\)](#)。

```
$ aws lambda get-function \
--function-name helloworld \
--region region \
--profile adminuser
```

以下是一个示例响应。

```
{
    "Code": {
        "RepositoryType": "S3",
        "Location": "pre-signed-url"
    },
    "Configuration": {
        "FunctionName": "helloworld",
        "MemorySize": 128,
        "CodeSize": 287,
        "FunctionArn": "arn:aws:lambda:us-west-2:account-id:function:helloworld",
        "Handler": "helloworld.handler",
        "Role": "arn:aws:iam::account-id:role/LambdaExecRole",
        "Timeout": 3,
        "LastModified": "2015-04-07T22:02:58.854+0000",
        "Runtime": "nodejs6.10",
        "Description": ""
    }
}
```

}

如果您仅需要函数配置信息（而不需要预签名 URL），则可以使用 Lambda CLI `get-function-configuration` 命令。

```
$ aws lambda get-function-configuration \
--function-name helloworld \
--region region \
--profile adminuser
```

下一步

[步骤 5：删除 Lambda 函数和 IAM 角色 \(AWS CLI\) \(p. 261\)](#)

步骤 5：删除 Lambda 函数和 IAM 角色 (AWS CLI)

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

```
$ aws lambda delete-function \
--function-name helloworld \
--region region \
--profile adminuser
```

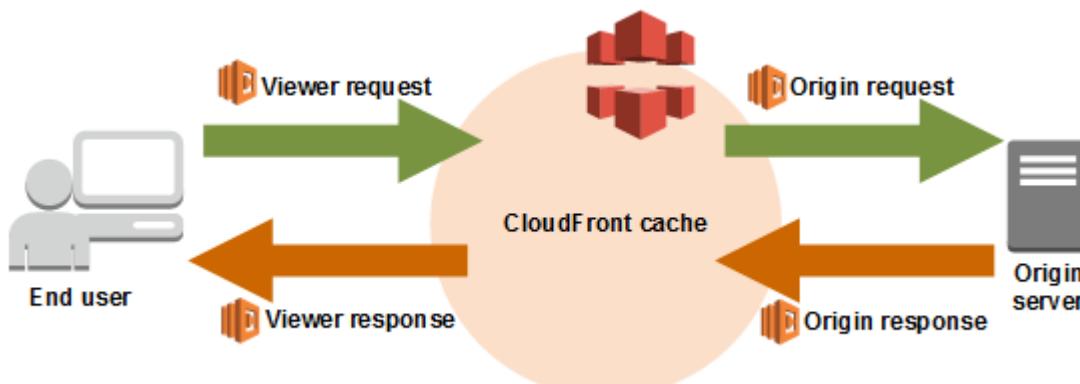
删除 IAM 角色

在删除 Lambda 函数之后，还可以删除您在 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@Edge

Lambda@Edge 允许您运行 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)



您也可以生成对查看器的响应，而不必将请求发送到源。

在 Node.js 6.10 中编写 CloudFront 的 Lambda 函数。使用 Lambda@Edge，您可构建多种解决方案，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](#)。

后续链接与已移动到的 Amazon CloudFront 开发人员指南 中的[将 CloudFront 与 Lambda@Edge 一起使用](#)的内容有关。如果您已对此内容添加了书签，则建议您更新这些书签。以下各部分将为您提供更新的链接。

主题

- [如何为 Lambda@Edge 创建和使用 Lambda 函数 \(p. 262\)](#)
- [如何删除 Lambda 函数的副本 \(p. 262\)](#)
- [设置 Lambda@Edge 的 IAM 权限和角色 \(p. 262\)](#)
- [创建 Lambda@Edge 函数 \(p. 262\)](#)
- [为 Lambda@Edge 函数添加触发器 \(AWS Lambda 控制台\) \(p. 262\)](#)
- [为 Lambda@Edge 编写函数 \(p. 263\)](#)
- [为 Lambda@Edge 编辑 Lambda 函数 \(p. 263\)](#)
- [测试和调试 \(p. 263\)](#)
- [Lambda@Edge 限制 \(p. 263\)](#)

如何为 Lambda@Edge 创建和使用 Lambda 函数

此内容已移至 Amazon CloudFront 开发人员指南 中的[如何创建 Lambda 函数并将其用于 Lambda@Edge](#)。

如何删除 Lambda 函数的副本

此内容已移至 Amazon CloudFront 开发人员指南 中的[如何删除 Lambda 删除的副本](#)。

设置 Lambda@Edge 的 IAM 权限和角色

此内容已移至 Amazon CloudFront 开发人员指南 中的[为 Lambda@Edge 设置 IAM 权限和角色](#)。

创建 Lambda@Edge 函数

此内容已移至 Amazon CloudFront 开发人员指南 中的[创建 Lambda@Edge 函数](#)。

为 Lambda@Edge 函数添加触发器 (AWS Lambda 控制台)

此内容已移至 Amazon CloudFront 开发人员指南 中的[为 Lambda@Edge 函数添加触发器 \(AWS Lambda 控制台\)](#)。

为 Lambda@Edge 编写函数

此内容已移至 Amazon CloudFront 开发人员指南 中的为 Lambda@Edge 编写函数。

为 Lambda@Edge 编辑 Lambda 函数

此内容已移至 Amazon CloudFront 开发人员指南 中的为 Lambda@Edge 编辑 Lambda 函数。

测试和调试

此内容已移至 Amazon CloudFront 开发人员指南 中的测试和调试。

Lambda@Edge 限制

此内容已移至 Amazon CloudFront 开发人员指南 中的 Lambda@Edge 限制。

部署基于 Lambda 的应用程序

基于 Lambda 的应用程序 (也称作无服务器应用程序) 由通过事件触发的函数组成。典型的无服务器应用程序包含一个或多个通过事件 (如向 Amazon S3 上传对象、Amazon SNS 通知和 API 操作) 触发的函数。这些函数既可独立运行，也可利用其他资源 (如 DynamoDB 表或 Amazon S3 存储桶)。最基本的无服务器应用程序仅包含一个函数。

AWS Lambda 提供您可以使用的 API 操作，以便以 ZIP 文件格式提供部署程序包来创建和更新 Lambda 函数。然而，本机制可能不方便用于自动执行函数部署步骤，或者协调事件源和下游资源等其他无服务器应用程序元素之间的部署和更新。例如，为了部署 Amazon SNS 触发器，您需要更新函数、Amazon SNS 主题、函数与主题之间的映射，以及 DynamoDB 表等函数所需的任何其他下游资源。

您可以通过以下方式部署您的无服务器应用程序：

- AWS CLI - 使用 `aws cloudformation deploy` 命令。有关更多信息，请参阅[部署 \(p. 287\)](#)，这部分内容包含在有关创建无服务器应用程序的教程中。有关更多信息，请参阅[创建您自己的无服务器应用程序 \(p. 286\)](#)。Lambda 还提供了其他 AWS CLI 操作，供您用于部署您的无服务器应用程序：
 - [CreateFunction \(p. 387\)](#)
 - [UpdateFunctionConfiguration \(p. 477\)](#)
- AWS CloudFormation - 您可以使用 AWS CloudFormation 指定、部署和配置无服务器应用程序。AWS CloudFormation 是一项服务，可帮助您对 AWS 资源进行建模和设置，以便能花较少的时间管理这些资源，而将更多的时间花在运行于 AWS 中的应用程序上。您可以创建一个描述您所需的所有 AWS 资源 (如 Lambda 函数或 DynamoDB 表) 的模板，并且 AWS CloudFormation 将负责为您预配和配置这些资源。您无需单独创建和配置 AWS 资源，且无需了解 what—AWS CloudFormation 处理所有这些工作时所依赖的内容。有关更多信息，请参阅AWS CloudFormation 用户指南中的[AWS CloudFormation 概念](#)。
- AWS SAM - AWS SAM 支持一些特殊资源类型，可简化如何表达无服务器应用程序的函数、API、映射和 DynamoDB 表；还支持这些服务的一些功能，例如环境变量。这些资源的 AWS CloudFormation 描述符合[AWS 无服务器应用程序模型](#)。为了部署您的应用程序，只需在 AWS CloudFormation 模板文件 (在 JSON 或 YAML 中写入) 中指定您想要作为应用程序一部分的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[使用 AWS 无服务器应用程序模型 \(AWS SAM\) \(p. 281\)](#)。

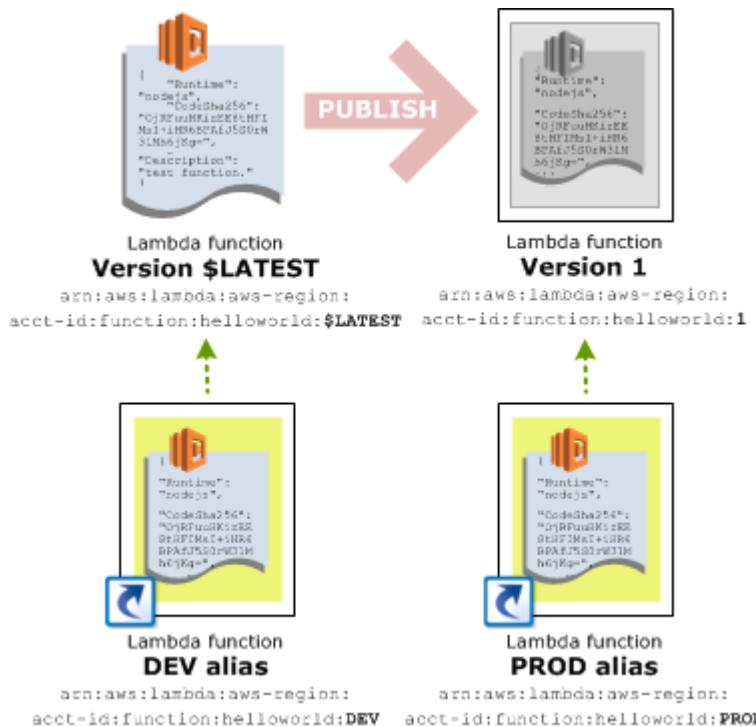
在您了解 AWS 无服务器应用程序模型 (AWS SAM) 之前，我们建议您阅读以下部分，了解 Lambda 函数版本控制、别名以及如何将流量转移到函数修订，这是无服务器应用程序开发的关键部分。有关更多信息，请参阅[AWS Lambda 函数版本控制和别名 \(p. 264\)](#)。

AWS Lambda 函数版本控制和别名

通过使用版本控制，您可以在 AWS Lambda 中更好地管理生产环境中的函数代码。在 AWS Lambda 中使用版本控制时，您可以发布 Lambda 函数的一个或多个版本。因此，您可在开发工作流 (例如开发、测试和生产) 中使用 Lambda 函数的不同变型。

每个 Lambda 函数版本具有唯一的 Amazon 资源名称 (ARN)。在发布某个版本后，它是不可变的 (即无法更改)。

AWS Lambda 还支持为每个 Lambda 函数版本创建别名。从概念上讲，AWS Lambda 别名是指向特定 Lambda 函数版本的指针。它也是类似于 Lambda 函数的资源，每个别名具有唯一的 ARN。每个别名为它指向的函数版本保留一个 ARN。别名只能指向函数版本，而不能指向其他别名。与版本 (不可变) 不同，别名是可变的 (即，可以更改别名)。您可以更新别名以指向不同的函数版本。



利用别名，可以从 Lambda 函数版本的映射及其事件源中抽象化将新的 Lambda 函数版本提升到生产中的过程。

例如，假设 Amazon S3 是在存储桶中创建新对象时调用您的 Lambda 函数的事件源。当 Amazon S3 为事件源时，您将事件源映射信息存储在存储桶通知配置中。在该配置中，您可以指定 Amazon S3 可调用的 Lambda 函数 ARN。不过，在这种情况下，每次发布新的 Lambda 函数版本时，您都需要更新通知配置以便 Amazon S3 调用正确的版本。

相反，假设您在通知配置中指定别名 ARN（例如，PROD 别名 ARN），而不是指定函数 ARN。在将新的 Lambda 函数版本提升到生产环境时，您只需要更新 PROD 别名以指向最新的稳定版本。您不需要在 Amazon S3 中更新通知配置。

当您需要回滚到 Lambda 函数之前的版本时，也同样适用。在该方案中，您只需要更新 PROD 别名以指向不同的函数版本。不需要更新事件源映射。

在构建涉及多个依赖项和开发人员的应用程序时，我们建议您使用版本控制和别名以部署 Lambda 函数。

有关详细信息，请参阅以下主题。

主题

- [AWS Lambda 版本控制简介 \(p. 265\)](#)
- [AWS Lambda 别名简介 \(p. 269\)](#)
- [版本控制、别名和资源策略 \(p. 276\)](#)
- [使用 AWS 管理控制台、AWS CLI 或 Lambda API 操作管理版本控制 \(p. 278\)](#)
- [使用别名的流量转移 \(p. 280\)](#)

AWS Lambda 版本控制简介

接下来，您可以了解如何创建 Lambda 函数并从该函数中发布版本。您还可以了解如何在具有一个或多个发布的版本时更新函数代码和配置信息。此外，您还可以找到有关如何删除函数版本（特定版本或整个 Lambda 函数及其所有版本和关联别名）的信息。

创建 Lambda 函数 (\$LATEST 版本)

在创建 Lambda 函数时，只有一个版本，即，\$LATEST 版本。



Lambda function
Version \$LATEST
arn:aws:lambda:aws-region:
acct-id:function:helloworld:\$LATEST
arn:aws:lambda:aws-region:
acct-id:function:helloworld

您可使用此函数的 Amazon 资源名称 (ARN) 来引用它。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。

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。有关更多信息，请参阅 [AWS Lambda 别名简介 \(p. 269\)](#)。

非限定的 ARN 具有其自己的资源策略。

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

Note

除非您选择发布版本，否则，\$LATEST 函数版本是您拥有的唯一 Lambda 函数版本。您可以在事件源映射中使用限定或非限定 ARN 以调用 \$LATEST 版本。

以下是 CreateFunction API 调用的示例响应。

```
{  
    "CodeSize": 287,  
    "Description": "test function."  
    "FunctionArn": "arn:aws:lambda:aws-region:acct-id:function:helloworld",  
    "FunctionName": "helloworld",  
    "Handler": "helloworld.handler",  
    "LastModified": "2015-07-16T00:34:31.322+0000",  
    "MemorySize": 128,  
    "Role": "arn:aws:iam::acct-id:role/lambda_basic_execution",  
    "Runtime": "nodejs6.10",  
    "Timeout": 3,  
    "CodeSHA256": "OjRFuuHKizEE8tHFIMsI+iHR6BPAfJ5S0rW31Mh6jKg=",  
    "Version": "$LATEST"  
}
```

有关更多信息，请参阅 [CreateFunction \(p. 387\)](#)。

在该响应中，AWS Lambda 返回新创建的函数的非限定 ARN 及其版本 \$LATEST。此响应还显示 Version 为 \$LATEST。CodeSha256 是您上传的部署程序包的校验和。

发布 AWS Lambda 函数版本

当您发布版本时，AWS Lambda 在 \$LATEST 版本中创建了 Lambda 函数代码的快照副本（和配置）。已发布的版本是不可变的。也就是说，您无法更改代码或配置信息。新版本具有包含版本号后缀的唯一 ARN，如下所示。



您可以使用任何以下方法发布版本：

- 显式发布版本 – 您可以使用 PublishVersion API 操作显式发布版本。有关更多信息，请参阅 [PublishVersion \(p. 447\)](#)。该操作使用 \$LATEST 版本中的代码和配置创建新的版本。
- 在创建或更新 Lambda 函数时发布版本 – 也可以使用 CreateFunction 或 UpdateFunctionCode 请求并在请求中添加可选的 publish 参数以发布版本：
 - 在 CreateFunction 请求中指定 publish 参数以创建新的 Lambda 函数 (\$LATEST 版本)。然后，您可以立即创建一个快照并将其指定为版本 1 以发布新函数。有关 CreateFunction 的更多信息，请参阅 [CreateFunction \(p. 387\)](#)。
 - 在 UpdateFunctionCode 请求中指定 publish 参数以更新 \$LATEST 版本中的代码。然后，您可以从 \$LATEST 中发布版本。有关 UpdateFunctionCode 的更多信息，请参阅 [UpdateFunctionCode \(p. 470\)](#)。

如果在创建 Lambda 函数时指定 publish 参数，AWS Lambda 在响应中返回的函数配置信息将显示新发布的版本的版本号。在以下示例中，版本为 1。

```
{  
    "CodeSize": 287,  
    "Description": "test function."  
    "FunctionArn": "arn:aws:lambda:aws-region:acct-id:function:helloworld",  
    "FunctionName": "helloworld",  
    "Handler": "helloworld.handler",  
    "LastModified": "2015-07-16T00:34:31.322+0000",  
    "MemorySize": 128,  
    "Role": "arn:aws:iam::acct-id:role/lambda_basic_execution",  
    "Runtime": "nodejs6.10",  
    "Timeout": 3,  
    "CodeSHA256": "OjRFuuHKizEE8tHFIMsI+iHR6BPAfJ5S0rW31Mh6jKg=",  
    "Version": "1"  
}
```

Note

只有在尚未发布代码或者代码相对于 \$LATEST 版本发生了变化时，Lambda 才会发布新版本。如果未进行更改，将返回发布的 \$LATEST 版本。

我们建议您在创建 Lambda 函数或更新 Lambda 函数代码的同时发布版本。在多个开发人员参与同一 Lambda 函数开发时，该建议尤其适用。可以在请求中使用 publish 参数来执行此操作。

在多个开发人员处理某个项目时，可能会存在以下情况：开发人员 A 创建一个 Lambda 函数 (\$LATEST 版本)。在开发人员 A 发布该版本之前，开发人员 B 可能会更新与 \$LATEST 版本关联的代码 (部署程序包)。在此情况下，您会丢失开发人员 A 上传的原始代码。如果两个开发人员都添加了 publish 参数，则会防止出现上述的争用情况。

Note

发布的版本是不可变的。也就是说，您无法更改与版本关联的代码或配置信息。

Lambda 函数的每个版本均为具有 Amazon 资源名称 (ARN) 的唯一资源。以下示例显示了 helloworld Lambda 函数的版本号 1 的 ARN。

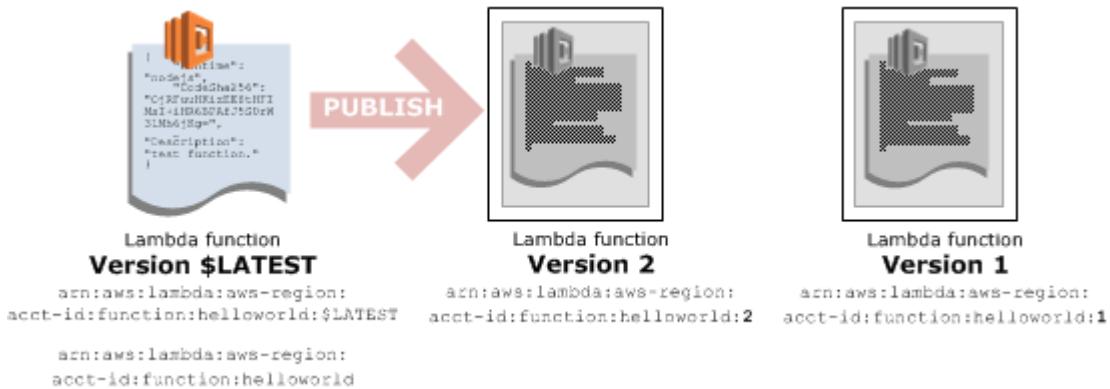
```
arn:aws:lambda:aws-region:acct-id:function:helloworld:1
```

Note

该 ARN 是限定的，其中版本号为后缀。发布的版本只能有一个限定的 ARN。

您可以为 Lambda 函数发布多个版本。当您发布版本时，AWS Lambda 会复制 \$LATEST 版本（代码和配置信息）以创建新版本。在发布额外的版本时，AWS Lambda 分配单调递增的序列号以进行版本控制，即使已删除并重新创建该函数也是如此。绝不会重复使用版本号，即使是已删除并重新创建的函数。这种方法意味着，函数版本的使用者确信该版本的可执行文件绝不会发生变化 (除非将其删除)。

如果要重复使用某个限定词，请在您的版本中使用别名。别名可以删除并且可使用同一名称重新创建。



更新 Lambda 函数代码和配置

AWS Lambda 在 \$LATEST 版本中保留最新的函数代码。在更新函数代码时，AWS Lambda 会替换 Lambda 函数的 \$LATEST 版本中的代码。有关更多信息，请参阅[UpdateFunctionCode \(p. 470\)](#)。

发布的版本是不可变的。您无法更新与发布的版本关联的代码或配置信息。

对于在更新 Lambda 函数代码时发布新版本，您有以下选择：

- 在相同的更新代码请求中发布版本 – 使用 `UpdateFunctionCode` API 操作 (建议)。

- 先更新代码，然后显式发布版本 – 使用 PublishVersion API 操作。

您可以更新 Lambda 函数的 \$LATEST 版本的代码和配置信息（例如，描述、内存大小和执行超时）。但是，发布的版本是不可变的。也就是说，您无法更改代码或配置信息。

删除 Lambda 函数和特定版本

利用版本控制，您有以下选择：

- 删除特定版本 - 您可通过在 DeleteFunction 请求中指定要删除的版本来删除 Lambda 函数版本。如果具有依赖于该版本的别名，请求将失败。AWS Lambda 仅在没有依赖此版本的别名时删除此版本。有关别名的更多信息，请参阅[AWS Lambda 别名简介 \(p. 269\)](#)。
- 删除整个 Lambda 函数（及其所有版本和别名）– 要删除 Lambda 函数及其所有版本，请不要在 DeleteFunction 请求中指定任何版本。这样做将删除整个函数，包括它的所有版本和别名。

Important

可以删除特定函数版本，但无法删除 \$LATEST。

相关主题

[AWS Lambda 别名简介 \(p. 269\)](#)

[使用 AWS 管理控制台、AWS CLI 或 Lambda API 操作管理版本控制 \(p. 278\)](#)

AWS Lambda 别名简介

您可以为 Lambda 函数创建一个或多个别名。AWS Lambda 别名类似于指向特定 Lambda 函数版本的指针。有关版本控制的更多信息，请参阅[AWS Lambda 版本控制简介 \(p. 265\)](#)。

通过使用别名，您可以访问别名指向的 Lambda 函数（例如，调用函数），而调用方无需了解别名指向的特定版本。

AWS Lambda 别名支持以下使用案例：

- 根据需要，更轻松地支持提升和回滚新的 Lambda 函数版本 – 在最初创建 Lambda 函数（\$LATEST 版本）后，您可以发布该函数的版本 1。通过创建名为 PROD 的指向版本 1 的别名，现在可以使用 PROD 别名调用 Lambda 函数的版本 1。

现在您可以使用所有改进来更新代码（\$LATEST 版本），然后发布另一个改进后的稳定版本（版本 2）。可以通过重新映射 PROD 别名以使其指向版本 2，来将版本 2 提升到生产中。如果发现问题，您可以通过重新映射 PROD 别名以使其指向版本 1，来轻松将生产版本回滚到版本 1。

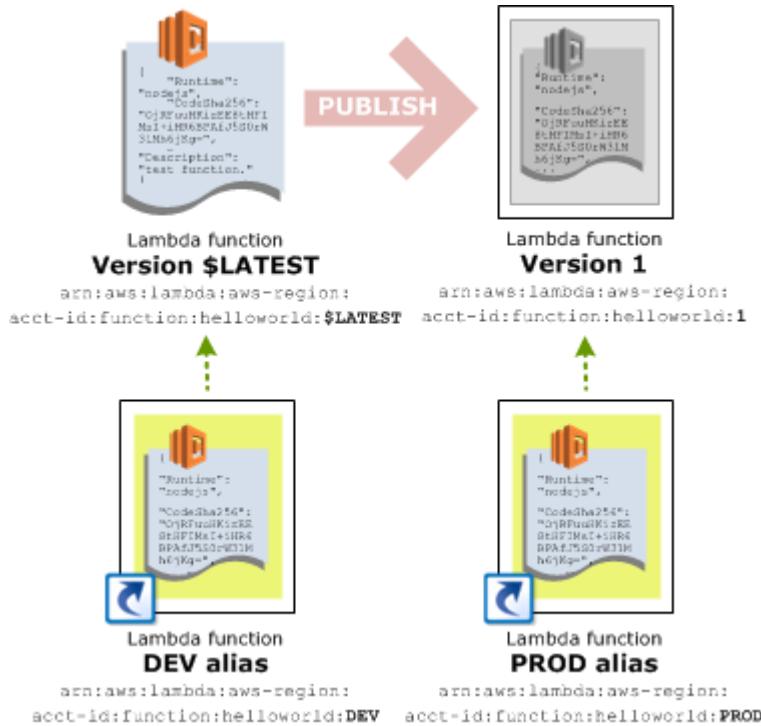
Note

在此情况下，术语提升 和回滚 指的是将别名重新映射到其他函数版本。

- 简化事件源映射管理 – 您可以在事件源映射中使用别名 ARN，而不是使用 Lambda 函数的 Amazon 资源名称（ARN）。这种方法意味着，在提升新版本或回滚到以前的版本时，您不需要更新事件源映射。

AWS Lambda 别名是类似于 Lambda 函数的资源。但无法单独创建别名。您可以为现有 Lambda 函数创建别名。如果 Lambda 函数是资源，则可将 AWS Lambda 别名视为与 Lambda 函数关联的子资源。

Lambda 函数和别名都是 AWS Lambda 资源；与所有其他 AWS 资源类似，它们均具有唯一的 ARN。以下示例显示了一个 Lambda 函数（\$LATEST 版本）与一个发布的版本。每个版本均有一个指向它的别名。



可以使用函数 ARN 或别名 ARN 访问函数。

- 由于非限定函数的函数版本始终映射到 \$LATEST，因此，您可以使用限定或非限定函数 ARN 访问该版本。下面显示了一个具有 \$LATEST 版本后缀的限定函数 ARN。

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

- 当使用任一别名时，您使用的是限定的 ARN。每个别名 ARN 均具有别名后缀。

arn:aws:lambda:aws-region:acct-id:function:helloworld:PROD
arn:aws:lambda:aws-region:acct-id:function:helloworld:BETA
arn:aws:lambda:aws-region:acct-id:function:helloworld:DEV

AWS Lambda 提供了以下 API 操作以创建和管理别名：

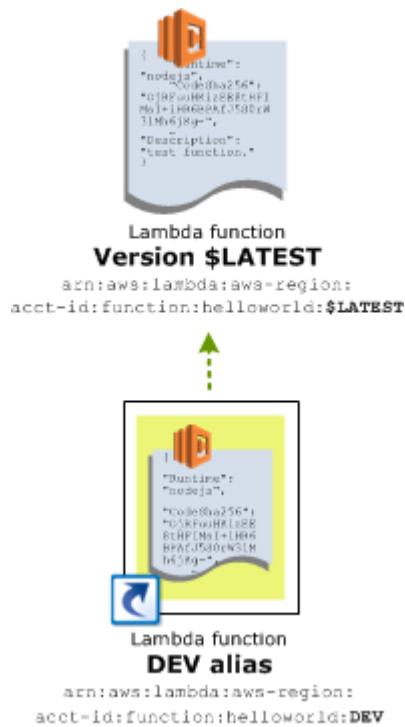
- [CreateAlias \(p. 378\)](#)
- [UpdateAlias \(p. 462\)](#)
- [GetAlias \(p. 407\)](#)
- [ListAliases \(p. 433\)](#)
- [DeleteAlias \(p. 395\)](#)

示例：使用别名管理 Lambda 函数版本

以下是如何使用版本控制和别名将 Lambda 函数的新版本提升到生产中的示例方案。

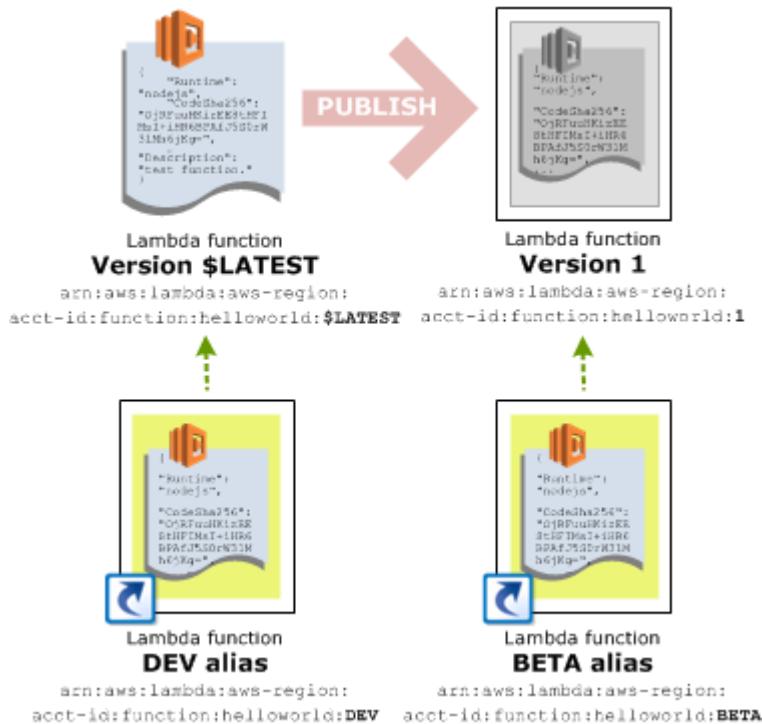
最初，您创建一个 Lambda 函数。

创建的函数是 \$LATEST 版本。您也可以创建指向新创建的函数的别名（DEV，用于开发）。开发人员可使用此别名在开发环境中通过事件源测试函数。



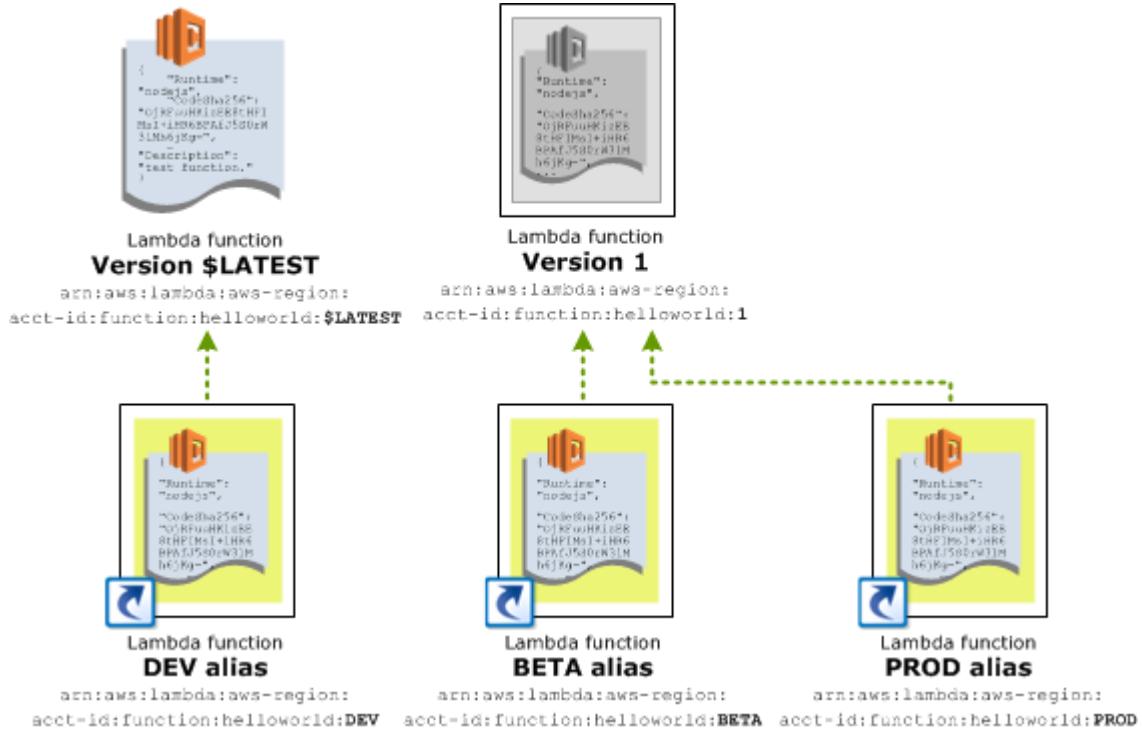
然后，您在测试环境中以可靠的方式使用事件源测试函数版本，同时继续开发较新的版本。

您可从 \$LATEST 中发布版本并使另一个别名 (BETA) 指向它。通过使用这种方法，您可以将测试事件源与该特定别名相关联。在事件源映射中，使用 BETA 别名将您的 Lambda 函数与事件源关联。



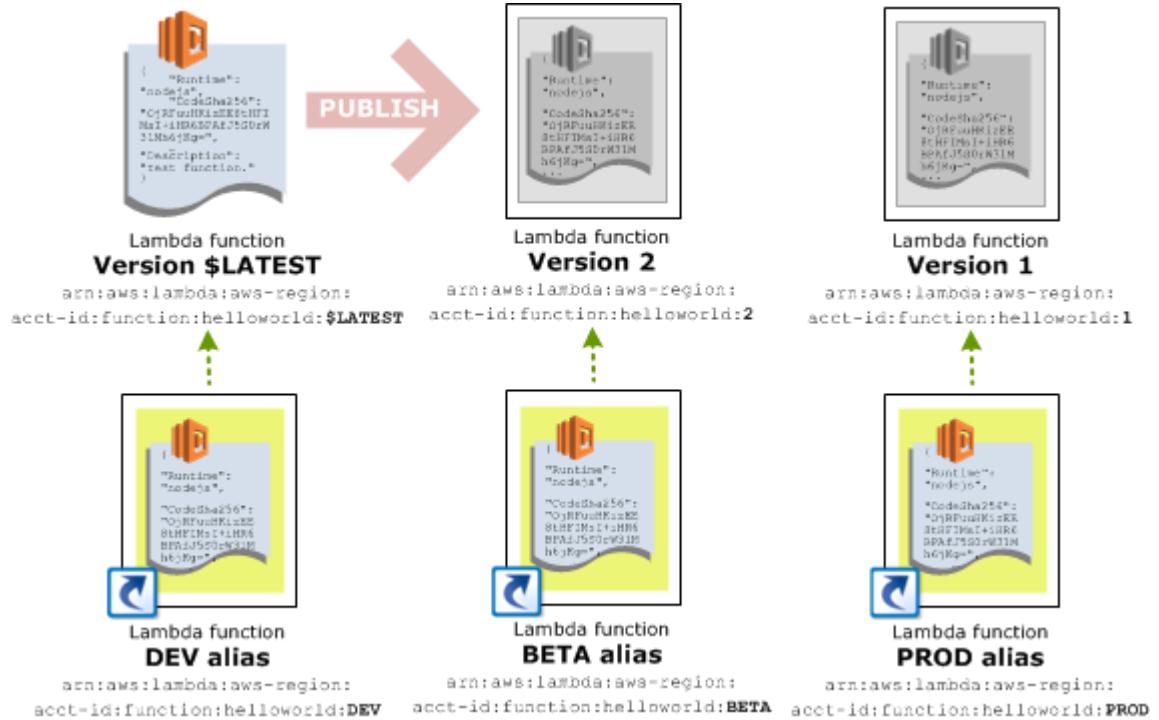
接下来，您将 Lambda 函数版本提升到生产环境以处理生产环境中的事件源。

在测试函数的 BETA 版本后，您可以创建映射到版本 1 的别名以定义生产版本。在这种方法中，您将生产事件源指向该特定版本。可通过在所有生产事件源映射中创建 PROD 别名并使用 PROD 别名 ARN 来执行此操作。



您继续开发，发布更多版本以及进行测试。

在开发代码时，您可以上传更新的代码以更新 \$LATEST 版本，然后将 BETA 别名指向该版本以发布到 Beta 测试。通过这种简单的 Beta 别名重新映射，您可以将 Lambda 函数的版本 2 发布到 Beta 测试，而无需更改任何事件源。这种方法说明了如何通过别名控制在开发环境中与特定事件源一起使用的函数版本。



如果要尝试使用 AWS Command Line Interface 创建该设置，请参阅[教程：使用 AWS Lambda 别名 \(p. 273\)](#)。

相关主题

[AWS Lambda 版本控制简介 \(p. 265\)](#)

[使用别名的流量转移 \(p. 280\)](#)

[教程：使用 AWS Lambda 别名 \(p. 273\)](#)

[使用 AWS 管理控制台、AWS CLI 或 Lambda API 操作管理版本控制 \(p. 278\)](#)

教程：使用 AWS Lambda 别名

这个基于 AWS CLI 的教程将创建 Lambda 函数版本以及指向该版本的别名（如[示例：使用别名管理 Lambda 函数版本 \(p. 270\)](#)中所述）。

该示例使用 us-west-2 (美国西部俄勒冈) 区域创建 Lambda 函数和别名。

创建 Lambda 函数 (helloworld)。

```
aws lambda create-function \
--region region \
--function-name helloworld \
--zip-file fileb://file-path/helloworld.zip \
--role arn:aws:iam::account-id:role/lambda_basic_execution \
--handler helloworld.handler \
--runtime nodejs6.10 \
--profile adminuser
```

响应返回将 \$LATEST 显示为函数版本的配置信息，如以下示例响应所示。

```
{  
    "CodeSha256": "OjRFuuHKizEE8tHFIMsI+iHR6BPAfJ5S0rW31Mh6jKg=",  
    "FunctionName": "helloworld",  
    "CodeSize": 287,  
    "MemorySize": 128,  
    "FunctionArn": "arn:aws:lambda:us-west-2:account-id:function:helloworld",  
    "Version": "$LATEST",  
    "Role": "arn:aws:iam::account-id:role/lambda_basic_execution",  
    "Timeout": 3,  
    "LastModified": "2015-09-30T18:39:53.873+0000",  
    "Handler": "helloworld.handler",  
    "Runtime": "nodejs6.10",  
    "Description": ""  
}
```

1. 创建一个部署程序包，您可以上传该程序包以创建 Lambda 函数：

- a. 打开文本编辑器，然后复制以下代码。

```
console.log('Loading function');

exports.handler = function(event, context, callback) {
    console.log('value1 =', event.key1);
    console.log('value2 =', event.key2);
    console.log('value3 =', event.key3);
    callback(null, "message");
};
```

- b. 将该文件保存为 helloworld.js。

- c. 将 helloworld.js 文件压缩为 helloworld.zip。

2. 创建一个 AWS Identity and Access Management (IAM) 角色 (执行角色)，您可以在创建 Lambda 函数时指定该角色：

- a. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
- b. 按照 IAM 用户指南 中 [IAM 角色](#) 的步骤操作，创建 IAM 角色 (执行角色)。遵循步骤创建角色时，请注意以下事项：

- 对于选择角色类型，请选择 AWS 服务角色，然后选择 AWS Lambda。
- 对于附加策略，请选择名为 AWSLambdaBasicExecutionRole 的策略。

- c. 写下 IAM 角色的 Amazon 资源名称 (ARN)。在下一步中创建 Lambda 函数时，您将需要此值。

3. 创建一个指向 helloworld Lambda 函数的 \$LATEST 版本的别名 (DEV)。

```
aws lambda create-alias \  
--region region \  
--function-name helloworld \  
--description "sample alias" \  
--function-version "\$LATEST" \  
--name DEV \  
--profile adminuser
```

响应返回别名信息，包括它指向的函数版本以及别名 ARN。ARN 与具有别名后缀的函数 ARN 相同。以下是一个示例响应。

```
{  
    "AliasArn": "arn:aws:lambda:us-west-2:account-id:function:helloworld:DEV",
```

```
"FunctionVersion": "$LATEST",
"Name": "DEV",
"Description": "sample alias"
}
```

4. 发布 helloworld Lambda 函数的版本。

```
aws lambda publish-version \
--region region \
--function-name helloworld \
--profile adminuser
```

响应返回函数版本的配置信息，包括版本号和具有版本后缀的函数 ARN。以下是一个示例响应。

```
{
  "CodeSha256": "OjRFuuHKizEE8tHFIMsI+iHR6BPAfJ5S0rW31Mh6jKg=",
  "FunctionName": "helloworld",
  "CodeSize": 287,
  "MemorySize": 128,
  "FunctionArn": "arn:aws:lambda:us-west-2:account-id:function:helloworld:1",
  "Version": "1",
  "Role": "arn:aws:iam::account-id:role/lambda_basic_execution",
  "Timeout": 3,
  "LastModified": "2015-10-03T00:48:00.435+0000",
  "Handler": "helloworld.handler",
  "Runtime": "nodejs6.10
  ",
  "Description": ""
}
```

5. 为 helloworld Lambda 函数版本 1 创建别名 (BETA)。

```
aws lambda create-alias \
--region region \
--function-name helloworld \
--description "sample alias" \
--function-version 1 \
--name BETA \
--profile adminuser
```

现在您为 helloworld 函数创建了两个别名。DEV 别名指向 \$LATEST 函数版本，BETA 别名指向 Lambda 函数的版本 1。

6. 假设您要将 helloworld 函数的版本 1 放在生产环境中。创建另一个指向版本 1 的别名 (PROD)。

```
aws lambda create-alias \
--region region \
--function-name helloworld \
--description "sample alias" \
--function-version 1 \
--name PROD \
--profile adminuser
```

此时，BETA 和 PROD 别名均指向 Lambda 函数的版本 1。

7. 您现在可以发布更新的版本（例如，版本 2），但首先您需要更新代码并上传修改后的部署程序包。如果 \$LATEST 版本未更改，则无法发布其多个版本。假设您已更新部署程序包、将其上传并发布了版本 2，您现在可以更改 BETA 别名以指向 Lambda 函数的版本 2。

```
aws lambda update-alias \
--region region \
```

```
--function-name helloworld \
--function-version 2 \
--name BETA \
--profile adminuser
```

现在您具有指向 Lambda 函数的不同版本的三个别名（DEV 别名指向 \$LATEST 版本、BETA 别名指向版本 2、PROD 别名指向 Lambda 函数的版本 1）。

有关使用 AWS Lambda 控制台管理版本控制的信息，请参阅[使用 AWS 管理控制台、AWS CLI 或 Lambda API 操作管理版本控制 \(p. 278\)](#)。

在推模型中授予权限

在推模型中（请参阅[事件源映射 \(p. 142\)](#)），事件源（例如 Amazon S3）会调用您的 Lambda 函数。这些事件源保留一个映射，以指定在发生事件时它们调用的函数版本或别名。请注意以下几点：

- 建议您在映射配置中指定现有 Lambda 函数别名（请参阅[AWS Lambda 别名简介 \(p. 269\)](#)）。例如，如果事件源是 Amazon S3，则在存储桶通知配置中指定别名 ARN，以便 Amazon S3 在检测到特定事件时能够调用别名。
- 在推模型中，使用附加到 Lambda 函数的资源策略向事件源授予权限。在版本控制中，您添加的权限特定于您在 AddPermission 请求中指定的限定词（请参阅[版本控制、别名和资源策略 \(p. 276\)](#)）。

例如，以下 AWS CLI 命令为 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 \
--region region \
--function-name helloworld \
--qualifier PROD \
--statement-id 1 \
--principal s3.amazonaws.com \
--action lambda:InvokeFunction \
--source-arn arn:aws:s3:::examplebucket \
--source-account 111111111111 \
--profile adminuser
```

在此情况下，Amazon S3 现在可以调用 PROD 别名，然后 AWS Lambda 可执行 PROD 别名指向的 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。

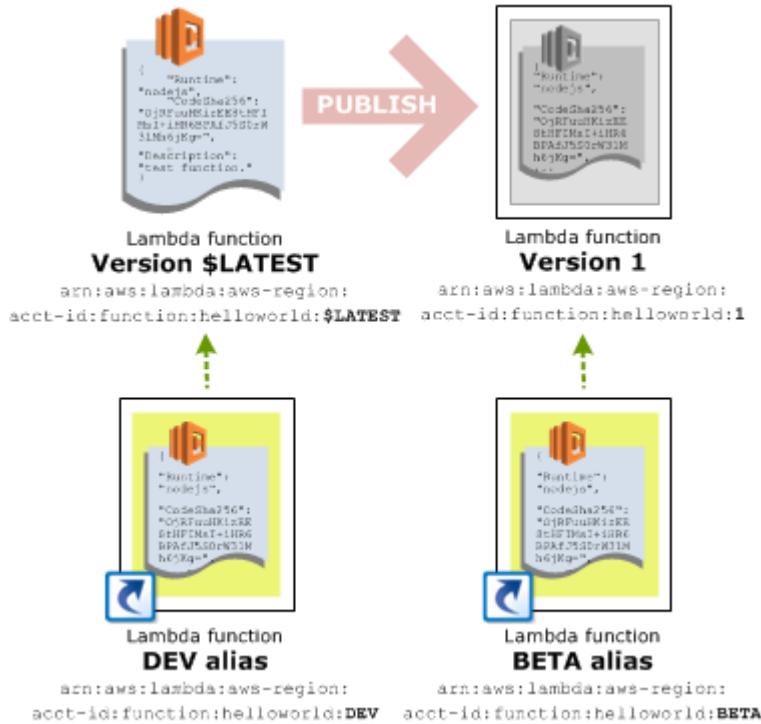
有关如何处理 Amazon S3 事件的信息，请参阅[教程：将 AWS Lambda 与 Amazon S3 结合使用 \(p. 166\)](#)。

Note

如果您使用 AWS Lambda 控制台为您的 Lambda 函数添加事件源，则控制台会为您添加必要权限。

版本控制、别名和资源策略

利用版本控制和别名，您可以使用各种 ARN 访问 Lambda 函数。例如，考虑以下情形。



例如，您可以使用以下两种 ARN 之一调用 helloworld 函数版本 1：

- 使用限定函数 ARN，如下所示。

```
arn:aws:lambda:aws-region:acct-id:function:helloworld:1
```

Note

非限定函数 ARN (没有版本或别名后缀的函数 ARN) 映射到 \$LATEST 版本。

- 使用 BETA 别名 ARN，如下所示。

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

在推送模型中，事件源（例如 Amazon S3 和自定义应用程序）可以调用任一 Lambda 函数版本，前提是您通过使用与 Lambda 函数关联的访问策略来向这些事件源授予必要的权限。有关推模型的更多信息，请参阅事件源映射 (p. 142)。

假定您授予权限，下一个问题是，“事件源是否能使用任一关联的 ARN 来调用函数版本？”答案是，这取决于您如何在添加权限请求中标识函数（请参阅 [AddPermission \(p. 373\)](#)）。了解这一点的关键是，您授予权限仅适用于添加权限请求中使用的 ARN：

- 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 将导致权限错误）。
- 如果使用别名（如 helloworld:BETA），该权限仅在使用 BETA 别名 ARN 调用 helloworld 函数时有效（使用任何其他 ARN 将导致权限错误，包括别名指向的函数版本 ARN）。
- 如果您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 将导致权限错误）。

Note

请注意，即使访问策略仅作用于非限定的 ARN，调用的 Lambda 函数的代码和配置仍来自函数版本 \$LATEST。非限定的函数 ARN 映射到 \$LATEST 版本，但您添加的权限是 ARN 特定的。

- 如果通过 \$LATEST 版本 (helloworld:\$LATEST) 使用限定函数名称，该权限仅在使用限定 ARN 调用 helloworld 函数版本 \$LATEST 时有效 (使用非限定 ARN 将导致权限错误)。

使用 AWS 管理控制台、AWS CLI 或 Lambda API 操作管理版本控制

您可使用 AWS 软件开发工具包（如果需要，可直接调用 AWS Lambda API）、AWS Command Line Interface (AWS CLI) 或 AWS Lambda 控制台以编程方式来管理 Lambda 函数版本控制。

AWS Lambda 提供以下 API 来管理版本控制和别名：

[PublishVersion \(p. 447\)](#)

[ListVersionsByFunction \(p. 444\)](#)

[CreateAlias \(p. 378\)](#)

[UpdateAlias \(p. 462\)](#)

[DeleteAlias \(p. 395\)](#)

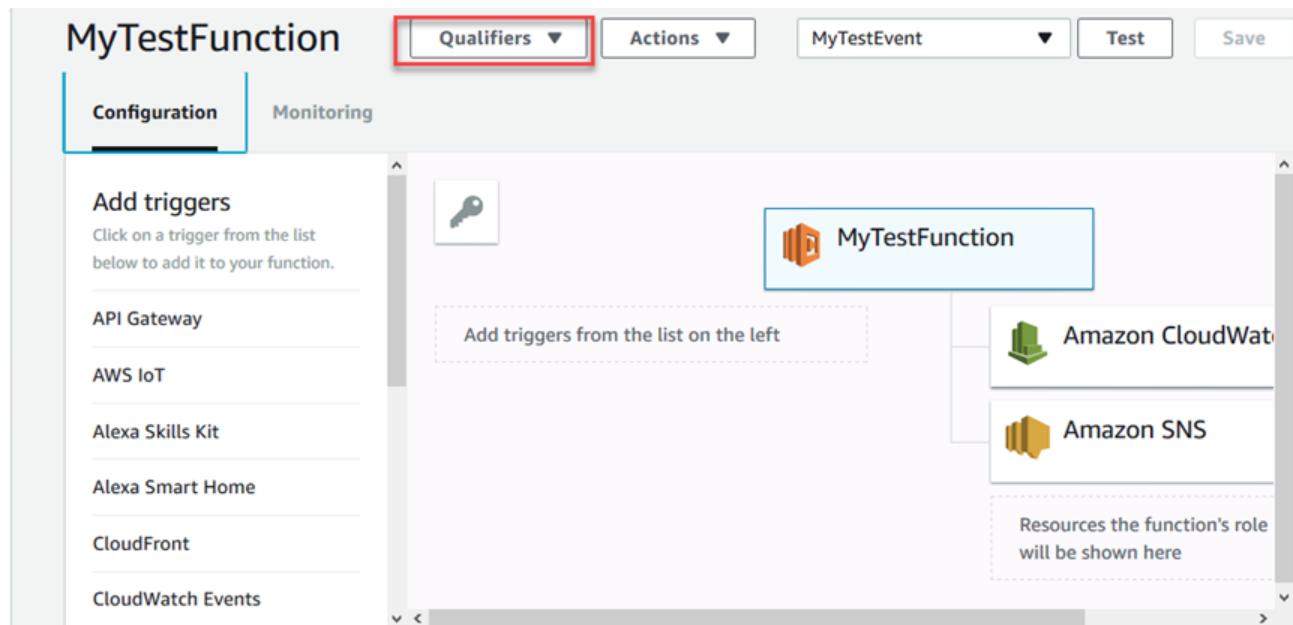
[GetAlias \(p. 407\)](#)

[ListAliases \(p. 433\)](#)

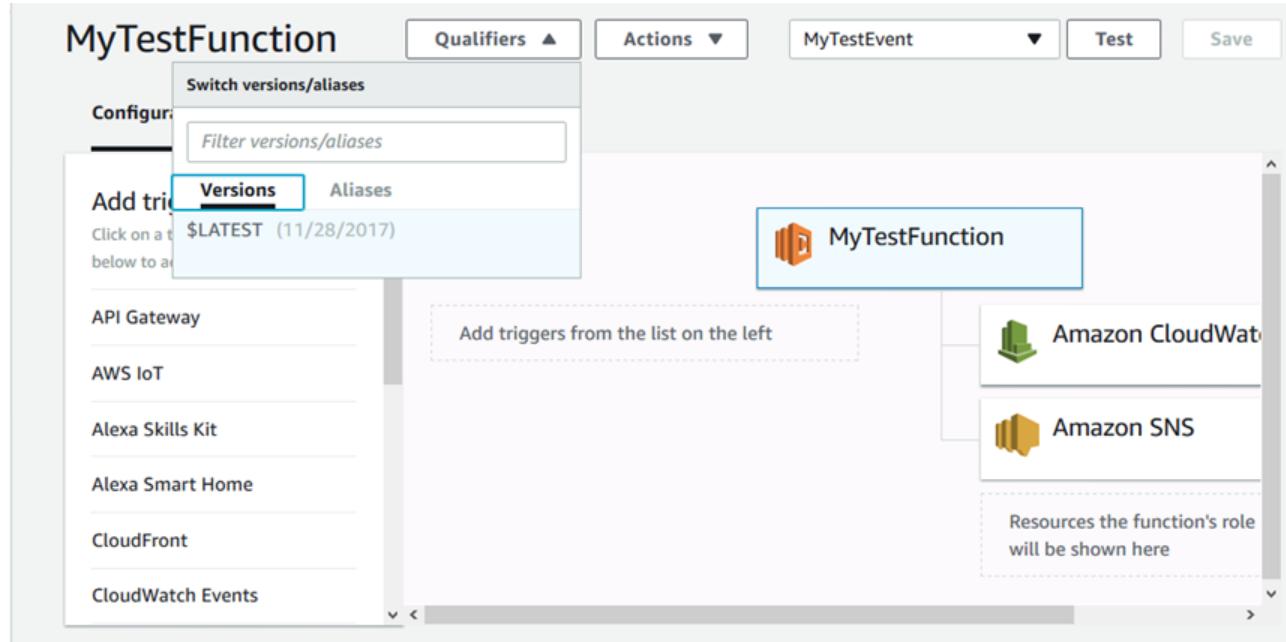
除了这些 API 之外，现有的相关 API 也支持与版本控制相关的操作。

有关如何使用 AWS CLI 的示例，请参阅[教程：使用 AWS Lambda 别名 \(p. 273\)](#)。

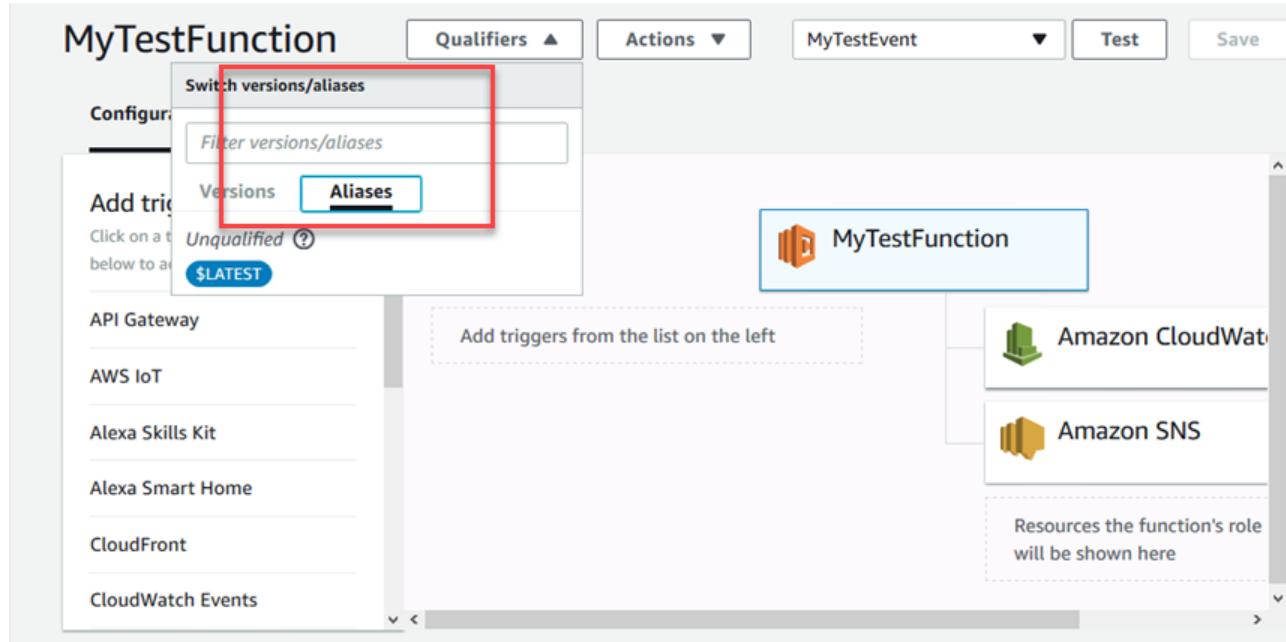
本节介绍如何使用 AWS Lambda 控制台管理版本控制。在 AWS Lambda 控制台中，选择函数，然后选择 Qualifiers。



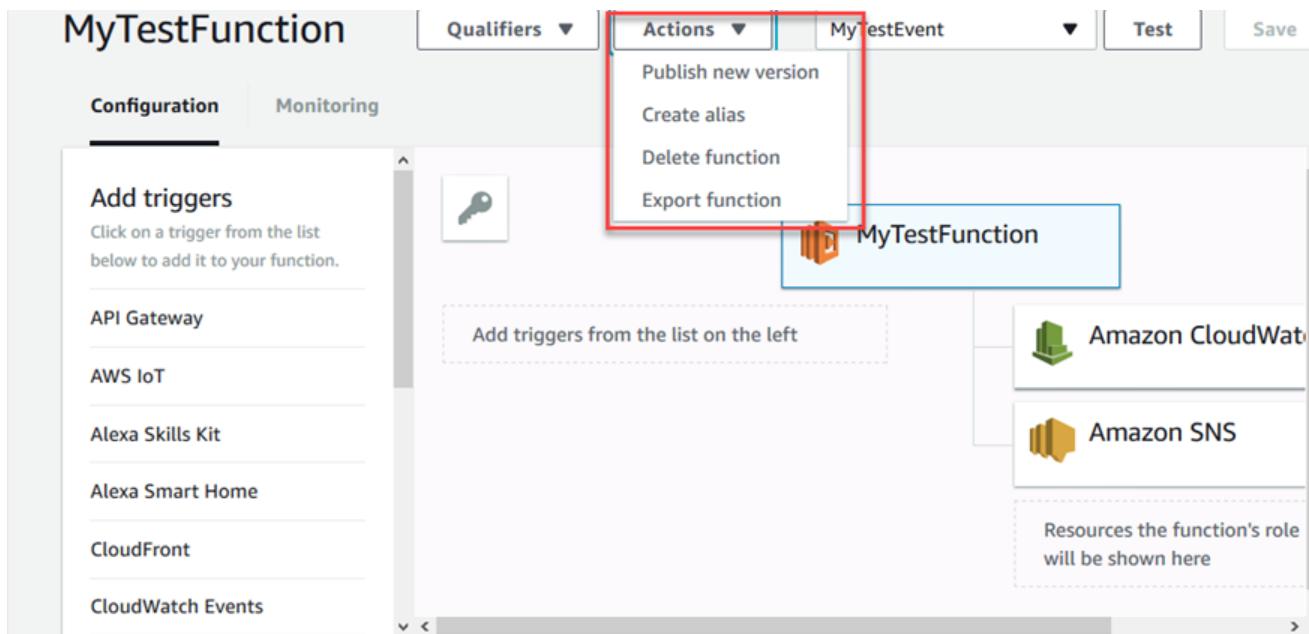
展开的 Qualifiers 菜单显示 Versions 和 Aliases 选项卡，如以下屏幕截图中所示。在 Versions 窗格中，可以查看所选函数的版本列表。如果以前没有为选定的函数发布版本，版本窗格仅列出 \$LATEST 版本，如下所示。



选择 Aliases 选项卡以查看该函数的别名列表。最初，没有任何别名，如下所示。



现在，您可以使用操作菜单为选定的 Lambda 函数发布版本或创建别名。



要了解有关版本控制和别名的信息，请参阅[AWS Lambda 函数版本控制和别名 \(p. 264\)](#)。

使用别名的流量转移

默认情况下，别名指向单个 Lambda 函数版本。在更新别名以指向不同的函数版本时，传入请求流量将立即指向更新的版本。这会导致该别名出现新版本引入的任何潜在的不稳定性问题。要最大限度降低这种影响，您可以实施 Lambda 别名的 `routing-config` 参数以允许指向 Lambda 函数的两个不同版本，并指示发送到每个版本的传入流量百分比。

例如，您可以指定在分析生产环境就绪性时仅将 2% 传入流量路由到新版本，而将其余 98% 路由到原始版本。随着新版本的不断成熟，您可以根据需要逐渐更新比率，直到确定新版本是稳定的。然后，您可以更新别名以将所有流量路由到新版本。

Note

您可以将别名最多指向两个 Lambda 函数版本。此外：

- 两个版本应具有相同的 [死信队列 \(p. 361\)](#) 配置（或没有 DLQ 配置）。
- 两个版本应具有相同的 IAM 执行角色。
- 在将别名指向多个版本时，别名不能指向 `$LATEST`。

使用别名的流量转移 (CLI)

要配置别名以使用 [CreateAlias \(p. 378\)](#) 操作根据权重在两个函数版本之间转移流量，您需要配置 `routing-config` 参数。以下示例将别名指向两个不同的 Lambda 函数版本，版本 2 接收 2% 调用流量，其余 98% 调用版本 1。

```
aws lambda create-alias --name alias_name --function-name function-name \
--function-version 1
--routing-config AdditionalVersionWeights={"2":0.02}
```

您可以使用 [UpdateAlias \(p. 462\)](#) 操作更新新版本（版本 2）的传入流量百分比。例如，您可以将新版本的调用流量提高到 5%，如下所示。

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

要将所有流量路由到版本 2，请再次使用 UpdateAlias 操作更改 function-version 属性以指向版本 2。然后，将 routing-config 参数设置为空字符串，如下所示。

```
aws lambda update-alias --name alias name --function-name function-name \  
--function-version 2 --routing-config ''
```

使用别名的流量转移 (控制台)

您可以使用 Lambda 控制台配置使用别名的流量转移，如下所述：

1. 打开您的 Lambda 函数，并确认具有至少两个以前发布的版本。否则，您可以转到 [AWS Lambda 版本控制简介 \(p. 265\)](#) 以了解有关版本控制的更多信息，然后发布第一个函数版本。
2. 对于操作，请选择创建别名。
3. 在创建新的别名窗口中，指定名称*、描述 (可选) 以及别名将指向的 Lambda 函数版本* 的值。此处的版本是 1。
4. 在其他版本中，指定以下内容：
 - a. 指定第二个 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%，则自动为第一个版本分配 90%。
5. 选择 Create。

确定已调用的版本

当别名在两个函数版本之间转移流量时，可以通过两种方法确定已调用的 Lambda 函数版本：

1. 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。下面是一个示例。

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

Lambda 使用 Executed Version 维度按执行的版本筛选指标数据。这仅适用于别名调用。有关更多信息，请参阅 [AWS Lambda CloudWatch 维度 \(p. 302\)](#)。

2. 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。

主题

- [使用 AWS 无服务器应用程序模型 \(AWS SAM\) \(p. 281\)](#)
- [基于 Lambda 应用程序的自动化部署 \(p. 288\)](#)

使用 AWS 无服务器应用程序模型 (AWS SAM)

AWS 无服务器应用程序模型 (AWS SAM) 是定义无服务器应用程序的模型。AWS SAM 本身受 AWS CloudFormation 支持，且为表达无服务器资源定义简化的语法。规范目前包含 API、Lambda 函数和 Amazon DynamoDB 表。Apache 2.0 下的规范适用于 AWS 合作伙伴和客户，以便他们在自己的工具集中采用和扩展。有关规范详情，请参阅 [AWS 无服务器应用程序模型](#)。

AWS SAM 中的无服务器资源

带有符合 AWS SAM 模型的无服务器资源的 AWS CloudFormation 模板称为 SAM 文件或模板。

以下示例介绍如何利用 AWS SAM 声明无服务器应用程序的常见组件。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。

Lambda 函数

以下内容介绍您用来描述 Lambda 函数的表示法：

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

  FunctionName:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: runtime
      CodeUri: s3://bucketName/codepackage.zip
```

Handler 属性的 handler 值指向包含代码的模块，您的 Lambda 函数将在调用时执行。Handler 属性的 index 值表示包含代码的文件名称。您可以尽可能多地声明您的无服务器应用程序所需的函数。

您还可以声明环境变量，这是您可以为应用程序设置的配置设置。下面介绍具有两个 Lambda 函数和一个指向 DynamoDB 表的环境变量的无服务器应用程序的示例。您可以更新环境变量，无需修改、重新打包或者重新部署您的 Lambda 函数代码。有关更多信息，请参阅 [环境变量 \(p. 354\)](#)。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  PutFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs6.10
      Policies: AWSLambdaDynamoDBExecutionRole
      CodeUri: s3://bucketName/codepackage.zip
      Environment:
        Variables:
          TABLE_NAME: !Ref Table
  DeleteFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs6.10
      Policies: AWSLambdaDynamoDBExecutionRole
      CodeUri: s3://bucketName/codepackage.zip
      Environment:
        Variables:
          TABLE_NAME: !Ref Table
  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: streamview type
```

请注意顶部的表示法：

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

为了包含 AWS CloudFormation 模板中 AWS 无服务器应用程序模型所定义的对象，需要这样表示。

SimpleTable

SimpleTable 是使用单属性主密钥创建 DynamoDB 表的资源。如果无服务器应用程序要交互的数据只需要通过单值密钥访问，则您可以使用此简化版。您可以更新之前的示例，以使用 SimpleTable，如下所示：

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Resources:  
  TableName:  
    Type: AWS::Serverless::SimpleTable  
    Properties:  
      PrimaryKey:  
        Name: id  
        Type: String  
      ProvisionedThroughput:  
        ReadCapacityUnits: 5  
        WriteCapacityUnits: 5
```

事件

事件是触发 Lambda 函数的 AWS 资源，例如 Amazon API Gateway 终端节点或 Amazon SNS 通知。Events 属性是您能够为每个函数设置多个事件的数组。以下内容介绍用来描述 Lambda 函数以及作为事件源的 DynamoDB 表的表示法。

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
Resources:  
  FunctionName:  
    Type: AWS::Serverless::Function  
    Properties:  
      Handler: index.handler  
      Runtime: nodejs6.10  
    Events:  
      Stream:  
        Type: DynamoDB  
        Properties:
```

```
Stream: !GetAtt DynamoDBTable.StreamArn
BatchSize: 100
StartingPosition: TRIM_HORIZON
TableName:
Type: AWS::DynamoDB::Table
Properties:
AttributeDefinitions:
- AttributeName: id
AttributeType: S
KeySchema:
- AttributeName: id
KeyType: HASH
ProvisionedThroughput:
ReadCapacityUnits: 5
WriteCapacityUnits: 5
```

如上所述，您可以设置多个将触发 Lambda 函数的事件源。以下示例显示了可由 HTTP PUT 或 POST 事件触发的 Lambda 函数。

API

使用 AWS SAM 定义 API 的方式有两种。以下示例使用 Swagger 配置基础的 Amazon API Gateway 资源：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
Api:
Type: AWS::Serverless::Api
Properties:
StageName: prod
DefinitionUri: swagger.yml
```

在下一个示例中，AWS::Serverless::Api 资源类型是从 AWS::Serverless::Function 资源定义的 API 事件联合添加的隐式资源类型。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
GetFunction:
Type: AWS::Serverless::Function
Properties:
Handler: index.get
Runtime: nodejs6.10
CodeUri: s3://bucket/api_backend.zip
Policies: AmazonDynamoDBReadOnlyAccess
Environment:
Variables:
TABLE_NAME: !Ref Table
Events:
GetResource:
Type: Api
Properties:
Path: /resource/{resourceId}
Method: get

PutFunction:
Type: AWS::Serverless::Function
Properties:
Handler: index.put
Runtime: nodejs6.10
```

```
CodeUri: s3://bucket/api_backend.zip
Policies: AmazonDynamoDBFullAccess
Environment:
  Variables:
    TABLE_NAME: !Ref Table
Events:
  PutResource:
    Type: Api
    Properties:
      Path: /resource/{resourceId}
      Method: put

DeleteFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.delete
    Runtime: nodejs6.10
    CodeUri: s3://bucket/api_backend.zip
    Policies: AmazonDynamoDBFullAccess
    Environment:
      Variables:
        TABLE_NAME: !Ref Table
    Events:
      DeleteResource:
        Type: Api
        Properties:
          Path: /resource/{resourceId}
          Method: delete

Table:
  Type: AWS::Serverless::SimpleTable
```

在上述示例中，AWS CloudFormation 将自动生成带有路径 "/resource/{resourceId}" 和方法 GET、PUT 和 DELETE 的 Amazon API Gateway API。

权限

您可以提供用作该函数执行角色的 AWS Identity and Access Management (IAM) 角色的 Amazon 资源名称 (ARN)，如下所示：

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  FunctionName:
    Type: AWS::Serverless::Function
    Properties:
      Role:role arn
```

或者，您可以向 Lambda 函数资源提供一个或多个托管策略。然后，AWS CloudFormation 将使用托管策略和默认 Lambda 基本执行策略创建新的角色。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  FunctionName:
    Type: AWS::Serverless::Function
    Properties:
      Policies: AmazonDynamoDBFullAccess
```

如果以上角色和策略均未提供，则使用 Lambda 基本执行权限创建默认执行角色。

Note

除了使用无服务器资源之外，您还可以使用常规 AWS CloudFormation 语法，以表达同一模板中的资源。尽管 SAM 模型目前不包含任何资源，但仍可使用 AWS CloudFormation 语法在 AWS CloudFormation 模板中创建资源。此外，作为使用 SAM 模型的替代方案，您可以使用 AWS CloudFormation 语法表达无服务器资源。有关使用常规 CloudFormation 语法将 Lambda 函数指定为您 SAM 模板一部分的信息，请参阅 AWS CloudFormation 用户指南中的 [AWS::Lambda::Function](#)。

有关无服务器应用程序示例的完整列表的信息，请参阅[如何使用 AWS Lambda 的示例 \(p. 165\)](#)。

下一步

[创建您自己的无服务器应用程序 \(p. 286\)](#)

创建您自己的无服务器应用程序

在以下教程中，您将创建包含单个 Node.js 函数的简单无服务器应用程序，该函数返回您指定为环境变量的 Amazon S3 存储桶的名称。请遵循以下步骤：

1. 将以下 Node.js 代码复制并粘贴到文本文件中，并将其保存为 `index.js`

```
var AWS = require('aws-sdk');

exports.handler = function(event, context, callback) {
    var bucketName = process.env.S3_BUCKET;
    callback(null, bucketName);
}
```

2. 将以下内容粘贴为文本文件并将其保存为 `example.yaml`。请注意，`Runtime` 参数使用 `nodejs6.10`，但您可以指定 `nodejs4.3`。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  TestFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: index.handler
      Runtime: nodejs6.10
      Environment:
        Variables:
          S3_BUCKET: bucket-name
```

3. 创建名为 `examplefolder` 的文件夹，并将 `example.yaml` 文件和 `index.js` 文件放置在该文件夹内。

您的实例文件夹现在包含下列两份文件，您可以在之后使用它们打包无服务器应用程序。

- `example.yaml`
- `index.js`

打包和部署

您在创建 Lambda 函数处理程序和 `example.yaml` 文件后，可以使用 AWS CLI 打包和部署您的无服务器应用程序。

包装

要打包您的应用程序，请创建 package 命令将用来上传您的 ZIP 部署程序包的 Amazon S3 存储桶（如果您还未在 example.yaml 文件中指定）。您可以使用以下命令创建 Amazon S3 存储桶：

```
aws s3 mb s3://bucket-name --region region
```

下一步，打开命令提示符并键入以下命令：

```
aws cloudformation package \  
--template-file file path/example.yaml \  
--output-template-file serverless-output.yaml \  
--s3-bucket s3-bucket-name
```

程序包命令返回 AWS SAM 模板，在这种情况下，指的是包含指向您指定的 Amazon S3 存储桶中的部署 zip 的 CodeUri 的 serverless-output.yaml。此模板表示您的无服务器应用程序。现在您已准备好部署它。

部署

要部署应用程序，请运行以下命令：

```
aws cloudformation deploy \  
--template-file serverless-output.yaml \  
--stack-name new-stack-name \  
--capabilities CAPABILITY_IAM
```

请注意，您为 --template-file 参数指定的值是程序包命令返回的 SAM 模板的名称。此外，--capabilities 参数是可选的。AWS::Serverless::Function 资源将隐式创建角色（如果模板中未指定），以执行 Lambda 函数。您可以使用 --capabilities 参数显示确认允许 AWS CloudFormation 代表您创建角色。

当您运行 aws cloudformation deploy 命令时，该命令可创建 AWS CloudFormation ChangeSet，这是 AWS CloudFormation 堆栈更改的列表，然后对其进行部署。一些堆栈模板可能包含可能影响您的 AWS 账户的资源，例如，通过创建新的 AWS Identity and Access Management (IAM) 用户。对于这些堆栈，您必须指定 --capabilities 参数以显示确认它们的功能。有关更多信息，请参阅 AWS CloudFormation API Reference 中的 [CreateChangeSet](#)。

要验证您的结果，请打开 AWS CloudFormation 控制台，以查看新创建的 AWS CloudFormation 堆栈；以及 Lambda 控制台，以查看您的函数。

有关无服务器应用程序示例的完整列表的信息，请参阅 [如何使用 AWS Lambda 的示例 \(p. 165\)](#)。

导出无服务器应用程序

您可以导出无服务器应用程序，并使用 Lambda 控制台将其部署到不同的 AWS 区域或开发阶段等。当您导出 Lambda 函数时，则为您提供表示无服务器应用程序的 ZIP 部署程序包和 SAM 模板。然后您可以使用之前重新部署部分描述的 package 和 deploy 命令。

您还可以选择其中一个 Lambda 蓝图创建 ZIP 程序包，以便您进行打包和部署。请按照以下步骤执行此操作：

使用 Lambda 控制台导出无服务器应用程序

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. 执行以下任一操作：

- 使用 Lambda 蓝图创建函数 - 选择蓝图并按照以下步骤创建 Lambda 函数。有关示例，请参阅[创建简单的 Lambda 函数 \(p. 8\)](#)。当您到达审核页面时，选择导出函数。
 - 创建函数 - 选择创建函数，然后创建您的函数。您在创建 Lambda 函数后，可选择函数以将其导出。选择操作，然后选择导出函数。
 - 打开现有的 Lambda 函数 - 通过依次选择 Function name、Actions、Export function 来打开此函数。
3. 在导出您的函数窗口，您有以下选择：
- 选择下载 AWS SAM 文件，这定义 Lambda 函数和组成无服务器应用程序的其他资源。
 - 选择下载部署程序包，这包含您的 Lambda 函数代码和任何相关库。

使用 AWS SAM 文件和 ZIP 部署程序包，并按照[打包和部署 \(p. 286\)](#)中的步骤重新部署无服务器应用程序。

基于 Lambda 应用程序的自动化部署

在上一部分中，您了解到如何创建 SAM 模板、生成部署程序包并使用 AWS CLI 手动部署无服务器应用程序。在本部分，您将利用以下 AWS 服务完全自动化部署过程。

- CodePipeline：您使用 CodePipeline 建模、可视化和自动化释放无服务器应用程序必需的步骤。有关更多信息，请参阅[什么是 AWS CodePipeline](#)？
- CodeBuild：您使用 CodeBuild 构建、本地测试和打包无服务器应用程序。有关更多信息，请参阅[什么是 AWS CodeBuild](#)？
- AWS CloudFormation：您使用 AWS CloudFormation 部署应用程序。有关更多信息，请参阅[什么是 AWS CloudFormation](#)？
- CodeDeploy：您可以使用 [AWS CodeDeploy](#) 为您的无服务器应用程序实现逐步部署更新。有关此操作的更多信息，请参阅[逐步代码部署 \(p. 293\)](#)。

以下各部分演示如何结合使用所有这些工具来整合您的无服务器应用程序。

下一步

[为您的无服务器应用程序构建管道 \(p. 288\)](#)

为您的无服务器应用程序构建管道

在下面的教程中，您将创建自动化部署无服务器应用程序的 AWS CodePipeline。首先，您将要设置源阶段以触发您的管道。在本教程中：

- 我们将使用 GitHub。有关如何创建 GitHub 存储库的说明，请参阅[在 GitHub 中创建存储库](#)。
- 您需要创建一个 AWS CloudFormation 角色并向该角色添加 AWSLambdaExecute 策略，如下所述：
 1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
 2. 按照 IAM 用户指南 中的[创建向 AWS 服务委托权限的角色](#)创建一个 IAM 角色 (执行角色)，然后转到为 AWS 服务创建角色部分。遵循步骤创建角色时，请注意以下事项：
 - 在 Select Role Type 中，选择 AWS Service Roles，然后选择 CloudFormation。选择 Next: Permissions。
 - 在 Attach permissions policies 中，使用搜索栏查找然后选择 AWSLambdaExecute。选择 Next: Review。
 - 在 Role Name 中，使用在 AWS 账户内唯一的名称 (例如 cloudformation-lambda-execution-role)，然后选择 Create role。

- 打开您刚刚创建的角色，在 Permissions 选项卡下，选择 Add inline policy。
- 在 Create Policy 中选择 JSON 选项卡，然后输入以下内容：

Note

确保将 *region* 和 *id* 占位符替换为您的区域和账户 ID。

```
{  
  "Statement": [  
    {  
      "Action": [  
        "s3:GetObject",  
        "s3:GetObjectVersion",  
        "s3:GetBucketVersioning"  
      ],  
      "Resource": "*",  
      "Effect": "Allow"  
    },  
    {  
      "Action": [  
        "s3:PutObject"  
      ],  
      "Resource": [  
        "arn:aws:s3:::codepipeline*"  
      ],  
      "Effect": "Allow"  
    },  
    {  
      "Action": [  
        "lambda:*"  
      ],  
      "Resource": [  
        "arn:aws:lambda:region:id:function:/*"  
      ],  
      "Effect": "Allow"  
    },  
    {  
      "Action": [  
        "apigateway:/*"  
      ],  
      "Resource": [  
        "arn:aws:apigateway:region::/*"  
      ],  
      "Effect": "Allow"  
    },  
    {  
      "Action": [  
        "iam:GetRole",  
        "iam>CreateRole",  
        "iam>DeleteRole",  
        "iam:PutRolePolicy"  
      ],  
      "Resource": [  
        "arn:aws:iam::id:role/*"  
      ],  
      "Effect": "Allow"  
    },  
    {  
      "Action": [  
        "iam:AttachRolePolicy",  
        "iam>DeleteRolePolicy",  
        "iam:DetachRolePolicy"  
      ],  
      "Resource": [  
        "arn:aws:iam::id:role/*"  
      ]  
    }  
  ]  
}
```

```
    "arn:aws:iam::id:role/*"
],
"Effect": "Allow"
},
{
"Action": [
"iam:PassRole"
],
"Resource": [
"*"
],
"Effect": "Allow"
},
{
"Action": [
"cloudformation>CreateChangeSet"
],
"Resource": [
"arn:aws:cloudformation:region:aws:transform/Serverless-2016-10-31"
],
"Effect": "Allow"
},
{
"Action": [
"codedeploy>CreateApplication",
"codedeploy>DeleteApplication",
"codedeploy:RegisterApplicationRevision"
],
"Resource": [
"arn:aws:codedeploy:region:id:application:)"
],
"Effect": "Allow"
},
{
"Action": [
"codedeploy>CreateDeploymentGroup",
"codedeploy>CreateDeployment",
"codedeploy:GetDeployment"
],
"Resource": [
"arn:aws:codedeploy:region:id:deploymentgroup:)"
],
"Effect": "Allow"
},
{
"Action": [
"codedeploy:GetDeploymentConfig"
],
"Resource": [
"arn:aws:codedeploy:region:id:deploymentconfig:)"
],
"Effect": "Allow"
}
],
"Version": "2012-10-17"
}
```

- 选择 Validate Policy，然后选择 Apply Policy。

步骤 1：设置存储库

在设置存储库时，您可以使用任意 Lambda 支持的运行时。以下示例将使用 Node.js。

要设置您的存储库，请执行以下操作：

- 添加包含以下代码的 index.js 文件：

```
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(),
    });
};
```

- 添加包含以下内容的 samTemplate.yaml 文件。这是在您的应用程序中定义资源的 SAM 模板。此 SAM 模板定义 API 网关 触发的 Lambda 函数。请注意，runtime 参数使用 nodejs6.10，但您还可以指定 nodejs4.3。有关 AWS SAM 的更多信息，请参阅 [AWS 无服务器应用程序模型](#)。

```
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: nodejs6.10
      CodeUri: ./
    Events:
      MyTimeApi:
        Type: Api
        Properties:
          Path: /TimeResource
          Method: GET
```

- 添加 buildspec.yml 文件。构建规范是构建命令和相关设置的集合，以 YAML 格式并且 AWS CodeBuild 可用其运行某一版本。有关更多信息，请参阅 [为 AWS CodeBuild 构建规范参考](#)。在本例中，以下是构建操作过程：
 - 使用 `npm` 来安装时间程序包。
 - 运行 `Package` 命令，以在管道中为后续部署步骤准备部署程序包。有关程序包命令的更多信息，请参阅 [将本地项目上传至 S3 存储桶](#)。

```
version: 0.1
phases:
  install:
    commands:
      - npm install time
      - aws cloudformation package --template-file samTemplate.yaml --s3-bucket bucket-name
                                                --output-template-file NewSamTemplate.yaml
  artifacts:
    type: zip
    files:
      - samTemplate.yaml
```

请注意，您需要为您的 Amazon S3 存储桶提供 `--s3-bucket` 参数值，这类似于您将要采取的步骤（如果您打算使用 SAM 手动打包部署程序包），正如之前教程中 [包装 \(p. 287\)](#) 步骤所述。

步骤 2：创建您的管道

按照以下步骤创建您的 AWS CodePipeline。

1. 登录 AWS 管理控制台并打开 AWS CodePipeline 控制台。
2. 选择 Get Started Now。
3. 在管道名称：中，输入您的管道名称，然后选择下一步。
4. 在源提供商：中，选择 GitHub。
5. 选择连接到 GitHub：，然后选择您想要连接的存储库和分支。每个推送到您选择的分支的 Git 都将触发管道。选择下一步。
6. 选择 AWS CodeBuild 作为您的构建提供商。
7. 选择 Create a new build project，然后输入项目名称。
8. 选择 Ubuntu 作为操作系统。
9. 选择 Node.js 作为运行时。
10. 在版本中，选择 aws/codebuild/nodejs:*version*
11. 在 Build specification 中选择 Use the buildspec.yml in the source code root directory
12. 选择 Save build project。

Note

系统将代表您自动创建 AWS CodeBuild 服务角色。

选择下一步。

13. 在 Deployment provider* 中选择 AWS CloudFormation。

通过选择此选项，AWS CloudFormation 命令将用于部署 SAM 模板。有关更多信息，请参阅 [AWS SAM 中的无服务器资源 \(p. 282\)](#)。

14. 在 Action mode: 中，选择 Create or replace a change set。
15. 在堆栈名称：中，输入 MyBetaStack。
16. 在 Change set name: 中，输入 MyChangeSet。
17. 在 Template file: 中，输入 samTemplate.yaml。
18. 在 功能：中，选择 CAPABILITY_IAM。
19. 在 Role 中，选择您在本教程开始时创建的 AWS CloudFormation 角色，然后选择 Next step。
20. 选择创建角色。选择下一步，然后选择允许。选择下一步。
21. 检查您的管道，然后选择创建管道。

步骤 3：更新生成的服务策略

完成以下步骤，以允许 CodeBuild 将构建项目上传到 Amazon S3 存储桶。

1. 转至 IAM 管理控制台。
2. 选择 Roles。
3. 打开为您的项目生成的服务角色，通常为 code-build-*project-name*-service-role。
4. 在 Permissions 选项卡下，选择 Add inline policy。
5. 在 service 中，选择 Choose a service。

6. 在 Select a service below 中，选择 S3。
7. 在 Actions 中，选择 Select actions。
8. 在 Access level groups 下展开 Write，然后选择 PutObject。
9. 选择 Resources，然后选中 Any 复选框。
10. 选择查看策略。
11. 输入 Name*，然后选择 Create policy。然后返回到您在上一部分中创建的管道。

步骤 4：完成您的测试部署阶段

使用以下步骤完成您的测试阶段。

1. 选择 Edit。
2. 选择



图标 (位于 MyBetaStack 旁边)。

3. 在 Action category: 中，如果尚未选择，请选择 Deploy。
4. 在 Deployment provider* 中，如果尚未选择，请选择 AWS CloudFormation。
5. 在 Action mode* 中，选择 Execute a change set。这类似于您将要手动部署程序包时要采取的步骤，如之前教程的部署 (p. 287) 步骤所述。CreateChangeSet 将 SAM 模板转换为完整的 AWS CloudFormation 格式，而 deployChangeSet 部署 AWS CloudFormation 模板。
6. 在 Stack name* 中，输入或选择 MyBetaStack。
7. 在 Change set name* 中，输入 MyChangeSet。
8. 选择 Add Action。
9. 选择保存管道更改。
10. 选择 Save and continue。

您的管道已就绪。任何推送到您连接到此管道的分支的 Git 都将触发部署。要测试您的管道并首次部署应用程序，请执行以下操作之一：

- 将 Git 推送到连接到您的管道的分支。
- 转到 AWS CodePipeline 控制台，选择您创建的管道名称，然后选择 Release change。

下一步

[逐步代码部署 \(p. 293\)](#)

逐步代码部署

如果您使用 AWS SAM 创建无服务器应用程序，则会内置 [AWS CodeDeploy](#) 来实现安全的 Lambda 部署。只需几行配置，SAM 即可为您完成以下操作：

- 部署您的 Lambda 函数的新版本，并自动创建指向新版本的别名。
- 逐步将客户流量转移到新版本，直到您确认它按预期方式运行或者回滚更新。
- 定义转移流量前和转移流量后的测试函数，来验证新部署的代码是否已正确配置并且您的应用程序是否按预期方式运行。
- 如果触发 CloudWatch 警报，则回滚部署。

这些都可以通过更新 SAM 模板来完成。下面的示例演示了一个使用代码部署的简单版本，它将客户流量逐步转移到您新部署的版本：

```
Resources:
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs4.3
    CodeUri: s3://bucket/code.zip

  AutoPublishAlias: live

  DeploymentPreference:
    Type: Canary10Percent10Minutes
    Alarms:
      # A list of alarms that you want to monitor
      - !Ref AliasErrorMetricGreaterThanZeroAlarm
      - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
    Hooks:
      # Validation Lambda functions that are run before & after traffic shifting
      PreTraffic: !Ref PreTrafficLambdaFunction
      PostTraffic: !Ref PostTrafficLambdaFunction
```

在上面对 SAM 模板的修订中，提供以下功能：

- AutoPublishAlias - 通过添加此属性并指定别名，AWS SAM 可执行以下操作：
 - 根据对 Lambda 函数的 Amazon S3 URI 的更改，检测何时部署新代码。
 - 使用最新代码创建和发布该函数的更新版本。
 - 使用您提供的名称创建别名（除非已存在别名）并指向 Lambda 函数的更新版本。函数调用应该使用别名限定词来利用这一功能。如果您不熟悉 Lambda 函数版本控制和别名，请参阅[AWS Lambda 函数版本控制和别名 \(p. 264\)](#)。
- 部署首选项类型 - 在上面的示例中，10% 的客户流量将立即转移到您的新版本，并且在 10 分钟后，所有流量都将转移到新版本。但是，如果您的前期/后期挂钩测试失败或者触发了 CloudWatch 警报，CodeDeploy 将回滚您的部署。下表概述了除上面使用的选项之外其他可用的流量转移选项。请注意以下几点：
 - 金丝雀部署：流量将通过两次递增进行转移。您可以从预定义的金丝雀部署选项中选择，这些选项指定在第一次增量中转移到更新后的 Lambda 函数版本的流量百分比以及以分钟为单位的间隔；然后指定在第二次增量中转移剩余的流量。
 - 线性部署：流量使用相等的增量转移，在每次递增之间间隔的分钟数相同。您可以从预定义的线性选项中进行选择，这些选项指定在每次增量中转移的流量百分比以及每次增量之间的分钟数。
 - 一次性部署：所有流量均从原始 Lambda 函数一次性地转移到更新后的 Lambda 函数版本。

部署首选项类型

Canary10Percent30Minutes

Canary10Percent5Minutes

Canary10Percent10Minutes

Canary10Percent15Minutes

Linear10PercentEvery10Minutes

Linear10PercentEvery1Minute

部署首选项类型
Linear10PercentEvery2Minutes
Linear10PercentEvery3Minutes
AllAtOnce

- 警报 - 将由部署中出现的任何错误触发并自动回滚部署的 CloudWatch 警报。例如，如果您正在部署的更新代码在应用程序中生成错误，或者您指定的任何 AWS Lambda 或自定义 CloudWatch 指标违反了警报阈值。
- 挂钩 - 转移流量前和转移流量后的测试函数，这些函数在开始将流量转移到新版本之前以及完成将流量转移到新版本之后运行健全性检查。
 - 转移流量前：在开始转移流量之前，CodeDeploy 将调用转移流量前挂钩 Lambda 函数。此 Lambda 函数必须回调 CodeDeploy，以指示成功或失败。在失败时，该函数将中止并向 AWS CloudFormation 报告失败情况。在成功时，CodeDeploy 将继续转移流量。
 - 转移流量后：在转移完流量之后，CodeDeploy 将调用转移流量后挂钩 Lambda 函数。与转移流量前挂钩类似，该函数必须回调 CodeDeploy 来报告成功或失败。使用转移流量后挂钩可以运行集成测试或其他验证操作。

有关更多信息，请参阅 [SAM 安全部署参考](#)。

对基于 Lambda 的应用程序进行监控和问题排查

AWS Lambda 将自动跟踪 Lambda 函数调用的行为，并提供您可监控的反馈。此外，它提供的指标可让您分析完整的函数调用频谱，包括事件源集成以及下游资源是否按预期方式执行。以下部分提供了有关可用来分析 Lambda 函数调用行为的工具的指导信息：

主题

- [使用 Amazon CloudWatch \(p. 296\)](#)
- [使用 AWS X-Ray \(p. 302\)](#)

使用 Amazon CloudWatch

AWS Lambda 会自动替您监控 Lambda 函数，并通过 Amazon CloudWatch 报告各项指标。为帮助您监控代码的运行情况，Lambda 会自动跟踪请求数、每个请求的延迟、产生错误的请求数，并发布相关的 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](#)。

您可以借助 AWS Lambda 控制台、CloudWatch 控制台及其他 Amazon Web Services (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。

- [访问 AWS Lambda 的 Amazon CloudWatch 指标 \(p. 297\)](#)
- [AWS Lambda 指标 \(p. 300\)](#)

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，并将您的代码的所有日志推送到与 Lambda 函数关联的 CloudWatch Logs 组 (`/aws/lambda/<####>`)。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](#)。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 \(p. 299\)](#)。

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 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 [管理权限：使用 IAM 角色（执行角色）\(p. 339\)](#)。

AWS Lambda 诊断场景

下面的章节举例讲解如何使用 CloudWatch 的日志记录和监控功能监控和诊断 Lambda 函数。

诊断场景 1：Lambda 函数不按预期运行

在该场景中，您刚刚完成了 [教程：将 AWS Lambda 与 Amazon S3 结合使用 \(p. 166\)](#)。但您创建的“在创建 S3 对象时向 Amazon S3 上传缩略图的 Lambda 函数”不按预期运行。在向 Amazon S3 上传对象时，您发现该函数并未上传缩略图像。您可以通过以下方式诊断该问题。

确定您的 Lambda 函数为何不按预期运行

1. 检查代码并验证其是否正常运行。如果错误率增加，则表明该函数可能未正常工作。

您可以像对待任何其他 Node.js 函数一样在本地测试代码，也可以使用控制台的测试调用功能或使用 AWS CLI `Invoke` 命令在 Lambda 控制台中测试代码。每次为响应事件而执行该代码时，它都会向与 Lambda 函数关联的日志组（即 `/aws/lambda/<####>`）中写入一个日志条目。

以下是日志中可能会显示的错误的一些示例：

- 如果在日志中看到堆栈跟踪信息，则可能是代码存在错误。请审查代码并调试堆栈跟踪所指的错误。
- 如果在日志中看到 `permissions denied` 错误，则可能是所提供的用作执行角色的 IAM 角色没有必要权限。请检查 IAM 角色是否拥有访问您的代码引用的任何 AWS 资源的所有必要权限。为确保正确设置执行角色，请参阅[管理权限：使用 IAM 角色（执行角色）\(p. 339\)](#)。
- 如果在日志中看到 `timeout exceeded` 错误，则表明您的超时设置短于函数代码的运行时间。这可能是因为超时设置过低，或代码执行时间过长。
- 如果在日志中看到 `memory exceeded` 错误，则表明内存设置过低。请将其设置为更高的值。有关内存大小限制的信息，请参阅[CreateFunction \(p. 387\)](#)。更改内存设置时，也可能会改变使用期间的计费方式。有关定价的信息，请参阅[AWS Lambda 产品网站](#)。

2. 检查 Lambda 函数并验证其是否收到请求。

即使函数代码按预期运行，也能正确响应测试调用，但函数仍可能无法收到来自 Amazon S3 的请求。如果 Amazon S3 能够调用该函数，您应看到 CloudWatch 请求指标数值增加。如果 CloudWatch 请求数没有增加，请检查与该函数关联的访问权限策略。

诊断场景 2：Lambda 函数执行延迟增加

在该场景中，您刚刚完成了[教程：将 AWS Lambda 与 Amazon S3 结合使用 \(p. 166\)](#)。但您创建的“在创建 S3 对象时向 Amazon S3 上传缩略图的 Lambda 函数”不按预期运行。向 Amazon S3 上传对象时，您看到正在上传缩略图像，但代码执行时间长于预期。您可以通过几种不同的方法来诊断该问题。例如，您可以监控 Lambda 函数的延迟 CloudWatch 指标以确认延迟是否增加。或者，您也可以检查 Lambda 函数的 CloudWatch 错误数指标是否增加（如果增加，则可能是超时错误导致的）。

确定 Lambda 函数的执行延迟为何会增加

1. 使用不同的内存设置测试代码。

如果代码执行时间过长，可能是代码没有足够的计算资源来执行其逻辑。请尝试增加分配给函数的内存，并使用 Lambda 控制台的测试调用功能再次测试代码。在函数的日志条目中，您会看到占用内存、代码执行时间和所分配的内存。更改内存设置可能会改变对代码持续时间的收费方式。有关定价的信息，请参阅[AWS Lambda](#)。

2. 调查使用日志的执行瓶颈的来源。

您可以像对待任何其他 Node.js 函数一样在本地测试代码，也可以使用 Lambda 控制台的测试调用功能或 AWS CLI 的 `asyncInvoke` 命令在 Lambda 中测试代码。每次为响应事件而执行该代码时，它都会向与 Lambda 函数关联的日志组（即名为 `aws/lambda/<####>` 的日志组）中写入一个日志条目。在代码的各个部分（如对其他服务的调用）添加日志记录语句，了解执行代码的不同部分需要花费多长时间。

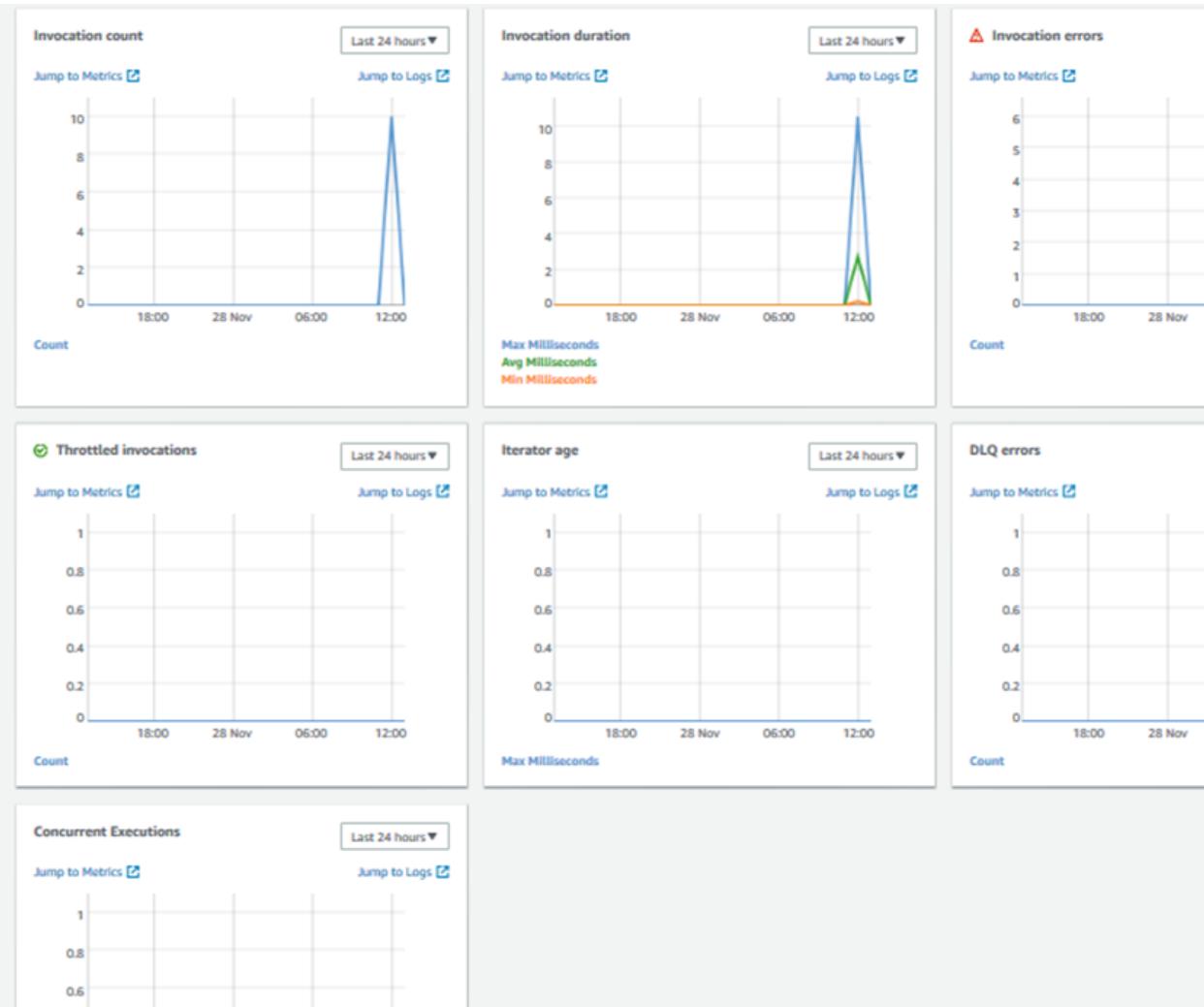
访问 AWS Lambda 的 Amazon CloudWatch 指标

AWS Lambda 会自动替您监控函数，并通过 Amazon CloudWatch 报告各项指标。上述指标包括总请求数、延迟和错误率。有关 Lambda 指标的更多信息，请参阅[AWS Lambda 指标 \(p. 300\)](#)。有关 CloudWatch 的更多信息，请参阅[Amazon CloudWatch 用户指南](#)。

您可以借助 Lambda 控制台、CloudWatch 控制台、AWS CLI 或 CloudWatch API 监控 Lambda 指标和查看日志。以下过程介绍如何使用这些不同的方式访问指标。

使用 Lambda 控制台访问指标

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. 在 Functions 页面上，选择函数名称，然后选择 Monitoring 选项卡。



所示为 Lambda 函数指标的图形化表示。

3. 选择 Jump to logs 以查看日志。

使用 CloudWatch 控制台访问指标

1. 通过以下网址打开 CloudWatch 控制台：<https://console.aws.amazon.com/cloudwatch/>。
2. 从导航栏中选择区域。
3. 在导航窗格中，选择 Metrics。
4. 在 CloudWatch Metrics by Category 窗格中，选择 Lambda Metrics。
5. (可选) 在图形窗格中，选择一个统计数据和时间段，然后使用这些设置创建 CloudWatch 警报。

使用 AWS CLI 访问指标

使用 [list-metrics](#) 和 [get-metric-statistics](#) 命令。

使用 CloudWatch CLI 访问指标

使用 [mon-list-metrics](#) 和 [mon-get-stats](#) 命令。

使用 CloudWatch API 访问指标

使用 [ListMetrics](#) 和 [GetMetricStatistics](#) 操作。

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。

<step>

如果您之前未创建 Lambda 函数，请参阅[入门 \(p. 3\)](#)。

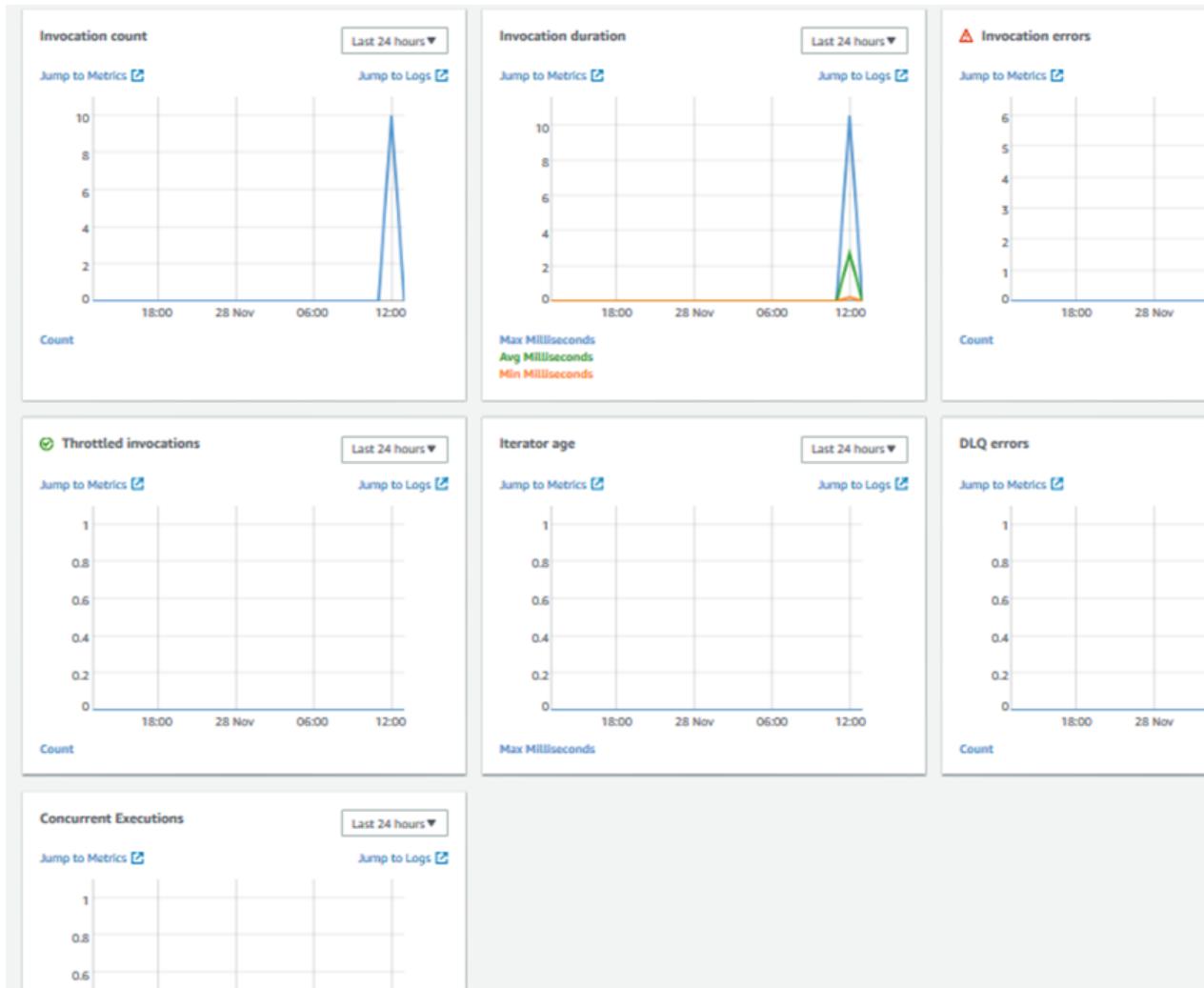
</step>

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. 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. 在 Functions 页面上，选择函数名称，然后选择 Monitoring 选项卡。



所示为 Lambda 函数指标的图形化表示。

3. 选择 Jump to logs 以查看日志。

有关访问 CloudWatch Logs 的更多信息，请参阅以下指南：

- [Amazon CloudWatch 用户指南](#)
- [Amazon CloudWatch Logs API Reference](#)
- [Amazon CloudWatch 用户指南 中的监控日志文件](#)

AWS Lambda 指标

本主题介绍 AWS Lambda 命名空间、指标和维度。AWS Lambda 会自动替您监控函数，并通过 Amazon CloudWatch 报告各项指标。这些指标包括总调用数、错误数、持续时间、限制、DLQ 错误和基于流的调用的迭代器期限。

CloudWatch 基本上是一个指标存储库。指标是 CloudWatch 中的核心概念，代表了数据点的时间顺序集。您（或 AWS 服务）可将指标数据点发布到 CloudWatch，并且您可以按一组有序的时间序列数据来检索关于这些数据点的统计数据。

指标通过名称、命名空间以及一个或多个维度进行唯一定义。每个数据点都有一个时间戳和一个度量单位（可选）。当请求统计数据时，返回的数据流根据命名空间、指标名称和维度加以识别。有关 CloudWatch 的更多信息，请参阅 [Amazon CloudWatch 用户指南](#)。

AWS Lambda CloudWatch 指标

AWS / Lambda 命名空间包括以下指标。

指标	描述
Invocations	<p>测量某个函数在响应某个事件或调用 API 调用时被调用的次数。这将替代已弃用的 RequestCount 指标。这包括成功调用和失败调用，但不包括被阻止的尝试。这等于函数的计费请求数。请注意，如果这些指标具有非零值，则 AWS Lambda 仅将这些指标发送到 CloudWatch。</p> <p>单位：计数</p>
Errors	<p>测量由于函数出错（响应代码 4XX）而导致失败的调用数。这将替代已弃用的 ErrorCount 指标。失败的调用可能会触发成功的重试尝试。这包括：</p> <ul style="list-style-type: none">已处理的异常（例如 context.fail(error)）导致代码退出的未处理的异常内存不足异常超时权限错误 <p>这不包括由于调用速率超出默认并行限制（错误代码 429）导致失败的调用或由于内部服务错误（错误代码 500）导致的失败。</p> <p>单位：计数</p>
Dead Letter Error	<p>当 Lambda 无法将失败的事件负载写入您配置的死信队列时将提高。原因可能如下：</p> <ul style="list-style-type: none">权限错误下游服务限制资源配置错误超时 <p>单位：计数</p>
Duration	<p>测量从函数代码作为调用的结果开始执行到其停止执行所经过的时间。这将替代已弃用的 Latency 指标。可能的最大数据点值为函数超时配置。计费的持续时间将向上摄入到近 100 毫秒。请注意，如果这些指标具有非零值，则 AWS Lambda 仅将这些指标发送到 CloudWatch。</p> <p>单位：毫秒</p>
Throttles	<p>测量由于调用速率超过客户的并行限制（错误代码 429）而被阻止的 Lambda 函数调用尝试次数。失败的调用可能会触发成功的重试尝试。</p> <p>单位：计数</p>
IteratorAge	<p>仅为基于流的调用发出（由 Amazon DynamoDB 流或 Kinesis 流触发的功能）。度量每一批已处理记录的最后一条记录的年限。年限是 Lambda 收到批次的时间与批次中的最后一条记录写入到流的时间之差。</p>

指标	描述
	单位：毫秒
ConcurrentExecutions	针对账户中的所有函数以及指定了自定义并发限制的函数，作为聚合指标发布。不适用于版本或别名。衡量给定函数在给定时间点并发执行的总和。如果在一个时间段内聚合，则必须将其视为平均指标。
	单位：计数
UnreservedConcurrentExecutions	仅作为账户中所有函数的聚合指标发出。不适用于函数、版本或别名。表示没有指定自定义并发限制的函数的并发总和。
	单位：计数

错误/调用比率

当计算 Lambda 函数调用的错误率时，区分调用请求和实际调用非常重要。错误率有可能超过已计费 Lambda 函数调用的数量。Lambda 仅在执行 Lambda 函数代码时才报告调用指标。如果调用请求产生某种限制或其他初始化错误，并因此阻止调用 Lambda 函数代码，则 Lambda 将报告错误，但不记录调用指标。

- 函数执行时，Lambda 会发出 Invocations=1。如果未执行 Lambda 函数，则什么也不发出。
- Lambda 为每个调用请求发出一个 Errors 数据点。Errors=0 表示不存在函数执行错误。Errors=1 表示存在函数执行错误。
- Lambda 为每个调用请求发出一个 Throttles 数据点。Throttles=0 表示不存在调用限制。Throttles=1 表示存在调用限制。

AWS Lambda CloudWatch 维度

您可以用下表中的维度来优化针对您的 Lambda 函数返回的指标。

维度	描述
FunctionName	按 Lambda 函数筛选指标数据。
Resource	按 Lambda 函数资源（如函数版本或别名）筛选指标数据。
Version	筛选您为 Lambda 版本请求的数据。
Alias	筛选您为 Lambda 别名请求的数据。
Executed Version	按 Lambda 函数版本筛选指标数据。这仅适用于别名调用。

使用 AWS X-Ray

基于 Lambda 的典型应用程序包含一个或多个通过事件（如将对象上传到 Amazon S3、Amazon SNS 通知和 API 操作）触发的函数。这些函数在触发后通常会调用下游资源，如 DynamoDB 表或 Amazon S3 存储桶，或执行其他 API 调用。AWS Lambda 针对函数的所有调用，利用 Amazon CloudWatch 自动发送指标和日志。但是，此机制可能无法方便地跟踪调用您的 Lambda 函数的事件源，或跟踪函数进行的下游调用。有关跟踪的工作方式的完整概述，请参阅 [AWS X-Ray](#)。

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 \(p. 303\)](#) 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

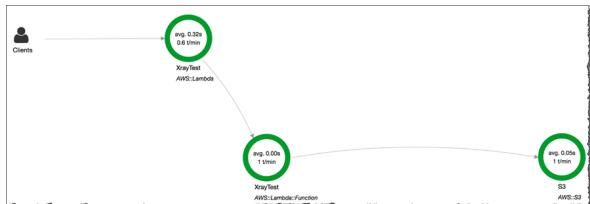
只有目前与 X-Ray 集成的服务可在您的 Lambda 跟踪之外显示为独立跟踪。有关当前支持 X-Ray 的服务列表，请参阅[将 AWS X-Ray 与其他 AWS 服务集成](#)。

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 函数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 服务的调用。
- 自定义子分段 - 自定义子分段或用户注释，可以使用 X-Ray 开发工具包添加到 Lambda 函数分段中。

Note

对于每个跟踪调用，Lambda 会发送 Lambda 服务分段及其所有子分段。这些片段无论运行时如何都将发出，并要求您使用 XRay SDK for AWS API 调用。

利用 Lambda 设置 AWS X-Ray

以下是利用 Lambda 设置 X-Ray 的详细信息。

开始前的准备工作

要使用 Lambda CLI 启用对 Lambda 函数的跟踪，您必须首先对函数的执行角色添加跟踪权限。要实现此目的，请执行以下步骤：

- 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
- 查找您的 Lambda 函数的执行角色。
- 附加以下托管策略：AWSXrayWriteOnlyAccess

要了解有关这些策略的更多信息，请参阅 [AWS X-Ray](#)。

如果您要使用 Lambda 控制台将跟踪模式更改为“活跃”，将如下节所述，自动添加跟踪权限。

跟踪

通过跟踪 ID 来跟踪请求在您的应用程序中传输的路径。跟踪会收集单个请求 (通常是 HTTP GET 或 POST 请求) 生成的所有分段。

Lambda 函数的跟踪有两种模式：

- 传递：如果您为函数的执行角色添加了跟踪权限，这是所有 Lambda 函数的默认设置。此方法意味着，只有上游服务 (例如 AWS Elastic Beanstalk) 启用 X-Ray 后才会跟踪 Lambda 函数。
- 活跃：如果 Lambda 函数具有此设置，Lambda 会自动根据 X-Ray 指定的采样算法对调用请求进行采样。

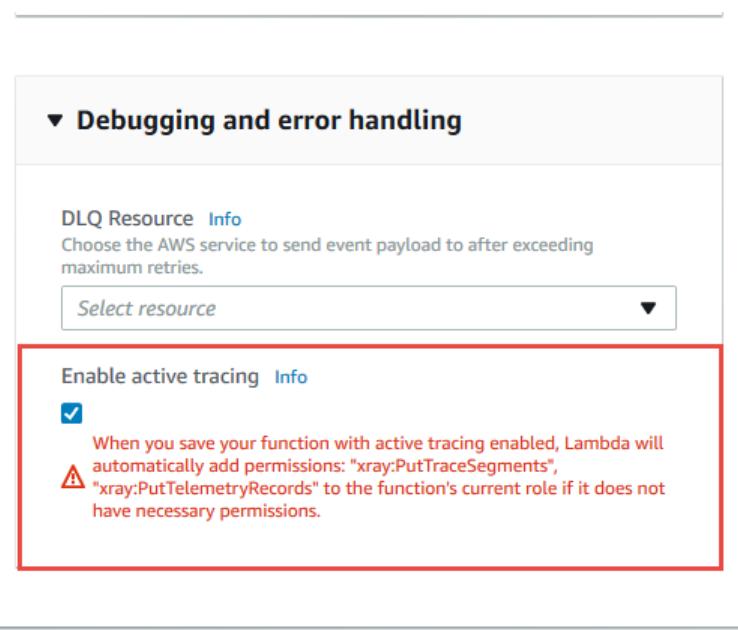
Note

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。默认的采样算法是每分钟 1 个请求，超过此限制的请求采样 5%。但是，如果函数的流量不大，采样率可能会增大。

您可以使用 Lambda 管理控制台或 Lambda [CreateFunction \(p. 387\)](#) 或 [UpdateFunctionConfiguration \(p. 477\)](#) API 操作，更改 Lambda 函数的跟踪模式。

如果您使用 Lambda 控制台，则适用以下规则：

- 如果您将函数的跟踪模式更改为“活跃”，跟踪权限会自动附加到该函数的执行角色。如果收到错误信息，说明 Lambda 无法将 AWSXrayWriteOnlyAccess 策略添加到函数的执行角色，请登录 IAM 控制台 <https://console.aws.amazon.com/iam/>，手动添加策略。
- 要启用活动跟踪，请转到 Configuration 选项卡，然后选择 Enable active tracing 框。



如果您使用 Lambda [CreateFunction \(p. 387\)](#) 或 [UpdateFunctionConfiguration \(p. 477\)](#) API 操作：

- 如果您希望跟踪模式为“活跃”，请将 `TracingConfig` 参数的 `Mode` 属性设置为 `Active`。请注意，任何新函数的默认跟踪模式都是 `PassThrough`。
- 任何新建或更新的 Lambda 函数的 `$LATEST` 版本均会设置为您指定的值。

Note

如果您没有为函数的执行角色添加跟踪权限，将收到一条错误消息。有关更多信息，请参阅 [开始前的准备工作 \(p. 304\)](#)。

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，任何分析代码都必须在 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：

- [Node.js \(p. 305\)](#)
- [Java \(p. 307\)](#)
- [Python \(p. 308\)](#)
- [转到 \(p. 310\)](#)

Node.js

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：

```
var AWSXRay = require('aws-xray-sdk-core');
var 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：

```
s3Client = 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：

```
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));

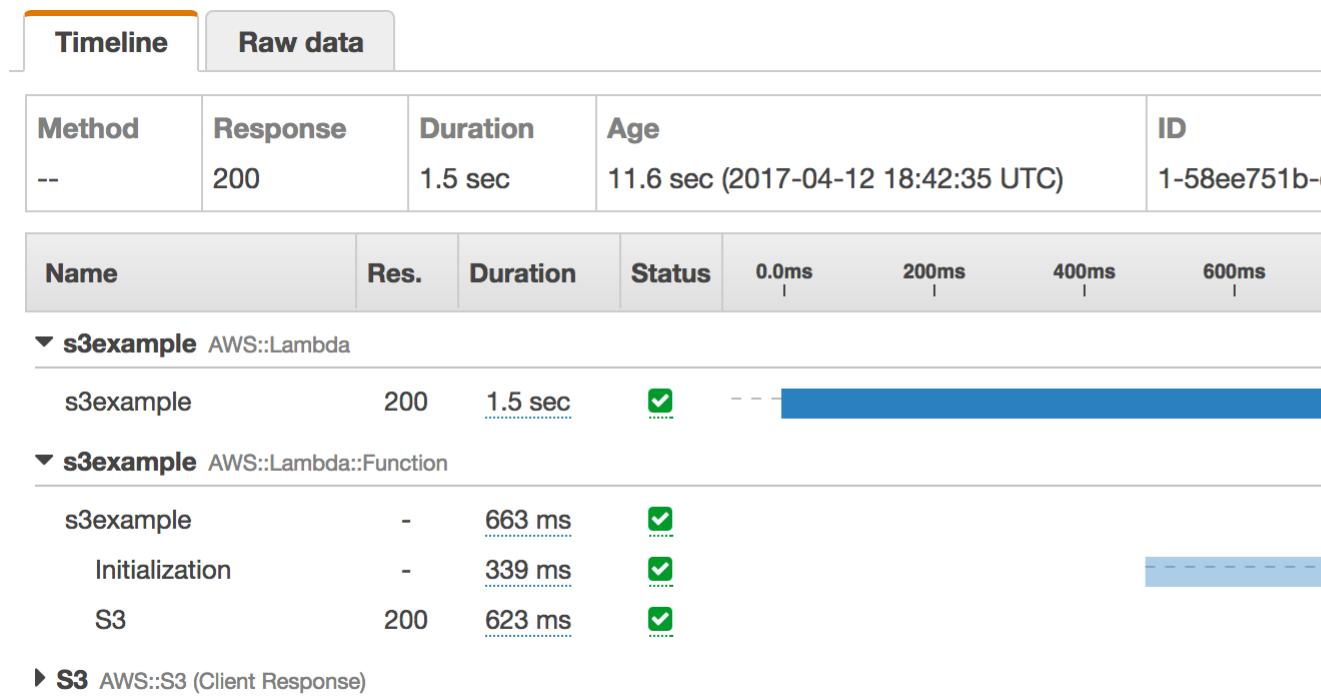
s3 = new AWS.S3({signatureVersion: 'v4'});

exports.handler = (event, context, callback) => {

    var params = {Bucket: BUCKET_NAME, Key: BUCKET_KEY, Body: BODY};

    s3.putObject(params, function(err, data) {
        if (err)
            { console.log(err) }
        else {
            console.log('success!')
        }
    });
};
```

以下是上述代码发送的跟踪的样子（异步调用）：



Java

在 Java 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。要利用此功能，请在您的部署程序包中包括适用于 Java 的 AWS X-Ray 开发工具包。无需更改代码。只要您使用的 AWS 开发工具包版本为 1.11.48 或更高版本，则不需要添加任何其他代码行，就可以跟踪您的函数的下游调用。

AWS 开发工具包将动态导入 X-Ray 开发工具包，针对您的函数进行的下游调用发送子分段。您可以使用适用于 Java 的 X-Ray 开发工具包检测代码，从而发送自定义子分段和/或在 X-Ray 分段中添加注释。

以下示例使用适用于 Java 的 X-Ray 开发工具包检测 Lambda 函数，以发送自定义子分段并将自定义注释发送到 X-Ray：

```
package uptime;

import java.io.IOException;
import java.time.Instant;
import java.util.HashMap;
import java.util.Map;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;

import com.amazonaws.regions.Regions;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDB;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClientBuilder;
import com.amazonaws.services.dynamodbv2.model.AttributeValue;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.xray.AWSXRay;
import com.amazonaws.xray.proxies.apache.http.HttpClientBuilder;

public class Hello {
    private static final Log logger = LogFactory.getLog(Hello.class);

    private static final AmazonDynamoDB dynamoClient;
    private static final HttpClient httpClient;

    static {
        dynamoClient =
AmazonDynamoDBClientBuilder.standard().withRegion(Regions.US_EAST_1).build();
        httpClient = HttpClientBuilder.create().build();
    }
    public void checkUptime(Context context) {
        AWSXRay.createSubsegment("makeRequest", (subsegment) -> {

            HttpGet request = new HttpGet("https://aws.amazon.com/");
            boolean is2xx = false;

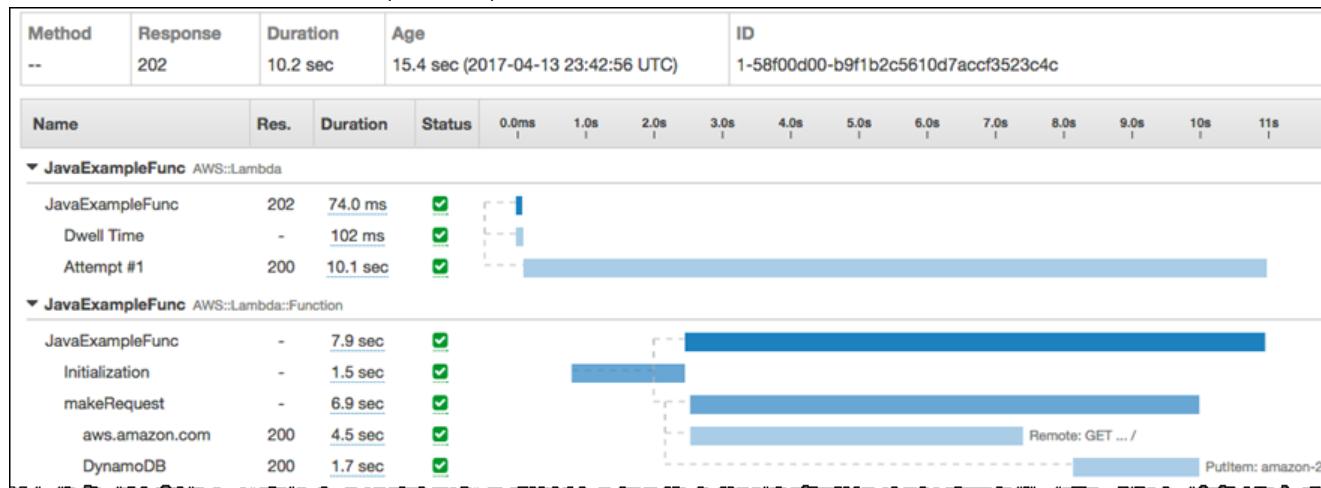
            try {
                HttpResponse response = httpClient.execute(request);
                is2xx = (response.getStatusLine().getStatusCode() / 100) == 2;
                subsegment.putAnnotation("statusCode",
response.getStatusLine().getStatusCode());
            } catch (IOException ioe) {
                logger.error(ioe);
            }
            Map<String, AttributeValue> item = new HashMap<>();
            item.put("Timestamp", new AttributeValue().withN("" +
Instant.now().getEpochSecond()));
        });
    }
}
```

```

        item.put("2xx", new AttributeValue().withBOOL(is2xx));
        dynamoClient.putItem("amazon-2xx", item);
    });
}
}

```

以下是上述代码发送的跟踪的样子 (同步调用) :



Python

在 Python 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 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.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)
```

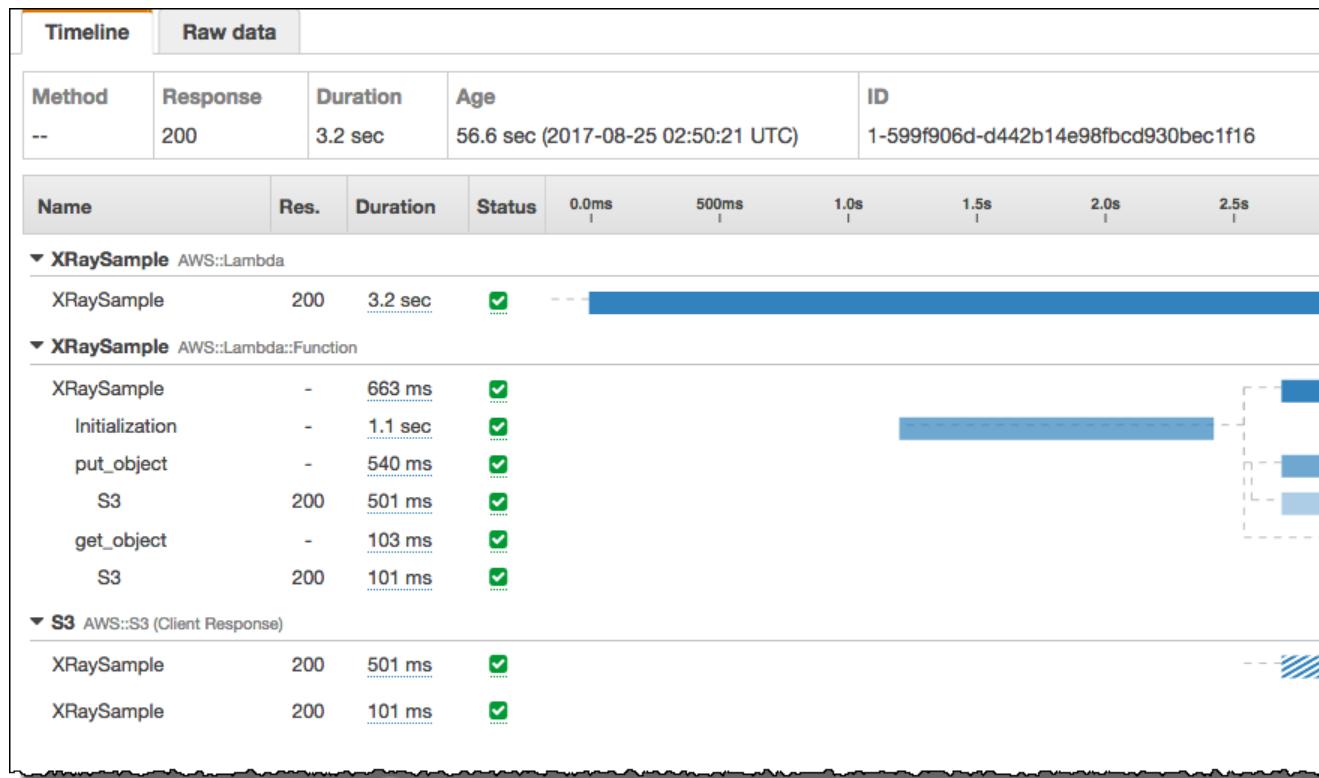
Note

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)：



转到

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 [Context 对象 \(Go\) \(p. 63\)](#) 作为其第一个参数，则该对象可传递给 X-Ray 软件开发工具包。Lambda 通过此上下文传递软件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  
}
```

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，如果为 Lambda 函数分配 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

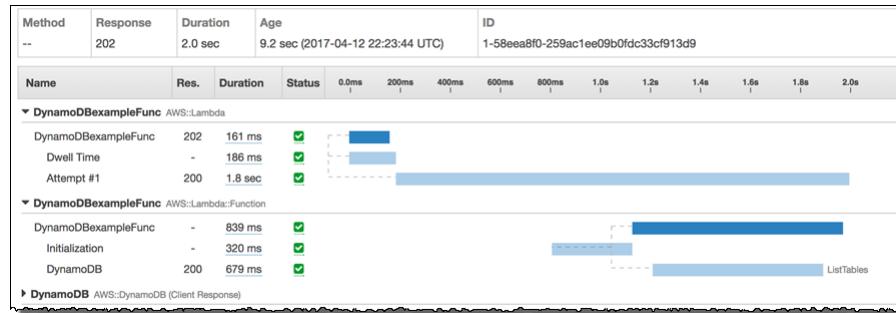
AWS Lambda 会自动生成三个环境变量，方便与 X-Ray 守护程序进行通信，并设置 X-Ray 开发工具包的配置：

- `_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 `_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 将此值设置为 `LOG_ERROR`。
- `AWS_XRAY_DAEMON_ADDRESS`：此环境变量公开 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。

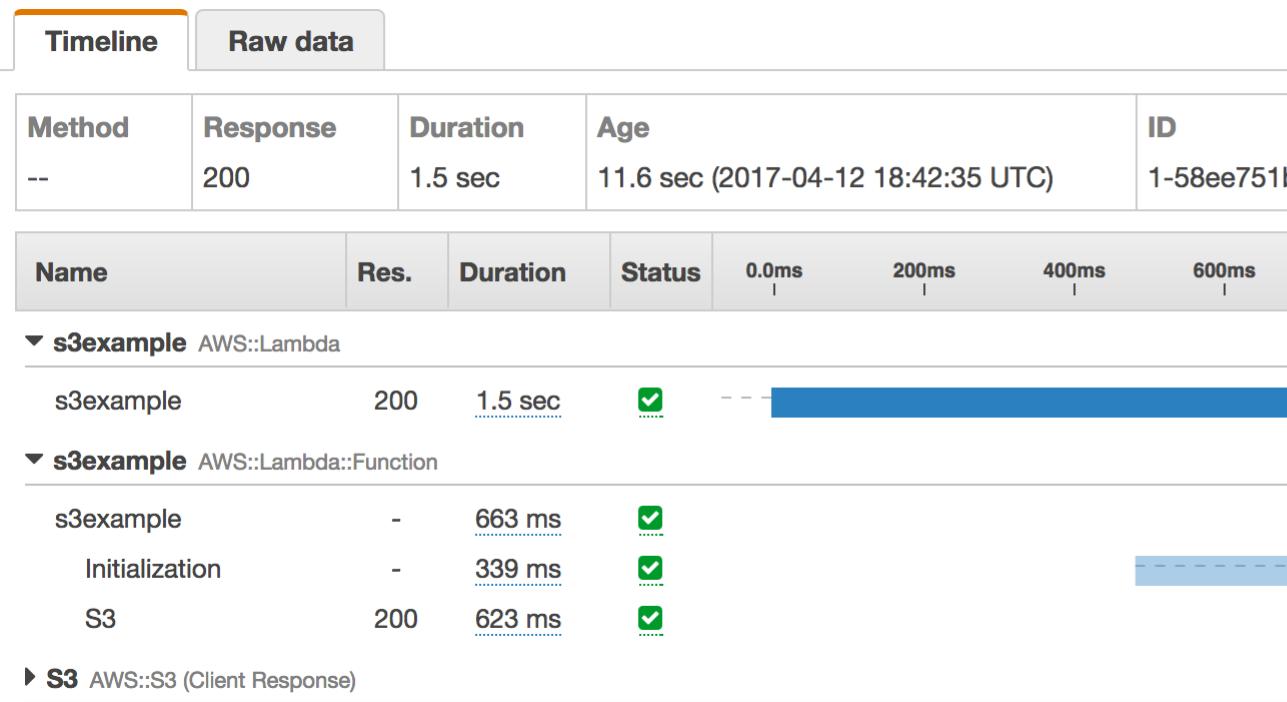
- 异步 - 以下示例展示具有一个成功调用和一个对 DynamoDB 进行下游调用的异步 Lambda 请求。



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](#)。

管理基于 Lambda 的应用程序

AWS Lambda 与 AWS 提供的许多管理工具集成，例如 AWS 标签、AWS CloudTrail 和 AWS IAM。以下各部分提供了有关如何管理基于 Lambda 的应用程序的指导信息（包括使用标签组织基于 Lambda 的应用程序、使用 CloudTrail 审计 AWS 上的活动），并向您介绍 AWS 安全模型以了解如何保护基于 Lambda 的应用程序。我们还讨论了 AWS Lambda 独有的管理任务，它将管理 Lambda 函数的并发执行行为。

以下各部分提供了有关如何组织和跟踪 Lambda 函数调用的指导信息，并向您介绍 AWS 安全模型以了解如何保护基于 Lambda 的应用程序：

标记 Lambda 函数

Lambda 函数可跨越不同区域覆盖多个应用程序。为了在跟踪每个函数调用的频率和成本时简化流程，您可以使用标签。标签是您附加到 AWS 资源的键/值对，以更好地组织这些资源。如果您具有同一类型的许多资源（对于 AWS Lambda 而言是函数），标签特别有用。通过使用标签，拥有数百个 Lambda 函数的客户可以筛选包含相同标签的函数，从而轻松地访问和分析一组特定函数。标记 Lambda 函数的两个主要优势是：

- 分组和筛选：通过应用标签，您可以使用 Lambda 控制台或 CLI 分离一组特定应用程序或结算部门中包含的 Lambda 函数。有关更多信息，请参阅 [筛选标记的 Lambda 函数 \(p. 316\)](#)。
- 成本分配：由于 Lambda 对标记的支持已与 AWS Billing 相集成，您可以将账单细分为多个动态类别，并将函数与特定成本中心相对应。例如，如果您利用 "Department" 键标记所有 Lambda 函数，可以按部门细分所有 AWS Lambda 成本。然后，您可以提供单个部门值，例如 "Department 1" 或 "Department 2"，将函数调用成本指向相应的成本中心。成本分配是以详细账单报告的形式呈现的，便于您对 AWS 成本进行分类和跟踪。

主题

- [为账单标记 Lambda 函数 \(p. 314\)](#)
- [对 Lambda 函数应用标签 \(p. 314\)](#)
- [筛选标记的 Lambda 函数 \(p. 316\)](#)
- [标签限制 \(p. 317\)](#)

为账单标记 Lambda 函数

您可以使用标签来管理 AWS 账单，使其反映您的成本结构。要执行此操作，您可以添加标签键，其值将包含在成本分配报告中。有关设置成本分配报告，在报告中将选定标签键作为行项目的更多信息，请参阅 AWS Account Billing 简介中的 [月度成本分配报告](#)。

如需查看组合资源的成本，请按具有相同标签键值的函数组织您的账单信息。例如，您可以将特定的应用程序名称用作几个 Lambda 函数的标签，然后组织账单信息，以查看在数个服务中使用该应用程序的总成本。有关更多信息，请参阅 AWS 账单和成本管理用户指南中的 [使用成本分配标签](#)。

Important

在 AWS Lambda 中，函数是可以标记的唯一资源。无法标记别名或特定函数版本。对函数别名或版本的调用将作为对原始函数的调用进行计费。

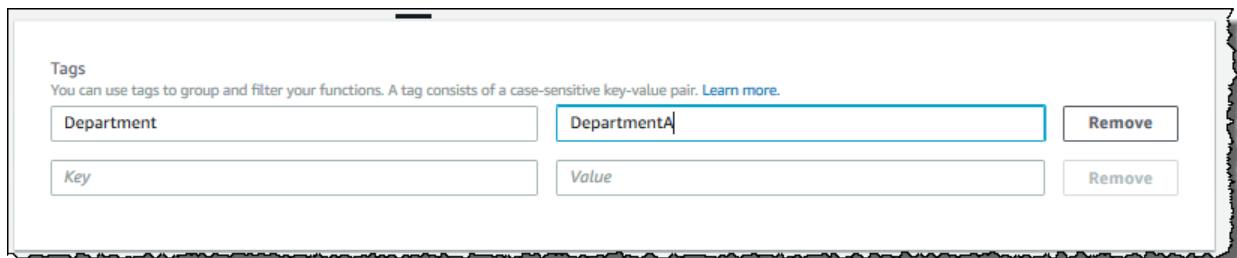
对 Lambda 函数应用标签

对 Lambda 函数的标记方式取决于您创建函数的方式。您可以使用 Lambda 控制台或 CLI 应用标签，如以下部分所述：

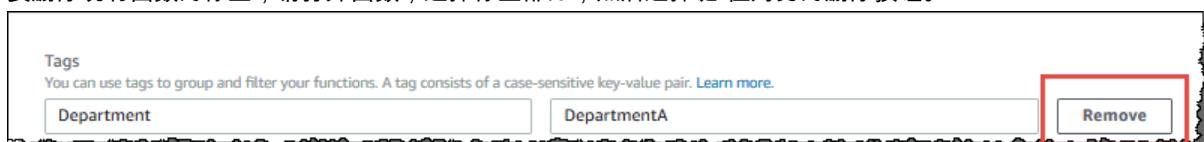
- 使用控制台对 Lambda 函数应用标签 (p. 315)
- 使用 CLI 对 Lambda 函数应用标签 (p. 315)

使用控制台对 Lambda 函数应用标签

您可以在配置选项卡中的标签部分下面向函数添加标签。



要删除现有函数的标签，请打开函数，选择标签部分，然后选择键-值对旁的删除按钮。



使用 CLI 对 Lambda 函数应用标签

当您使用 [CreateFunction \(p. 387\)](#) 命令创建新的 Lambda 函数时，可以填充 `Tags` 参数来添加标签。要指定多个标签值，请用引号引起来，如下所示：

Note

如果您尚未创建 `adminuser` 配置文件，请参阅[设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)。

```
$ aws lambda create-function \
--region region \
--function-name function-name
--role role-arn \
--handler handler-name \
--runtime runtime-value \
--runtime runtime \
--tags "DEPARTMENT=Department A, Department B" \
--profile adminuser \
--timeout 10 \
--memory-size 1024
```

要针对现有函数应用或添加更多标签，您可以使用 [TagResource \(p. 458\)](#) API，并提供 Lambda 函数 ARN (Amazon 资源名称) 和组成标签的键/值对。

```
$ aws lambda tag-resource \
--resource function arn \
--tags DEPARTMENT="Department C, Department D"
```

相反，如果您要删除 Lambda 函数的任意或所有标签，可以使用 [UntagResource \(p. 460\)](#) API 并再次提供函数 ARN (Amazon 资源名称)，以及要从函数中删除的标签键列表。

```
$ aws lambda untag-resource \
--resource function arn \
--tagkeys list of tag keys to be removed
```

筛选标记的 Lambda 函数

使用标签将 Lambda 函数分组后，您即可利用 Lambda 控制台或 AWS CLI 提供的筛选功能，根据您的特定要求查看函数。

使用控制台筛选 Lambda 函数

Lambda 控制台中包含一个搜索字段，可以根据一组指定的函数属性（包括标签）筛选函数。假设您有两个函数，名为 MyFunction 和 MyFunction2，它们都有名为 Department 的标签键。要查看这些函数，请选择搜索字段，注意将自动筛选出包含标签键的列表：

The screenshot shows the AWS Lambda 'Functions (25)' page. On the left, there is a sidebar with a 'Filter by tags and attributes or search by keyword' input field containing 'tag:Department'. Below it is a table with columns for 'Function attributes' (Description, Function name, Runtime, Tags, Department) and columns for 'Runtime' and 'Code size'. There are three entries in the table:

Function attributes	Runtime	Code size	
Description	mbda function.	Node.js 6.10	333 bytes
Function name	mbda function.	Node.js 6.10	333 bytes
Runtime	mbda function.	Python 2.7	360 bytes
Tags			
Department			

选择 Department 键。Lambda 将返回包含该键的所有函数。

现在，假设 MyFunction 标签的键值为“Department A”，MyFunction2 的键值为“Department B”。您可以通过选择 Department 键的值将搜索范围缩小，在本例中为 Department A，如下所示。

The screenshot shows the AWS Lambda 'Functions (25)' page with a search filter 'tag:Department:' followed by 'DepartmentA'. The results table shows two entries:

Function attributes	Runtime	Code size	
Description	mbda function.	Node.js 6.10	333 bytes
Function name	mbda function.	Node.js 6.10	333 bytes
Runtime	mbda function.	Python 2.7	360 bytes
Tags			
Department			
DepartmentA			
DepartmentB			

这样仅返回 MyFunction。

您还可以包含其他接受的函数属性，如说明、函数名称或运行时，进一步缩小搜索范围。

Note

每个 Lambda 函数最多支持 50 个标签。如果您删除 Lambda 函数，关联的标签也将被删除。

使用 CLI 筛选 Lambda 函数

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 \(p. 442\)](#)：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 \
--region region \
--profile adminuser
```

- [GetFunction \(p. 413\)](#)：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 function name \
--region region \
--profile adminuser
```

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](#)。

标签限制

以下限制适用于标签：

- 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 + - = . _ : / @。

使用 AWS CloudTrail 记录 AWS Lambda API 调用

AWS Lambda 与 AWS CloudTrail 集成在一起，后者是一种服务，它在 AWS 账户中捕获由 AWS Lambda 或代表它发出的 API 调用，并将日志文件传输到您指定的 Amazon S3 存储桶。CloudTrail 捕获从 AWS Lambda 控制台或从 AWS Lambda API 发出的 API 调用。通过使用 CloudTrail 收集的信息，您可以确定对 AWS Lambda 发出了什么请求、发出请求的源 IP 地址、何人发出的请求以及发出请求的时间等。要了解有关 CloudTrail 的更多信息，包括如何对其进行配置和启用，请参阅 [AWS CloudTrail User Guide](#)。

CloudTrail 中的 AWS Lambda 信息

在您的 AWS 账户中启用 CloudTrail 日志记录后，将在日志文件中跟踪对 AWS Lambda 操作执行的 API 调用。AWS Lambda 记录与其他 AWS 服务记录一起写入日志文件。CloudTrail 基于时间段和文件大小来确定何时创建新文件并向其写入内容。

支持以下操作：

- [AddPermission \(p. 373\)](#)
- [CreateEventSourceMapping \(p. 382\)](#)

- [CreateFunction \(p. 387\)](#)
(`CreateFunction` 的 CloudTrail 日志省略了 `zipFile` 参数。)
- [DeleteEventSourceMapping \(p. 397\)](#)
- [DeleteFunction \(p. 400\)](#)
- [GetEventSourceMapping \(p. 410\)](#)
- [GetFunction \(p. 413\)](#)
- [GetFunctionConfiguration \(p. 417\)](#)
- [GetPolicy \(p. 422\)](#)
- [ListEventSourceMappings \(p. 436\)](#)
- [ListFunctions \(p. 439\)](#)
- [RemovePermission \(p. 455\)](#)
- [UpdateEventSourceMapping \(p. 466\)](#)
- [UpdateFunctionCode \(p. 470\)](#)
(`UpdateFunctionCode` 的 CloudTrail 日志省略了 `zipFile` 参数。)
- [UpdateFunctionConfiguration \(p. 477\)](#)

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 用户凭证发出，通过某个角色或联合用户的临时安全凭证发出，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 [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",  
      "invokedFunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:myFunction",  
      "invocationType": "RequestResponse",  
      "logEvent": "{'functionName': 'myFunction', 'invocationType': 'RequestResponse', 'logStreamName': '2015/03/18/[$Function-$InvocationID]', 'logFile': '2015/03/18/[$Function-$InvocationID].log', 'logSize': 123456789, 'logTimestamp': '2015-03-18T19:03:36Z'}  
    }  
  ]  
}
```

```
"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。

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。此功能在默认情况下未启用，如果启用此功能，将产生额外费用。您可以使用 AWS CloudTrail 控制台或 [Invoke \(p. 425\)](#) CLI 操作来实现此目的。有关此选项的更多信息，请参阅[记录跟踪的数据和管理事件](#)。

AWS Lambda 的身份验证和访问控制

访问 AWS Lambda 时需要有 AWS 可以对您的请求进行身份验证的凭证。这些凭证必须有权访问 AWS 资源，如 AWS Lambda 函数或 Amazon S3 存储桶。下面几节提供详细的信息来说明如何使用 [AWS Identity and Access Management \(IAM\)](#) 和 Lambda 控制谁能访问您的资源，从而对这些资源进行保护：

- [身份验证 \(p. 320\)](#)
- [访问控制 \(p. 320\)](#)

身份验证

您可以以下面任一类型的身份访问 AWS：

- 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，使用您创建账户时所用的电子邮件地址和密码登录，即可获得该身份。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[使用根用户的最佳实践，仅将其用于创建您的首个 IAM 用户](#)。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 – [IAM 用户](#)是您的 AWS 账户中的一种身份，它具有特定的自定义权限（例如，用于在 Lambda 中创建 a function 的权限）。您可以使用 IAM 用户名和密码来登录以保护 AWS 网页，如 [AWS 管理控制台](#)、[AWS 开发论坛](#)或 [AWS Support Center](#)。

除了用户名和密码之外，您还可以为每个用户生成[访问密钥](#)。在通过[多个软件开发工具包](#)之一或使用 [AWS Command Line Interface \(CLI\)](#) 以编程方式访问 AWS 服务时，可以使用这些密钥。SDK 和 CLI 工具使用访问密钥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。Lambda supports 签名版本 4，后者是一种用于对入站 API 请求进行身份验证的协议。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](#)。

- IAM 角色 – [IAM 角色](#)是可在账户中创建的一种具有特定权限的 IAM 身份。它类似于 IAM 用户，但未与特定人员关联。利用 IAM 角色，您可以获得可用于访问 AWS 服务和资源的临时访问密钥。具有临时凭证的 IAM 角色在以下情况下很有用：
 - 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 身份提供商的既有用户身份。他们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](#)。
 - AWS 服务访问 – 您可以使用您的账户中的 IAM 角色向 AWS 服务授予对您账户中资源的访问权限。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，然后将该存储桶提供的数据加载到 Amazon Redshift 群集中。有关更多信息，请参阅 IAM 用户指南 中的[创建向 AWS 服务委派权限的角色](#)。
 - 运行在 Amazon EC2 上的应用程序 – 对于在 EC2 实例上运行、并发出 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](#)。

访问控制

您可以拥有有效的凭证来对自己的请求进行身份验证，但您还必须拥有权限才能创建或访问 AWS Lambda 资源。例如，您必须拥有创建 Lambda 函数，添加事件源和发布您的 Lambda 函数版本的权限。

以下部分介绍如何管理 AWS Lambda 的权限。我们建议您先阅读概述。

- 管理您的 AWS Lambda 资源的访问权限概述 (p. 321)
- 为 AWS Lambda 使用基于身份的策略 (IAM 策略) (p. 324)
- 对 AWS Lambda 使用基于资源的策略 (Lambda 函数策略) (p. 336)

管理您的 AWS Lambda 资源的访问权限概述

每个 AWS 资源都归某个 AWS 账户所有，创建和访问资源的权限由权限策略进行管理。账户管理员可以向 IAM 身份 (即：用户、组和角色) 附加权限策略，某些服务 (如 AWS Lambda) 也支持向资源附加权限策略。

Note

账户管理员 (或管理员用户) 是具有管理员权限的用户。有关更多信息，请参阅 IAM 用户指南 中的 [IAM 最佳实践](#)。

在授予权限时，您要决定谁获得权限，获得对哪些资源的权限，以及您允许对这些资源执行的具体操作。

主题

- [AWS Lambda 资源和操作 \(p. 321\)](#)
- [了解资源所有权 \(p. 321\)](#)
- [管理对资源的访问 \(p. 322\)](#)
- [指定策略元素：操作、效果、资源和委托人 \(p. 323\)](#)
- [在策略中指定条件 \(p. 324\)](#)

AWS Lambda 资源和操作

在 AWS Lambda 中，主要资源为 Lambda 函数 和事件源映射。您可以在 AWS Lambda 拉模型中创建事件源映射来将 Lambda 函数与事件源关联。有关更多信息，请参阅 [事件源映射 \(p. 142\)](#)。

AWS Lambda 还支持其他资源类型、别名 和版本。但是，只能在现有的 Lambda 函数范围内创建别名和版本。这些资源称作子资源。

这些资源和子资源具有与其关联的唯一 Amazon 资源名称 (ARN)，如下表所示。

资源类型	ARN 格式
函数	arn:aws:lambda: <i>region:account-id:function:function-name</i>
函数别名	arn:aws:lambda: <i>region:account-id:function:function-name:alias-name</i>
函数版本	arn:aws:lambda: <i>region:account-id:function:function-name:version</i>
事件源映射	arn:aws:lambda: <i>region:account-id:event-source-mapping:event-source-mapping-id</i>

AWS Lambda 提供一组操作来处理 Lambda 资源。有关可用操作的列表，请参阅 [Actions \(p. 371\)](#)。

了解资源所有权

资源所有者 是创建资源的 AWS 账户。也就是说，资源所有者是委托人实体 (根账户、IAM 用户或 IAM 角色) 的 AWS 账户。以下示例说明了它的工作原理：

- 如果使用您的 AWS 账户的根账户凭证来创建 Lambda 函数，则您的 AWS 账户就是该资源的所有者（在 Lambda 中，资源就是 Lambda 函数）。
- 如果您在您的 AWS 账户中创建 IAM 用户并对其授予创建 Lambda 函数的权限，则该用户可以创建 Lambda 函数。但是，您的 AWS 账户（也就是该用户所属的账户）拥有 Lambda 函数资源。
- 如果您在您的 AWS 账户中创建 IAM 角色，该角色具有创建 Lambda 函数的权限，则能够担任该角色的任何人都可以创建 Lambda 函数。您的 AWS 账户（也就是该用户所属的账户）拥有 Lambda 函数资源。

管理对资源的访问

权限策略 规定谁可以访问哪些内容。下一节介绍创建权限策略时的可用选项。

Note

本节讨论如何在 AWS Lambda 范围内使用 IAM。它不提供有关 IAM 服务的详细信息。有关完整的 IAM 文档，请参阅[什么是 IAM？](#)（在 IAM 用户指南 中）。有关 IAM 策略语法和介绍的信息，请参阅 IAM 用户指南 中的[AWS IAM 策略参考](#)。

附加到 IAM 身份的策略称作基于身份的策略（IAM 策略），附加到资源的策略称作基于资源的资源。AWS Lambda 同时支持基于身份的策略（IAM 策略）和基于资源的策略。

主题

- [基于身份的策略 \(IAM 策略\) \(p. 322\)](#)
- [基于资源的策略 \(Lambda 函数策略\) \(p. 323\)](#)

基于身份的策略 (IAM 策略)

您可以向 IAM 身份附加策略。例如，您可以执行以下操作：

- 向您账户中的用户或组附加权限策略 - 账户管理员可以使用与特定用户关联的权限策略授予该用户创建 Lambda 函数的权限。
- 向角色附加权限策略（授予跨账户权限）- 您可以向 IAM 角色附加基于身份的权限策略，以授予跨账户的权限。例如，账户 A 中的管理员可以创建一个角色，以向其他 AWS 账户（如账户 B）或某项 AWS 服务授予跨账户权限，如下所述：
 1. 账户 A 管理员可以创建一个 IAM 角色，然后向该角色附加授予其访问账户 A 中资源的权限策略。
 2. 账户 A 管理员可以向将账户 B 标识为能够担任该角色的委托人的角色附加信任策略。
 3. 之后，账户 B 管理员可以委派权限，指派账户 B 中的任何用户担任该角色。这样，账户 B 中的用户就可以创建或访问账户 A 中的资源了。如果您需要授予 AWS 服务权限来担任该角色，则信任策略中的委托人也可以是 AWS 服务委托人。

有关使用 IAM 委派权限的更多信息，请参阅 IAM 用户指南 中的[访问权限管理](#)。

以下示例策略授予对所有资源执行 `lambda>ListFunctions` 操作的权限。在当前实现中，Lambda 不支持使用某些 API 操作的资源 ARN（也称为资源级权限）标识特定资源，因此必须指定通配符 (*)。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ListExistingFunctions",  
            "Effect": "Allow",  
            "Action": [  
                "lambda>ListFunctions"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

```
    ]  
}
```

有关将基于身份的策略用于 Lambda 的更多信息，请参阅[为 AWS Lambda 使用基于身份的策略 \(IAM 策略\) \(p. 324\)](#)。有关用户、组、角色和权限的更多信息，请参阅 IAM 用户指南中的[身份 \(用户、组和角色\)](#)。

基于资源的策略 (Lambda 函数策略)

每个 Lambda 函数都可以有关联的基于资源的权限策略。对于 Lambda，Lambda 函数是主要资源，这些策略称作 Lambda 函数策略。作为将基于身份的策略用于 IAM 角色的替代方案，您可以使用 Lambda 函数策略授予跨账户权限。例如，您可以通过向 Lambda 函数策略添加权限而不是创建 IAM 角色来向 Amazon S3 授予调用您 Lambda 函数的权限。

Important

Lambda 函数策略主要是在以下情况下使用：您在 AWS Lambda 中设置事件源以向服务或事件源授予调用 Lambda 函数的权限（请参阅[Invoke \(p. 425\)](#)）。此情况的一个例外是在事件源（例如 Amazon DynamoDB 或 Kinesis）使用拉模式（其中，权限改为在 Lambda 函数执行角色中进行管理）的情况下。有关更多信息，请参阅[事件源映射 \(p. 142\)](#)。

以下是具有一个声明的示例 Lambda 函数策略。此声明向名为 HelloWorld 的 Lambda 函数上的 lambda:InvokeFunction 操作授予 Amazon S3 服务委托人权限。此条件可确保发生事件的存储桶由拥有该 Lambda 函数的同一账户拥有。

```
{  
  "Policy":{  
    "Version":"2012-10-17",  
    "Statement": [  
      {  
        "Effect":"Allow",  
        "Principal":{  
          "Service":"s3.amazonaws.com"  
        },  
        "Action":"lambda:InvokeFunction",  
        "Resource":"arn:aws:lambda:region:account-id:function>HelloWorld",  
        "Sid":"65bafc90-6a1f-42a8-a7ab-8aa9bc877985",  
        "Condition":{  
          "StringEquals":{  
            "AWS:SourceAccount": "account-id"  
          },  
          "ArnLike":{  
            "AWS:SourceArn": "arn:aws:s3:::ExampleBucket"  
          }  
        }  
      }  
    ]  
  }  
}
```

有关将基于资源的策略用于 Lambda 的更多信息，请参阅[对 AWS Lambda 使用基于资源的策略 \(Lambda 函数策略\) \(p. 336\)](#)。有关使用与基于资源的策略相对的 IAM 角色（基于身份的策略）的其他信息，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](#)。

指定策略元素：操作、效果、资源和委托人

对于每个 AWS Lambda 资源（请参阅[AWS Lambda 资源和操作 \(p. 321\)](#)），该服务都定义了一组 API 操作（请参阅[Actions \(p. 371\)](#)）。为授予这些 API 操作的权限，Lambda 定义了一组您可以在策略中指定的操作。请注意，执行某项 API 操作可能需要执行多个操作的权限。在授予特定操作的权限时，您也可以标识允许或拒绝对其执行操作的资源。

以下是最基本的策略元素：

- Resource - 在策略中，您可以使用 Amazon 资源名称 (ARN) 标识策略应用到的资源。有关更多信息，请参阅 [AWS Lambda 资源和操作 \(p. 321\)](#)。
- Action – 您可以使用操作关键字标识要允许或拒绝的资源操作。例如，`lambda:InvokeFunction` 权限允许执行 AWS Lambda `Invoke` 操作的用户权限。
- Effect — 您可以指定当用户请求特定操作（可以是允许或拒绝）时的效果。如果没有显式授予（允许）对资源的访问权限，则隐式拒绝访问。您也可显式拒绝对资源的访问，这样可确保用户无法访问该资源，即使有其他策略授予了访问权限的情况下也是如此。
- Principal – 在基于身份的策略 (IAM 策略) 中，附加了策略的用户是隐式委托人。对于基于资源的策略，您可以指定要接收权限的用户、账户、服务或其他实体（仅适用于基于资源的策略）。

有关 IAM 策略语法和介绍的更多信息，请参阅 IAM 用户指南 中的 [AWS IAM 策略参考](#)。

有关显示所有 AWS Lambda API 操作及其适用的资源的表，请参阅 [Lambda API 权限：操作、资源和条件参考 \(p. 341\)](#)。

在策略中指定条件

当您授予权限时，可使用 IAM 策略语言来指定规定策略何时生效的条件。例如，您可能希望策略仅在特定日期后应用。有关使用策略语言指定条件的更多信息，请参阅 IAM 用户指南 中的 [条件](#)。

要表示条件，您可以使用预定义的条件键。没有特定于 Lambda 的条件键。但有 AWS 范围内的条件密钥，您可以根据需要使用。有关 AWS 范围内的键的完整列表，请参阅 IAM 用户指南 中的 [条件的可用键](#)。

为 AWS Lambda 使用基于身份的策略 (IAM 策略)

本主题提供了基于身份的策略的示例，在这些策略中，账户管理员可以向 IAM 身份（即：用户、组和角色）附加权限策略。

Important

我们建议您首先阅读以下介绍性主题，这些主题讲解了管理 AWS Lambda 资源访问的基本概念和选项。有关更多信息，请参阅 [管理您的 AWS Lambda 资源的访问权限概述 \(p. 321\)](#)。

本主题的各个部分涵盖以下内容：

- [使用 AWS Lambda 控制台所需的权限 \(p. 325\)](#)
- [适用于 AWS Lambda 的 AWS 托管（预定义）策略 \(p. 325\)](#)
- [客户托管策略示例 \(p. 326\)](#)

下面介绍权限策略示例。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "CreateFunctionPermissions",  
            "Effect": "Allow",  
            "Action": [  
                "lambda:CreateFunction"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "PermissionToPassAnyRole",  
            "Effect": "Allow",  
            "Action": ["lambda:InvokeFunction"],  
            "Resource": "*"  
        }  
    ]  
}
```

```
        "Effect": "Allow",
        "Action": [
            "iam:PassRole"
        ],
        "Resource": "arn:aws:iam::account-id:role/*"
    }
}
```

该策略包含两条语句：

- 第一个语句通过对 Lambda 函数使用 Amazon 资源名称 (ARN) 来授予对资源的 AWS Lambda 操作 (`lambda:CreateFunction`) 权限。目前，AWS Lambda 不支持资源级的此特定操作的权限。因此，该策略指定通配符 (*) 作为 Resource 值。
- 第二条语句授予对 IAM 角色的 IAM 操作 (`iam:PassRole`) 权限。Resource 值结尾的通配符 (*) 表示该语句允许对任何 IAM 角色的 `iam:PassRole` 操作权限。要将此权限限制到特定角色，请使用特定角色名称替换资源 ARN 中的通配符 (*)。

该策略不指定 Principal 元素，因为在基于身份的策略中，您未指定获取权限的委托人。附加了策略的用户是隐式委托人。向 IAM 角色附加权限策略后，该角色的信任策略中标识的委托人将获取权限。

有关包含所有 AWS Lambda API 操作及其适用的资源和条件的表，请参阅 [Lambda API 权限：操作、资源和条件参考 \(p. 341\)](#)。

使用 AWS Lambda 控制台所需的权限

AWS Lambda 控制台为您提供了一个创建和管理 Lambda 函数的集成环境。此控制台提供了许多功能和工作流，通常需要创建 Lambda 函数的权限以及 [Lambda API 权限：操作、资源和条件参考 \(p. 341\)](#) 中记录的特定于 API 的权限。有关这些附加控制台权限的更多信息，请参阅[使用 AWS Lambda 控制台所需的权限 \(p. 329\)](#)。

适用于 AWS Lambda 的 AWS 托管（预定义）策略

AWS 通过提供由 AWS 创建和管理的独立 IAM 策略来解决很多常用案例。托管策略可授予常用案例的必要权限，因此，您可以免去调查都需要哪些权限的工作。有关更多信息，请参阅 IAM 用户指南中的 [AWS 托管策略](#)。

下面的 AWS 托管策略可附加到您账户中的用户，这些托管策略特定于 AWS Lambda 并且按使用案例场景进行分组：

- `AWSLambdaReadOnlyAccess` - 授予对 AWS Lambda 资源的只读访问权限。请注意，此策略不授予 `lambda:InvokeFunction` 操作权限。如果您希望用户调用 Lambda 函数，也可以附加 `AWSLambdaRole` AWS 托管策略。
- `AWSLambdaFullAccess` - 授予对 AWS Lambda 资源的完全访问权限。
- `AWSLambdaRole` - 授予调用任何 Lambda 函数的权限。

Note

您可以通过登录到 IAM 控制台并在该控制台中搜索特定策略来查看这些权限策略。

此外，还有其他 AWS 托管策略，这些策略非常适合用于您在创建 Lambda 函数时指定的 IAM 角色（执行角色）。有关更多信息，请参阅 [AWS Lambda 权限模型 \(p. 339\)](#)。

此外，您还可以创建自己的自定义 IAM 策略，以授予 AWS Lambda API 操作和资源的相关权限。您可以将这些自定义策略附加到需要上述权限的 IAM 用户和组或您为 Lambda 函数创建的自定义执行角色（IAM 角色）。

客户托管策略示例

此部分中的示例提供了一组可附加到用户的示例策略。如果您是首次创建策略，建议您先在账户中创建 IAM 用户并按顺序将策略附加到用户，如此部分中的步骤所述。

在将每个策略附加到用户时，可使用控制台验证该策略的效果。最初，用户没有权限并且无法在控制台中执行任何操作。在将策略附加到用户时，可以验证用户是否能在控制台中执行各种操作。

建议您使用两个浏览器窗口：一个浏览器窗口用于创建用户和授予权限，另一个浏览器窗口用于使用用户凭证登录 AWS 管理控制台，并在向用户授予权限时验证这些权限。

有关说明如何创建可用作 Lambda 函数的执行角色的 IAM 角色的示例，请参阅 IAM 用户指南 中的[创建 IAM 角色](#)。

示例步骤

- [步骤 1：创建一个 IAM 用户 \(p. 326\)](#)
- [步骤 2：允许用户列出 Lambda 函数 \(p. 326\)](#)
- [步骤 3：允许用户查看 Lambda 函数的详细信息 \(p. 326\)](#)
- [步骤 4：允许用户调用 Lambda 函数 \(p. 327\)](#)
- [步骤 5：允许用户监控 Lambda 函数和查看 CloudWatch Logs \(p. 327\)](#)
- [步骤 6：允许用户创建 Lambda 函数 \(p. 328\)](#)

步骤 1：创建一个 IAM 用户

首先，您需要创建一个 IAM 用户，将该用户添加到具有管理权限的 IAM 组，然后向您创建的 IAM 用户授予管理权限。您随后便可以使用一个特殊的 URL 和该 IAM 用户的凭证访问 AWS。

有关说明，请参阅 IAM 用户指南 中的[创建您的第一个 IAM 用户和管理员组](#)。

步骤 2：允许用户列出 Lambda 函数

您的账户中的 IAM 用户必须先具有 `lambda>ListFunctions` 操作权限，然后才能在控制台中查看任何内容。在您授予这些权限时，控制台会显示在用户所属的特定 AWS 区域中创建的 AWS 账户中的 Lambda 函数列表。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ListExistingFunctions",  
            "Effect": "Allow",  
            "Action": [  
                "lambda>ListFunctions"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

步骤 3：允许用户查看 Lambda 函数的详细信息

用户可以选择 Lambda 函数和查看该函数的详细信息（例如，别名、版本和其他配置信息），前提是用户具有以下 AWS Lambda 操作的权限：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "DisplayFunctionDetailsPermissions",  
            "Effect": "Allow",  
            "Action": [  
                "lambda>ListVersionsByFunction",  
                "lambda>ListAliases",  
                "lambda>GetFunction",  
                "lambda>GetFunctionConfiguration",  
                "lambda>ListEventSourceMapping",  
                "lambda>GetPolicy"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

步骤 4：允许用户调用 Lambda 函数

如果要向用户授予手动调用函数的权限，您需要授予 `lambda:InvokeFunction` 操作的权限，如下所示：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "InvokePermission",  
            "Effect": "Allow",  
            "Action": [  
                "lambda:InvokeFunction"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

步骤 5：允许用户监控 Lambda 函数和查看 CloudWatch Logs

当用户调用 Lambda 函数时，AWS Lambda 将执行此操作并返回结果。用户需要额外权限才能监控 Lambda 函数。

要允许用户在控制台的 Monitoring 选项卡上或控制台主页的网格视图中查看 Lambda 函数的 CloudWatch 指标，您必须授以下权限：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "CloudWatchPermission",  
            "Effect": "Allow",  
            "Action": [  
                "cloudwatch:GetMetricStatistics"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

要使用户能够在 AWS Lambda 控制台中单击指向 CloudWatch Logs 的链接并在 CloudWatch Logs 中查看日志输出，您必须授以下权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchLogsPerms",
            "Effect": "Allow",
            "Action": [
                "cloudwatchlog:DescribeLogGroups",
                "cloudwatchlog:DescribeLogStreams",
                "cloudwatchlog:GetLogEvents"
            ],
            "Resource": "arn:aws:logs:region:account-id:log-group:/aws/lambda/*"
        }
    ]
}
```

步骤 6：允许用户创建 Lambda 函数

如果您希望用户能够创建 Lambda 函数，则您必须授以下权限。由于当用户创建 Lambda 函数时，用户需要选择 IAM 执行角色（AWS Lambda 可担任此角色来执行 Lambda 函数），因此需要与 IAM 相关的操作的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListExistingRolesAndPolicies",
            "Effect": "Allow",
            "Action": [
                "iam>ListRolePolicies",
                "iam>ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CreateFunctionPermissions",
            "Effect": "Allow",
            "Action": [
                "lambda>CreateFunction"
            ],
            "Resource": "*"
        },
        {
            "Sid": "PermissionToPassAnyRole",
            "Effect": "Allow",
            "Action": [
                "iam>PassRole"
            ],
            "Resource": "arn:aws:iam:account-id:role/*"
        }
    ]
}
```

如果您希望用户能够在创建 Lambda 函数时创建 IAM 角色，则用户需要执行 `iam:PutRolePolicy` 操作的权限，如下所示：

```
{
    "Sid": "CreateARole",
    "Effect": "Allow",
    "Action": [
        "iam>CreateRole",
        "iam>PutRolePolicy"
    ],
    "Resource": "arn:aws:iam:account-id:role/*"
}
```

```
    "iam:CreatePolicy",
    "iam:PutRolePolicy",
    "iam:AttachRolePolicy"
],
"Resource": "arn:aws:iam::account-id:role/*"
}
```

Important

每个 IAM 角色均已附加一个权限策略，该策略向此角色授予了特定权限。无论用户是创建新角色还是使用现有角色，都必须拥有与角色关联的权限策略中授予的所有操作的权限。您必须相应地向用户授予其他权限。

使用 AWS Lambda 控制台所需的权限

要利用 AWS Lambda 控制台提供的集成体验，用户通常必须具备比引用表中所述 API 特定权限更多的权限，具体取决于您希望用户能够执行的操作。有关 Lambda API 操作的更多信息，请参阅 [Lambda API 权限：操作、资源和条件参考 \(p. 341\)](#)。

例如，假定您允许您账户中的 IAM 用户具备创建 Lambda 函数来处理 Amazon S3 对象创建事件的权限。为允许该用户配置 Amazon S3 作为事件源，控制台下拉列表将显示您的存储桶列表。但是，只有在登录用户具有相关 Amazon S3 操作的权限时，控制台才能显示存储桶列表。

以下部分描述了不同集成点所需的额外权限。

如果您刚刚开始管理权限，我们建议您通过示例演练开始进行，在其中您可以创建 IAM 用户、授予用户增量许可，以及使用 AWS Lambda 控制台验证权限是否有效（请参阅 [客户托管策略示例 \(p. 326\)](#)）。

主题

- [Amazon API Gateway \(p. 329\)](#)
- [Amazon CloudWatch Events \(p. 330\)](#)
- [Amazon CloudWatch Logs \(p. 331\)](#)
- [Amazon Cognito \(p. 331\)](#)
- [Amazon DynamoDB \(p. 332\)](#)
- [Amazon Kinesis Data Streams \(p. 333\)](#)
- [Amazon S3 \(p. 334\)](#)
- [Amazon SNS \(p. 334\)](#)
- [AWS IoT \(p. 335\)](#)

Note

所有这些权限策略授予特定 AWS 服务调用 Lambda 函数的权限。配置此集成的用户必须有权调用 Lambda 函数。否则，用户无法设置配置。您可以将 `AWSLambdaRole` AWS 托管（预定义）权限策略附加到用户以提供这些权限。

Amazon API Gateway

当您在控制台中配置 API 终端节点时，控制台发出多个 API 网关 API 调用。这些调用需要 `apigateway:*` 操作的权限，如下所示：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ApiGatewayPermissions",
```

```

    "Effect": "Allow",
    "Action": [
        "apigateway:/*"
    ],
    "Resource": "*"
},
{
    "Sid": "AddPermissionToFunctionPolicy",
    "Effect": "Allow",
    "Action": [
        "lambda:AddPermission",
        "lambda:RemovePermission",
        "lambda:GetPolicy"
    ],
    "Resource": "arn:aws:lambda:region:account-id:function:/*"
},
{
    "Sid": "ListEventSourcePerm",
    "Effect": "Allow",
    "Action": [
        "lambda>ListEventSourceMappings"
    ],
    "Resource": "*"
}
]
}

```

Amazon CloudWatch Events

您可以计划什么时候调用 Lambda 函数。选择现有 CloudWatch Events 规则（或者创建新规则）之后，AWS Lambda 在调用您的 Lambda 函数的 CloudWatch 中创建新目标。要完成目标创建，您需要授予以下额外的权限：

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EventPerms",
            "Effect": "Allow",
            "Action": [
                "events:PutRule",
                "events>ListRules",
                "events>ListRuleNamesByTarget",
                "events:PutTargets",
                "events:RemoveTargets",
                "events:DescribeRule",
                "events:TestEventPattern",
                "events>ListTargetsByRule",
                "events>DeleteRule"
            ],
            "Resource": "arn:aws:events:region:account-id:/*"
        },
        {
            "Sid": "AddPermissionToFunctionPolicy",
            "Effect": "Allow",
            "Action": [
                "lambda:AddPermission",
                "lambda:RemovePermission",
                "lambda:GetPolicy"
            ],
            "Resource": "arn:aws:lambda:region:account-id:function:/*"
        }
    ]
}

```

}

Amazon CloudWatch Logs

您可以让 Amazon CloudWatch Logs 服务发布事件并调用 Lambda 函数。在您配置此服务作为事件源时，控制台列出您账户中的日志组。要完成此列表操作，您需要授予 `logs:DescribeLogGroups` 权限，如下所示：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "CloudWatchLogsPerms",  
            "Effect": "Allow",  
            "Action": [  
                "logs:FilterLogEvents",  
                "logs:DescribeLogGroups",  
                "logs:PutSubscriptionFilter",  
                "logs:DescribeSubscriptionFilters",  
                "logs>DeleteSubscriptionFilter",  
                "logs:TestMetricFilter"  
            ],  
            "Resource": "arn:aws:logs:region:account-id:/*"  
        },  
        {  
            "Sid": "AddPermissionToFunctionPolicy",  
            "Effect": "Allow",  
            "Action": [  
                "lambda:AddPermission",  
                "lambda:RemovePermission",  
                "lambda:GetPolicy"  
            ],  
            "Resource": "arn:aws:lambda:region:account-id:function:/*"  
        },  
        {  
            "Sid": "ListEventSourceMappingsPerms",  
            "Effect": "Allow",  
            "Action": [  
                "lambda>ListEventSourceMappings"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Note

管理订阅筛选器需要所显示的额外权限。

Amazon Cognito

控制台列出了您账户中的身份池。在选择某个池之后，您可以配置该池将 Cognito sync trigger 作为事件源类型。为此，您需要授以下额外的权限：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "CognitoPerms1",  
            "Effect": "Allow",  
            "Action": [  
                "cognito-identity>ListIdentityPools"  
            ]  
        }  
    ]  
}
```

```

        ],
        "Resource": [
            "arn:aws:cognito-identity:region:account-id:/*"
        ]
    },
    {
        "Sid": "CognitoPerms2",
        "Effect": "Allow",
        "Action": [
            "cognito-sync:GetCognitoEvents",
            "cognito-sync:SetCognitoEvents"
        ],
        "Resource": [
            "arn:aws:cognito-sync:region:account-id:/*"
        ]
    },
    {
        "Sid": "AddPermissionToFunctionPolicy",
        "Effect": "Allow",
        "Action": [
            "lambda:AddPermission",
            "lambda:RemovePermission",
            "lambda:GetPolicy"
        ],
        "Resource": "arn:aws:lambda:region:account-id:function:/*"
    },
    {
        "Sid": "ListEventSourcePerms",
        "Effect": "Allow",
        "Action": [
            "lambda>ListEventSourceMappings"
        ],
        "Resource": "*"
    }
]
}

```

Amazon DynamoDB

控制台列出了您账户中的所有表。在您选择某个表之后，控制台进行检查，以查看该表是否存在 DynamoDB 流。否则，它将创建流。如果您希望用户能够配置 DynamoDB 流作为 Lambda 函数的事件源，需要授以下额外的权限：

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DDBpermissions1",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeStream",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable"
            ],
            "Resource": "arn:aws:dynamodb:region:account-id:table/*"
        },
        {
            "Sid": "DDBpermissions2",
            "Effect": "Allow",
            "Action": [
                "dynamodb>ListStreams",
                "dynamodb>ListTables"
            ],
            "Resource": "*"
        }
    ]
}

```

```

},
{
    "Sid": "LambdaGetPolicyPerm",
    "Effect": "Allow",
    "Action": [
        "lambda:GetPolicy"
    ],
    "Resource": "arn:aws:lambda:region:account-id:function:*"
},
{
    "Sid": "LambdaEventSourcePerms",
    "Effect": "Allow",
    "Action": [
        "lambda>CreateEventSourceMapping",
        "lambda>DeleteEventSourceMapping",
        "lambda:GetEventSourceMapping",
        "lambda>ListEventSourceMappings",
        "lambda:UpdateEventSourceMapping"
    ],
    "Resource": "*"
}
]
}

```

Important

对于从 DynamoDB 流进行读取的 Lambda 函数，与 Lambda 函数关联的执行角色必须具有正确的权限。因此，在您向执行角色授予权限之前，用户还必须具有相同的权限。您可以通过将 AWSLambdaDynamoDBExecutionRole 预定义策略先附加到用户，然后附加到执行角色，以此来授予这些权限。

Amazon Kinesis Data Streams

控制台列出了您账户中的 Kinesis 流。选择流之后，控制台在 AWS Lambda 中创建事件源映射。要使其运行，您需要授以下额外的权限：

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PermissionForDescribeStream",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream"
            ],
            "Resource": "arn:aws:kinesis:region:account-id:stream/*"
        },
        {
            "Sid": "PermissionForListStreams",
            "Effect": "Allow",
            "Action": [
                "kinesis>ListStreams"
            ],
            "Resource": "*"
        },
        {
            "Sid": "PermissionForGetFunctionPolicy",
            "Effect": "Allow",
            "Action": [
                "lambda:GetPolicy"
            ],
            "Resource": "arn:aws:lambda:region:account-id:function:*"
        }
    ]
}

```

```
        "Sid": "LambdaEventSourcePerms",
        "Effect": "Allow",
        "Action": [
            "lambda>CreateEventSourceMapping",
            "lambda>DeleteEventSourceMapping",
            "lambda>GetEventSourceMapping",
            "lambda>ListEventSourceMappings",
            "lambda>UpdateEventSourceMapping"
        ],
        "Resource": "*"
    }
}
```

Amazon S3

控制台填充 AWS 账户中的存储桶列表，并查找各存储桶的存储桶位置。配置 Amazon S3 作为事件源后，控制台会更新存储桶通知配置。要使其运行，您需要授以下额外的权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3Permissions",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:GetBucketNotification",
                "s3:PutBucketNotification",
                "s3>ListAllMyBuckets"
            ],
            "Resource": "arn:aws:s3:::/*"
        },
        {
            "Sid": "AddPermissionToFunctionPolicy",
            "Effect": "Allow",
            "Action": [
                "lambda>AddPermission",
                "lambda>RemovePermission"
            ],
            "Resource": "arn:aws:lambda:region:account-id:function:*"
        }
    ]
}
```

Amazon SNS

控制台列出您账户中的 Amazon Simple Notification Service (Amazon SNS) 主题。在您选择主题之后，AWS Lambda 将您的 Lambda 函数订阅到该 Amazon SNS 主题。要使其运行，您需要授以下额外的权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SNSPerms",
            "Effect": "Allow",
            "Action": [
                "sns>ListSubscriptions",
                "sns>ListSubscriptionsByTopic",
                "sns>ListTopics",
                "sns>Subscribe",
                "sns>Unsubscribe"
            ]
        }
    ]
}
```

```

        ],
        "Resource": "arn:aws:sns:region:account-id:*"
    },
    {
        "Sid": "AddPermissionToFunctionPolicy",
        "Effect": "Allow",
        "Action": [
            "lambda:AddPermission",
            "lambda:RemovePermission",
            "lambda:GetPolicy"
        ],
        "Resource": "arn:aws:lambda:region:account-id:*:function:*"
    },
    {
        "Sid": "LambdaListESMappingsPerms",
        "Effect": "Allow",
        "Action": [
            "lambda>ListEventSourceMappings"
        ],
        "Resource": "*"
    }
]
}

```

AWS IoT

控制台列出所有 AWS IoT 规则。选择某个规则之后，控制台在用户界面中填充与该规则关联的剩余信息。如果您选择现有规则，控制台使用信息进行更新，以将事件发送到 AWS Lambda。您还可以创建新规则。要进行这些操作，用户必须拥有以下额外的权限：

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "IoTperms",
            "Effect": "Allow",
            "Action": [
                "iot:GetTopicRule",
                "iot>CreateTopicRule",
                "iot:ReplaceTopicRule"
            ],
            "Resource": "arn:aws:iot:region:account-id:*"
        },
        {
            "Sid": "IoTlistTopicRulePerms",
            "Effect": "Allow",
            "Action": [
                "iot>ListTopicRules"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LambdaPerms",
            "Effect": "Allow",
            "Action": [
                "lambda>AddPermission",
                "lambda:RemovePermission",
                "lambda:GetPolicy"
            ],
            "Resource": "arn:aws:lambda:region:account-id:*:function:*"
        }
    ]
}

```

对 AWS Lambda 使用基于资源的策略 (Lambda 函数策略)

Lambda 函数是 AWS Lambda 中的资源之一。您可以将权限添加到与 Lambda 函数关联的策略。附加到 Lambda 函数的权限策略称作基于资源的策略 (在 Lambda 中，称作 Lambda 函数策略)。可以使用 Lambda 函数策略来管理 Lambda 函数调用权限 (请参阅 [Invoke \(p. 425\)](#))。

Important

在创建基于资源的策略之前，建议您首先阅读以下介绍性主题，这些主题讲解了管理 AWS Lambda 资源访问的基本概念和选项。有关更多信息，请参阅 [管理您的 AWS Lambda 资源的访问权限概述 \(p. 321\)](#)。

Lambda 函数策略主要是在以下情况下使用：您在 AWS Lambda 中设置事件源以向服务或事件源授予调用 Lambda 函数的权限 (请参阅 [Invoke \(p. 425\)](#))。此情况的一个例外是在事件源 (例如 Amazon DynamoDB 或 Kinesis) 使用拉模式 (其中，权限改为在 Lambda 函数执行角色中进行管理) 的情况下。有关更多信息，请参阅 [事件源映射 \(p. 142\)](#)。

此外，可利用 Lambda 函数策略轻松授予跨账户权限来调用 Lambda 函数。假设您希望授予跨账户权限 (例如，对 Amazon S3 的权限) 来调用 Lambda 函数。您可以在 Lambda 函数策略中添加相关权限，而不是创建 IAM 角色来授予跨账户权限。

Note

如果自定义应用程序及其调用的 Lambda 函数属于同一 AWS 账户，则无需使用附加到 Lambda 函数的策略来授予显式权限。

AWS Lambda 提供以下 API 操作来管理与 Lambda 函数关联的权限策略：

- [AddPermission \(p. 373\)](#)
- [GetPolicy \(p. 422\)](#)
- [RemovePermission \(p. 455\)](#)

Note

利用 AWS Lambda 控制台在 Lambda 函数策略中管理事件源及其权限是最轻松的。如果事件源的 AWS 服务控制台支持配置事件源映射，则也可使用该控制台。在配置新的事件源或修改现有事件源时，控制台将自动修改与 Lambda 函数关联的权限策略。

使用控制台在函数的详细信息页面选择触发器选项卡，然后选择查看函数策略，可以查看函数策略。控制台不支持直接在函数策略中修改权限。您必须使用 AWS CLI 或 AWS 开发工具包。以下是本主题中之前列出的 API 操作的 AWS CLI 示例：

示例

- [示例 1：允许 Amazon S3 调用 Lambda 函数 \(p. 336\)](#)
- [示例 2：允许 Amazon API Gateway 调用 Lambda 函数 \(p. 337\)](#)
- [示例 3：允许另一个 AWS 账户创建的应用程序调用 Lambda 函数 \(跨账户方案\) \(p. 338\)](#)
- [示例 4：检索 Lambda 函数策略 \(p. 338\)](#)
- [示例 5：从 Lambda 函数策略中删除权限 \(p. 338\)](#)
- [示例 6：使用 Lambda 函数版本控制、别名和权限 \(p. 339\)](#)

示例 1：允许 Amazon S3 调用 Lambda 函数

要向 Amazon S3 授予调用 Lambda 函数的权限，您可配置如下权限：

- 将 s3.amazonaws.com 指定为 principal 值。
- 指定 lambda:InvokeFunction 作为授予其权限的 action。

要确保从特定 AWS 账户拥有的特定存储桶中生成事件，您还可以：

- 指定存储桶 ARN 作为 source-arn 值以限制特定存储桶中的事件。
- 指定拥有存储桶的 AWS 账户 ID 以确保指定的存储桶由账户所有。

以下示例 AWS CLI 命令将权限添加到 helloworld Lambda 函数策略，用来向 Amazon S3 授予调用该函数的权限。

```
aws lambda add-permission \
--region region \
--function-name helloworld \
--statement-id 1 \
--principal s3.amazonaws.com \
--action lambda:InvokeFunction \
--source-arn arn:aws:s3:::examplebucket \
--source-account 111111111111 \
--profile adminuser
```

该示例假定 adminuser（具有完全权限）正在添加此权限。因此，--profile 参数指定 adminuser 配置文件。

作为响应，AWS Lambda 返回以下 JSON 代码。Statement 值是已添加到 Lambda 函数策略的语句的 JSON 字符串版本。

```
{
  "Statement": "{\"Condition\":{\"StringEquals\":{\"AWS:SourceAccount\": \"111111111111\"},\n    \"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:s3:::examplebucket\"}},\n    \"Action\":[\"lambda:InvokeFunction\"],\n    \"Resource\":\"arn:aws:lambda:us-west-2:111111111111:function:helloworld\n\", \n    \"Effect\":\"Allow\", \"Principal\":{\"Service\":\"s3.amazonaws.com\"},\n    \"Sid\":\"1\"}"
}
```

有关推模型的信息，请参阅[事件源映射 \(p. 142\)](#)。

示例 2：允许 Amazon API Gateway 调用 Lambda 函数

要授予权限以允许 Amazon API Gateway 调用 Lambda 函数，请执行以下操作：

- 将 apigateway.amazonaws.com 指定为 principal 值。
- 指定 lambda:InvokeFunction 作为授予其权限的 action。
- 指定 API 网关 终端节点 ARN 作为 source-arn 值。

以下示例 AWS CLI 命令将权限添加到 helloworld Lambda 函数策略，用来向 API 网关 授予调用该函数的权限。

```
aws lambda add-permission \
--region region \
--function-name helloworld \
--statement-id 5 \
--principal apigateway.amazonaws.com \
```

```
--action lambda:InvokeFunction \
--source-arn arn:aws:execute-api:region:account-id:api-id/stage/method/resource-path \
--profile adminuser
```

作为响应，AWS Lambda 返回以下 JSON 代码。Statement 值是已添加到 Lambda 函数策略的语句的 JSON 字符串版本。

```
{
    "Statement": "{\"Condition\": {\"ArnLike\": {\"AWS:SourceArn\": \"arn:aws:apigateway:us-east-1::my-api-id:/test/petstorewalkthrough/pets\"}},\n        \"Action\": [\"lambda:InvokeFunction\"],\n        \"Resource\": \"arn:aws:lambda:us-west-2:account-id:function:helloworld\", \n        \"Effect\": \"Allow\", \n        \"Principal\": {\"Service\": \"apigateway.amazonaws.com\"},\n        \"Sid\": \"5\"}"
}
```

示例 3：允许另一个 AWS 账户创建的用户应用程序调用 Lambda 函数（跨账户方案）

要向另一个 AWS 账户授予权限（即，创建跨账户方案），您需要指定 AWS 账户 ID 作为 principal 值，如以下 AWS CLI 命令中所示：

```
aws lambda add-permission \
--region region \
--function-name helloworld \
--statement-id 3 \
--principal 111111111111 \
--action lambda:InvokeFunction \
--profile adminuser
```

作为响应，AWS Lambda 返回以下 JSON 代码。Statement 值是已添加到 Lambda 函数策略的语句的 JSON 字符串版本。

```
{
    "Statement": "{\"Action\": [\"lambda:InvokeFunction\"],\n        \"Resource\": \"arn:aws:lambda:us-west-2:account-id:function:helloworld\", \n        \"Effect\": \"Allow\", \n        \"Principal\": {\"AWS\": \"account-id\"},\n        \"Sid\": \"3\"}"
}
```

示例 4：检索 Lambda 函数策略

要检索 Lambda 函数策略，请使用 get-policy 命令：

```
aws lambda get-policy \
--function-name example \
--profile adminuser
```

示例 5：从 Lambda 函数策略中删除权限

要从 Lambda 函数策略中删除权限，请使用 remove-permission 命令，并指定函数名称和语句 ID：

```
aws lambda remove-permission \
```

```
--function-name example \
--statement-id 1 \
--profile adminuser
```

示例 6：使用 Lambda 函数版本控制、别名和权限

有关针对 Lambda 函数版本和别名的权限策略的更多信息，请参阅[版本控制、别名和资源策略 \(p. 276\)](#)。

AWS Lambda 权限模型

要让基于 AWS Lambda 的端到端应用程序工作，您需要管理各种权限。例如：

- 对于事件源，除了基于流的服务（Amazon Kinesis Data Streams 和 DynamoDB 流）之外，您必须向事件源授予调用您的 AWS Lambda 函数的权限。
- 对基于流的事件源（Amazon Kinesis Data Streams 和 DynamoDB 流）而言，AWS Lambda 代表您轮询流并读取流上的新记录，因此您需要为相关流操作授予 AWS Lambda 权限。
- 当您的 Lambda 函数执行之后，您可以访问账户中的 AWS 资源（例如，从 S3 存储桶读取对象）。AWS Lambda 通过代入您在创建 Lambda 函数时提供的角色，代表您执行 Lambda 函数。因此，您需要授予角色您的 Lambda 函数所需的权限，例如 Amazon S3 操作读取对象的权限。

下面几节介绍了权限管理。

主题

- [管理权限：使用 IAM 角色（执行角色）\(p. 339\)](#)
- [管理权限：使用 Lambda 函数策略 \(p. 340\)](#)
- [建议阅读材料 \(p. 341\)](#)

管理权限：使用 IAM 角色（执行角色）

每个 Lambda 函数具有与之关联的 IAM 角色（执行角色）。您将在创建 Lambda 函数时指定 IAM 角色。您授予此角色的权限确定了 AWS Lambda 在代入该角色时可以执行的操作。您可以将两种类型的权限授予 IAM 角色：

- 如果您的 Lambda 函数代码访问其他 AWS 资源，例如从 S3 存储桶读取对象或者写入日志到 CloudWatch Logs，则您需要将相关 Amazon S3 和 CloudWatch 操作的权限授予角色。
- 如果事件源基于流（Amazon Kinesis Data Streams 和 DynamoDB 流），AWS Lambda 代表您轮询这些流。AWS Lambda 需要权限轮询流和读取流上的新记录，因此您需要将相关权限授予此角色。

有关 IAM 角色的更多信息，请参阅 IAM 用户指南 中的[角色（委托和联合）](#)。

Important

创建 IAM 角色的用户实际上将权限传递到 AWS Lambda 以代入此角色，这需要用户具有 `iam:PassRole` 操作的权限。如果管理员用户创建此角色，则除了授予 `iam:PassRole` 操作权限之外，您无需执行任何操作，因为管理员用户具有完整权限，包括 `iam:PassRole` 操作。

为了简化创建执行角色的过程，AWS Lambda 提供了以下您可以使用的 AWS 托管（预定义）权限策略。这些策略包括特定情景的常用权限：

- AWSLambdaBasicExecutionRole - 仅为写入日志的 Amazon CloudWatch Logs 操作授予权限。如果您的 Lambda 函数除了写入日志之外不访问任何其他 AWS 资源，您可以使用此策略。
- AWSLambdaKinesisExecutionRole - 授予 Amazon Kinesis Data Streams 操作和 CloudWatch Logs 操作权限。如果您在编写 Lambda 函数来处理 Kinesis 流事件，您可以附加此权限策略。
- AWSLambdaDynamoDBExecutionRole - 授予 DynamoDB 流操作和 CloudWatch Logs 操作的权限。如果您在编写 Lambda 函数来处理 DynamoDB 流事件，您可以附加此权限策略。
- AWSLambdaVPCAccessExecutionRole - 授予 Amazon Elastic Compute Cloud (Amazon EC2) 操作管理弹性网络接口 (ENI) 的权限。如果您正在编写 Lambda 函数来访问 Amazon Virtual Private Cloud (Amazon VPC) 服务中 VPC 的资源，则您可以附加此权限策略。该策略还授予 CloudWatch Logs 操作写入日志的权限。

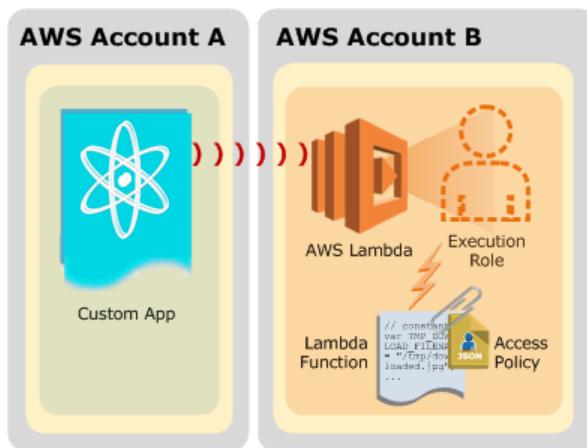
您可以在 IAM 控制台中查找这些 AWS 托管权限策略。搜索这些策略，您可以看到其中每个策略授予的权限。

管理权限：使用 Lambda 函数策略

所有支持的事件源，除了基于流的服务之外 (Kinesis 和 DynamoDB 流)，只要您授予了必需的权限就可以调用您的 Lambda 函数 (推模型)。例如，如果您希望在存储桶中创建对象时 Amazon S3 调用您的 Lambda 函数，则 Amazon S3 需要权限来调用您的 Lambda 函数。

您可以通过函数策略授予这些权限。AWS Lambda 为您提供 API 来管理函数策略中的权限。有关示例请查看 [AddPermission \(p. 373\)](#)。

您还可以使用函数策略授予跨账户权限。例如，如果用户定义的应用程序与其调用的 Lambda 函数隶属同一个 AWS 账户，则无需授予显式权限。否则，在与 Lambda 函数关联的权限策略中，拥有 Lambda 函数的 AWS 账户必须具有跨账户权限。



Note

您可以创建另一个 IAM 角色来向事件源（例如 Amazon S3 或 DynamoDB）授予调用 Lambda 函数的权限，而不是使用 Lambda 函数策略。但是，您可能会发现资源策略更易于设置，并且它们使您能够更容易地跟踪有权限调用您的 Lambda 函数的事件源。

有关 Lambda 函数策略的更多信息，请参阅对 AWS Lambda 使用基于资源的策略（Lambda 函数策略）(p. 336)。有关 Lambda 权限的更多信息，请参阅 AWS Lambda 的身份验证和访问控制 (p. 319)。

建议阅读材料

如果您是首次接触 AWS Lambda，我们建议您通读“工作原理”部分中的主题以熟悉 Lambda。下一个主题为[Lambda 执行环境和可用库 \(p. 366\)](#)。

在您阅读了“工作原理”部分中的所有主题之后，我们建议您查看[构建 Lambda 函数 \(p. 15\)](#)，尝试[入门 \(p. 3\)](#)练习，然后探讨[使用案例 \(p. 165\)](#)。每个使用情形都提供了设置端到端体验的分步说明。

Lambda API 权限：操作、资源和条件参考

在设置[访问控制 \(p. 320\)](#)和编写您可挂载到 IAM 身份的权限策略（基于身份的策略）时，可以使用下表作为参考。该列表包含每个 AWS Lambda API 操作，您可授予执行权限的对应操作，您可授予权限的 AWS 资源以及特定 API 操作的条件键。您需要在策略的 Action 字段中指定操作、在策略的 Resource 字段中指定资源值、在策略的 Condition keys 字段中指定条件键。

要指定操作，请在 API 操作名称之前使用 lambda: 前缀（例如，lambda:CreateFunction）。

Note

下表中 AWS Lambda Invoke API 的权限也可使用基于资源的策略授予。有关更多信息，请参阅[对 AWS Lambda 使用基于资源的策略（Lambda 函数策略）\(p. 336\)](#)。

您可以在 AWS Lambda 策略中使用 AWS 范围的条件键来表达条件。有关 AWS 范围内的键的完整列表，请参阅 IAM 用户指南 中的[条件的可用键](#)。

AWS Lambda 还针对一组有限的 API 操作提供预定义条件键。例如，您可以：

- 根据 Lambda 函数 ARN (Amazon 资源名称) 限制对以下操作的访问：
 - CreateEventSourceMapping
 - DeleteEventSourceMapping
 - UpdateEventSourceMapping

以下是适用此条件的示例策略：

```
"Version": "2012-10-17",
"Statement": [
    {
        "Sid": "DeleteEventSourceMappingPolicy",
        "Effect": "Allow",
        "Action": [
            "lambda:DeleteEventSourceMapping"
        ],
        "Resource": "arn:aws:lambda:region:account-id:event-source-mapping:UUID",
        "Condition": {"StringEquals": {"lambda:FunctionArn": "arn:aws:lambda:region:account-id:function:function-name"}}
    }
]
```

- 根据 AWS 服务委托人限制对以下操作的映射：
 - AddPermission
 - RemovePermission

以下是适用此条件的示例策略：

```
"Version": "2012-10-17",
"Statement": [
    {
```

```
        "Sid": "AddPermissionPolicy",
        "Effect": "Allow",
        "Action": [
            "lambda:AddPermission"
        ],
        "Resource": "arn:aws:lambda:region:account-id:function:function-name",
        "Condition": {"StringEquals": {"lambda:Principal": "s3.amazonaws.com"}}
    }
]
```

AWS Lambda API 和必需的操作权限

[AddPermission \(p. 373\)](#)

操作 : lambda:AddPermission

资源 : arn:aws:lambda:**region:account-id:**?/*

[CreateEventSourceMapping \(p. 382\)](#)

操作 : lambda>CreateEventSourceMapping

资源 : arn:aws:lambda:**region:account-id:**?

[CreateFunction \(p. 387\)](#)

操作 : lambda>CreateFunction

资源 : arn:aws:lambda:**region:account-id:**?

[DeleteEventSourceMapping \(p. 397\)](#)

操作 : lambda>DeleteEventSourceMapping

资源 : arn:aws:lambda:**region:account-id:**?

[DeleteFunction \(p. 400\)](#)

操作 : lambda>DeleteFunction

资源 : arn:aws:lambda:**region:account-id:**?

[GetEventSourceMapping \(p. 410\)](#)

操作 : lambda:GetEventSourceMapping

资源 : arn:aws:lambda:**region:account-id:**?

[GetFunction \(p. 413\)](#)

操作 : lambda:GetFunction

资源 : arn:aws:lambda:**region:account-id:**?

[GetFunctionConfiguration \(p. 417\)](#)

操作 : lambda>DescribeMountTargetSecurityGroups

资源 : arn:aws:lambda:**region:account-id:**?

[GetPolicy \(p. 422\)](#)

操作 : lambda>DescribeMountTargets

资源 : arn:aws:lambda:**region:account-id:**?

[Invoke \(p. 425\)](#)

操作 : lambda:DescribeTags

资源 : arn:aws:lambda:**region:account-id:?**

[InvokeAsync \(p. 430\)](#)

操作 : lambda:ModifyMountTargetSecurityGroups

资源 : arn:aws:lambda:**region:account-id:?**

[ListEventSourceMappings \(p. 436\)](#)

操作 : lambda>ListEventSourceMappings

资源 : arn:aws:lambda:**region:account-id:?**

[ListFunctions \(p. 439\)](#)

操作 : lambda>ListFunctions

资源 : arn:aws:lambda:**region:account-id:?**

[RemovePermission \(p. 455\)](#)

操作 : lambda:RemovePermission

资源 : arn:aws:lambda:**region:account-id:?**

[UpdateEventSourceMapping \(p. 466\)](#)

操作 : lambda:UpdateEventSourceMapping

资源 : arn:aws:lambda:**region:account-id:?**

[UpdateFunctionCode \(p. 470\)](#)

操作 : lambda:UpdateFunctionCode

资源 : arn:aws:lambda:**region:account-id:?**

[UpdateFunctionConfiguration \(p. 477\)](#)

操作 : lambda:UpdateFunctionConfiguration

资源 : arn:aws:lambda:**region:account-id:?**

策略模板

在控制台中使用某个蓝图创建 AWS Lambda 函数时，Lambda 允许您从 Lambda 策略模板列表为函数创建角色。通过选择这些模板之一，您的 Lambda 函数将自动使用已附加到该策略的必需权限创建角色。

下面列出了应用于 Policy templates 列表中的每个策略模板的权限。策略模板按照其对应的蓝图命名。Lambda 将自动使用适当的信息来填充占位符项目（例如 region 和 accountID）。有关使用策略模板创建 Lambda 函数的更多信息，请参阅[创建简单的 Lambda 函数 \(p. 8\)](#)。

将根据您创建的 Lambda 函数的类型自动应用以下模板：

基本：“基本 Lambda 权限”

{

```
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Action": "logs>CreateLogGroup",
        "Resource": "arn:aws:logs:region:accountId:*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "logs>CreateLogStream",
            "logs:PutLogEvents"
        ],
        "Resource": [
            "arn:aws:logs:region:accountId:log-group:[logGroups]]:/*"
        ]
    }
]
```

VPCAccess：“Lambda VPC 访问权限”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2>CreateNetworkInterface",
                "ec2>DeleteNetworkInterface",
                "ec2>DescribeNetworkInterfaces"
            ],
            "Resource": "*"
        }
    ]
}
```

Kinesis：“Lambda Kinesis 流轮询器权限”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:region:accountId:function:functionName*"
        },
        {
            "Effect": "Allow",
            "Action": "kinesis>ListStreams",
            "Resource": "arn:aws:kinesis:region:accountId:stream/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis>DescribeStream",
                "kinesis:GetRecords",
                "kinesis:GetShardIterator"
            ],
            "Resource": "arn:aws:kinesis:region:accountId:stream/streamName"
        }
    ]
}
```

```
    ]  
}
```

DynamoDB：“Lambda DynamoDB 流轮询器权限”

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "lambda:InvokeFunction",  
            "Resource": "arn:aws:lambda:region:accountId:function: functionName*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "dynamodb:DescribeStream",  
                "dynamodb:GetRecords",  
                "dynamodb:GetShardIterator",  
                "dynamodb>ListStreams"  
            ],  
            "Resource": "arn:aws:dynamodb:region:accountId:table(tableName)/stream/*"  
        }  
    ]  
}
```

Edge：“基本的 Edge Lambda 权限”

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs>CreateLogGroup",  
                "logs>CreateLogStream",  
                "logs:PutLogEvents"  
            ],  
            "Resource": [  
                "arn:aws:logs:*:*:*"  
            ]  
        }  
    ]  
}
```

RedrivePolicySNS：“死信队列 SNS 权限”

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sns:Publish"  
            ],  
            "Resource": "arn:aws:sns:region:accountId:topicName"  
        }  
    ]  
}
```

RedrivePolicySQS：“死信队列 SQS 权限”

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sns:SendMessage"  
            ],  
            "Resource": "arn:aws:sns:region:accountId:queueName"  
        }  
    ]  
}
```

以下模板根据您选择的蓝图选出。您还可以从下拉菜单中选择模板以添加额外权限：

CloudFormation：“CloudFormation 堆栈只读权限”

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "cloudformation:DescribeStacks"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

AMI：“AMI 只读权限”

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:DescribeImages"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

KMS：“KMS 解密权限”

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "kms:Decrypt"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

```
        "Resource": "*"
    }
}
```

S3：“S3 对象只读权限”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::/*"
        }
    ]
}
```

Elasticsearch：“Elasticsearch 权限”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "es:ESHttpPost"
            ],
            "Resource": "*"
        }
    ]
}
```

SES：“SES 退回邮件权限”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ses:SendBounce"
            ],
            "Resource": "*"
        }
    ]
}
```

TestHarness：“测试设备权限”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [

```

```
        "dynamodb:PutItem"
    ],
    "Resource": "arn:aws:dynamodb:region:accountId:table/*"
},
{
    "Effect": "Allow",
    "Action": [
        "lambda:InvokeFunction"
    ],
    "Resource": "arn:aws:lambda:region:accountId:function:*
}
]
```

Microservice：“简单微服务权限”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:DeleteItem",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:Scan",
                "dynamodb:UpdateItem"
            ],
            "Resource": "arn:aws:dynamodb:region:accountId:table/*"
        }
    ]
}
```

VPN：“VPN 连接监控权限”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeRegions",
                "ec2:DescribeVpnConnections"
            ],
            "Resource": "*"
        }
    ]
}
```

SQS：“SQS 轮询器权限”

```
{
    "Version": "2012-10-17",
```

```
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "sns:DeleteMessage",
            "sns:ReceiveMessage"
        ],
        "Resource": "arn:aws:sns:*
    },
    {
        "Effect": "Allow",
        "Action": [
            "lambda:InvokeFunction"
        ],
        "Resource": "arn:aws:lambda:region:accountId:function: functionName*"
    }
]
```

IoTButton：“AWS IoT 按钮权限”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns>ListSubscriptionsByTopic",
                "sns>CreateTopic",
                "sns>SetTopicAttributes",
                "sns>Subscribe",
                "sns>Publish"
            ],
            "Resource": "*"
        }
    ]
}
```

RekognitionNoDataAccess：“Amazon Rekognition 无数据权限”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition>CompareFaces",
                "rekognition>DetectFaces",
                "rekognition>DetectLabels"
            ],
            "Resource": "*"
        }
    ]
}
```

RekognitionReadOnlyAccess：“Amazon Rekognition 只读权限”

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "rekognition>ListCollections",
      "rekognition>ListFaces",
      "rekognition>SearchFaces",
      "rekognition>SearchFacesByImage"
    ],
    "Resource": "*"
  }
]
```

RekognitionWriteOnlyAccess：“Amazon Rekognition 只写权限”

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "rekognition>CreateCollection",
        "rekognition>IndexFaces"
      ],
      "Resource": "*"
    }
  ]
}
```

- 标记 Lambda 函数 (p. 314)
- 使用 AWS CloudTrail 记录 AWS Lambda API 调用 (p. 317)
- AWS Lambda 的身份验证和访问控制 (p. 319)
- 管理并发 (p. 350)

管理并发

AWS Lambda 的扩展单位是并发执行 (有关更多详细信息，请参阅[了解扩展行为 \(p. 147\)](#))。不过，并非所有情况下都需要无限地扩展。例如，您可能出于成本原因需要控制并发、需要调节处理批量事件所花费的时间或者只是想将它与下游资源匹配。为了帮助实现这一点，Lambda 同时提供了账户级别和函数级别的并发执行数限制控制。

账户级别并发执行数限制

默认情况下，AWS Lambda 将给定区域中所有函数的总并发执行数限制为 1000。您可以通过使用[GetAccountSettings \(p. 405\)](#) API 并查看 AccountLimit 对象来查看账户级别设置。可按如下所述方式提高此限制：

请求提高并发执行数限制

1. 打开[AWS Support Center](#) 页面，登录 (如有必要)，然后选择 Create case。
2. 对于 Regarding，选择 Service Limit Increase。

- 对于 Limit Type，选择 Lambda，在表单中填写所有必填字段，然后选择页面底部对应于首选联系方法的按钮。

函数级别并发执行数限制

默认情况下，系统会对所有函数的并发执行总数施加并发执行数限制。这种共享并发执行池称为非预留并发分配。如果您尚未设置任何函数级别的并发限制，则非预留并发限制与账户级别并发限制相同。账户级别限制一旦提高，非预留并发限制也会相应地提高。您可以通过使用 [GetAccountSettings \(p. 405\)](#) API 或 AWS Lambda 控制台来查看某个函数的非预留并发分配。对于计入共享并发执行池的函数，在使用 [GetFunctionConfiguration \(p. 417\)](#) API 进行查询时不会显示任何并发值。

您可以选择设置函数的并发执行数限制。您可能出于以下几个原因而选择这样做：

- 默认行为意味着，一个函数中并发执行数的激增会导致您使用执行数限制分离的函数受到抑制。如果为某个函数设置并发执行数限制，就将为该函数保留指定的并发执行数值。
- 函数将根据传入请求速率自动扩展，但并非架构中的所有资源都能够做到这一点。例如，关系数据库会对可处理的并发连接数加以限制。您可以为函数设置并发执行数限制，以便与其下游资源支持的值保持一致。
- 如果您的函数连接到基于 VPC 的资源，则每个并发执行都将占用所分配子网中的一个 IP。您可以为函数设置并发执行数限制，以匹配您拥有的子网大小限制。
- 如果您需要函数停止处理任何调用，则可选择将并发设置为 0 并限制所有传入执行。

通过为某个函数设置并发限制，Lambda 可确保专门对该函数应用分配，不管处理其余函数的流量多大都是如此。如果超出该限制，该函数将受到限制。该函数在受到限制时的行为取决于事件源。有关更多信息，请参阅 [限制行为 \(p. 352\)](#)。

Note

并发限制只能在函数级别上设置，不能针对单个版本设置。对给定函数的所有版本和别名的所有调用都会计入函数限制。

预留和非预留并发限制

如果您为某个函数设置并发执行数限制，则将从非预留并发池中扣除该值。例如，如果您账户的并发执行数限制为 1000，并且您具有 10 个函数，则可为一个函数指定 200 的限制，并为另一个函数指定 100 的限制。剩余的 700 将由其他 8 个函数共用。

Note

AWS Lambda 至少会为非预留并发池保留 100 的并发执行数，以便未设置特定限制的函数仍可处理请求。因此，在实践中，如果您的总账户限制为 1000，则您最多能够为单个函数分配 900。

设置每个函数的并发限制（控制台）

要使用 Lambda 控制台为 Lambda 函数设置并发限制，请执行以下操作：

- 登录 AWS 管理控制台并打开 AWS Lambda 控制台。
- 无论您是创建新的 Lambda 函数还是更新现有函数，设置并发限制的过程都是相同的。如果您是 Lambda 新手，并且不熟悉函数创建，请参阅 [创建简单的 Lambda 函数 \(p. 8\)](#)。
- 在配置选项卡下，选择并发。在预留并发中，将该值设置为希望为函数预留的最大并发执行数。请注意，在设置此值时，非预留账户并发值将自动更新，显示对账户中所有其他函数可用的剩余并发执行数。另请注意，如果您想阻止调用此函数，请将该值设置为 0。要删除此账户的专用配额值，请选择使用非预留账户并发。

设置每个函数的并发限制 (CLI)

要使用 AWS CLI 为 Lambda 函数设置并发限制，请执行以下操作：

- 使用 [PutFunctionConcurrency \(p. 453\)](#) 操作，并传入函数名和要为此函数分配的并发限制：

```
aws lambda put-function-concurrency --function-name function-name
--reserved-concurrent-executions limit value
```

要使用 AWS CLI 删除 Lambda 函数的并发限制，请执行以下操作：

- 使用 [DeleteFunctionConcurrency \(p. 403\)](#) 操作并传入函数名：

```
aws lambda delete-function-concurrency --function-name function-name
```

要使用 AWS CLI 查看 Lambda 函数的并发限制，请执行以下操作：

- 使用 [GetFunction \(p. 417\)](#) 操作并传入函数名：

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

Note

设置每函数并发会影响可用于其他函数的并发池。建议您仅将 [PutFunctionConcurrency \(p. 453\)](#) API 和 [DeleteFunctionConcurrency \(p. 403\)](#) API 的权限分配给管理用户，以便限制可进行这些更改的用户的数目。

限制行为

在达到与某个函数关联的并发限制时，任何进一步调用该函数的请求都将受到限制，即该调用不会执行您的函数。每个受限制调用都会提高该函数的 Amazon CloudWatch Throttles 指标的值。AWS Lambda 将根据受限制的调用请求的来源，使用不同的方式处理这些请求：

- 并非基于流的事件源：这些事件源中的一部分同步调用 Lambda 函数，另一些则异步调用 Lambda 函数。它们的处理方式各不相同：
- 同步调用：如果函数被同步调用并受限制，则 Lambda 会返回 429 错误，并且发出调用的服务将负责执行重试操作。`ThrottledReason` 错误代码说明您是达到了函数级别限制（如果已指定）还是账户级别限制（请参见以下注释）。每项服务可能都有自己的重试策略。例如，CloudWatch Logs 最多会重试失败的批处理五次，在重试之间存在一定的延迟。有关事件源及其调用类型的列表，请参阅[支持的事件源 \(p. 148\)](#)。

Note

如果您使用 RequestResponse 调用模式通过 AWS 开发工具包来直接调用该函数，则您的客户端将会收到 429 错误，并且您可以重试调用。

- 异步调用：如果您的 Lambda 函数被异步调用并受到限制，AWS Lambda 会自动重试受限制的事件，最长重试六个小时，并在重试之间有一定的延迟。请记住，在使用异步事件来调用 Lambda 函数之前，会将它们排队。
- 基于流的事件源：对于基于流的事件源（Kinesis 和 DynamoDB 流），AWS Lambda 会轮询您的流并调用您的 Lambda 函数。当您的 Lambda 函数受到限制时，Lambda 会一直尝试处理受限制的记录批次，

直至数据过期。对 Kinesis 来说，这个时间段可以长达七天。受限制的请求将被视为各分区的阻塞性请求，Lambda 不会从分区中读取任何新记录，直至受限制的记录批次过期或成功。如果流中有多个分区，Lambda 会继续调用未受限制的分区，直至有分区通过。

监控您的并发使用情况

为了解您的并发执行使用情况，AWS Lambda 提供以下指标：

- `ConcurrentExecutions`：此指标显示您的账户级别的并发执行数，以及任何有自定义并发限制的函数的并发执行数。
- `UnreservedConcurrentExecutions`：此指标显示已分配给默认“非预留”并发池的函数的并发执行总数。

要了解这些指标及其访问方式，请参阅[使用 Amazon CloudWatch \(p. 296\)](#)。

高级主题

以下部分提供了有关高级功能的指导信息以及有关如何构建 Lambda 应用程序的指导信息。其中包括：

- [环境变量 \(p. 354\)](#)
- [死信队列 \(p. 361\)](#)
- [使用 AWS Lambda 函数的最佳实践 \(p. 362\)](#)

环境变量

利用 Lambda 函数的环境变量，您可以将设置动态传递到函数代码和库，而无需对代码进行任何更改。环境变量是您使用 AWS Lambda 控制台、AWS Lambda CLI 或 AWS Lambda 软件开发工具包作为函数配置的一部分创建并修改的密钥值对。AWS Lambda 随后会使用相应语言所支持的标准 API（如适用于 Node.js 函数的 `process.env`）将这些密钥值对提供给您的 Lambda 函数代码。

您可以使用环境变量帮助库了解以下信息：安装文件的目录、存储输出的位置、存储连接和日志记录设置等。通过仅这些设置与应用程序逻辑分隔开，您在需要基于不同设置更改相应函数行为时将无需更新您的函数代码。

设置

假设您希望某个 Lambda 函数在经历从开发到部署的不同生命周期阶段时做出不同的行为。例如，开发、测试和生产阶段可能包含函数需要连接的数据库，而这些数据库需要不同的连接信息并且使用不同的表名称。您可以创建环境变量来引用相关数据库名称、连接信息或表名称，并根据相应函数正在执行的阶段（例如，开发、测试和生产）设置函数的值，同时您的函数代码将保持不变。

以下屏幕截图介绍如何使用 AWS 控制台修改函数的配置。第一个屏幕截图显示如何配置与测试阶段对应的函数的设置。第二个屏幕截图显示如何配置与生产阶段对应的函数的设置。

▼ Environment variables

You can define Environment Variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more.](#)

Database	Test_DB	Remove
DB_Connection	TEST	Remove
Key	Value	Remove

▼ Encryption configuration

Enable helpers for encryption in transit [Info](#)

KMS key to encrypt at rest [Info](#)

Select a KMS key to encrypt the environment variables at rest, or simply let Lambda manage the encryption.

(default) aws/lambda [▼](#) [Enter value](#)

You can define Environment Variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#).

Database	Prod_DB	Remove
DB_Connection	PROD	Remove
Key	Value	Remove

▼ Encryption configuration

Enable helpers for encryption in transit [Info](#)

KMS key to encrypt at rest [Info](#)

Select a KMS key to encrypt the environment variables at rest, or simply let Lambda manage the encryption.

(default) aws/lambda [▼](#) [Enter value](#)

请注意 Encryption configuration 部分。要了解有关如何使用此控制台的更多信息，请参阅[使用环境变量创建 Lambda 函数以存储敏感信息 \(p. 360\)](#)教程。

此外，您还可以使用 AWS CLI 创建包含环境变量的 Lambda 函数。有关更多信息，请参阅[CreateFunction \(p. 387\)](#) 和 [UpdateFunctionConfiguration \(p. 477\)](#) API。使用 AWS CloudFormation 创建和更新函数时，也支持环境变量。此外，环境变量还可用于配置特定于您的函数中包含的语言运行时或库的设置。例如，您可以修改 PATH 以指定存储可执行文件的目录。您还可以设置特定于运行时的环境变量，如 PYTHONPATH（适用于 Python）或 NODE_PATH（适用于 Node.js）。

以下示例创建了一个新的 Lambda 函数，它设置了 LD_LIBRARY_PATH 环境变量，用于指定共享库在运行时动态加载的目录。在此示例中，Lambda 函数代码使用的是 /usr/bin/test/lib64 目录中的共享库。请注意，Runtime 参数使用 nodejs6.10，但您还可以指定 nodejs4.3。

```
aws lambda create-function \
--region us-east-1
--function-name myTestFunction
--zip-file fileb://path/package.zip
--role role-arn
--environment Variables="{LD_LIBRARY_PATH=/usr/bin/test/lib64}"
--handler index.handler
--runtime nodejs6.10
--profile default
```

环境变量的命名规则

只要相应集合的总大小不超过 4KB，您可以创建的环境变量的数量就没有限制。

其他要求包括：

- 必须以字母 [a-zA-Z] 开头。
- 只能包含字母数字字符和下划线 ([a-zA-Z0-9_])。

此外，还有 AWS Lambda 预留的一组特定的密钥。如果您尝试为其中任意预留密钥设置值，将会收到提示禁止执行该操作的错误消息。有关这些密钥的更多信息，请参阅 [适用于 Lambda 函数的环境变量 \(p. 366\)](#)。

环境变量和函数版本控制

借助函数版本控制，您可以在 Lambda 函数从开发阶段到测试阶段再到生产阶段的过程中发布它的一个或多个版本，从而对您的 Lambda 函数代码进行管理。对于您发布的每个版本的 Lambda 函数，环境变量（以及 MemorySize 和 Timeout 限制等其他特定于函数的配置）会另存为相应版本的快照，且这些设置是不可变的（无法更改）。

随着应用程序和配置的要求不断变化，您可以创建新版本的 Lambda 函数并更新环境变量，以便在最新版本发布之前满足这些要求。您的函数的当前版本为 \$LATEST。

此外，您还可以创建别名，即指向特定函数版本的指针。别名的优势在于，如果您需要回滚到之前的函数版本，则可以将相应别名指向该版本，其中包含该版本需要的环境变量。有关更多信息，请参阅 [AWS Lambda 函数版本控制和别名 \(p. 264\)](#)。

环境变量加密

当您创建或更新使用环境变量的 Lambda 函数时，AWS Lambda 将使用 [AWS Key Management Service](#) 对其进行加密。当系统调用您的 Lambda 函数时，这些值已被加密且可供 Lambda 代码使用。

当您第一次在某个区域中创建或更新使用环境变量的 Lambda 函数时，系统会在 AWS KMS 中自动为您创建一个默认服务密钥。此密钥用于加密环境变量。但是，如果您希望使用加密帮助程序，并在创建 Lambda 函数后使用 KMS 加密环境变量，则您必须创建自己的 AWS KMS 键并选择它而不是默认键。默认键在选择时将给出错误。创建您自己的密钥可以实现更高的灵活性，让您可以创建、轮换、禁用和定义访问控制，并审核用于保护数据的加密密钥。有关更多信息，请参阅 [AWS Key Management Service 开发人员指南](#)。

如果您使用自己的密钥，则需要依据 [AWS Key Management Service 定价](#) 指南的规定付费。如果您使用 AWS Lambda 提供的默认的服务密钥，则无需付费。

如果您将默认 KMS 服务密钥用于 Lambda，则无需向您的函数执行角色中添加额外的 IAM 权限 – 您的角色无需更改便会自动生效。如果您提供自己的（自定义）KMS 密钥，则需要向您的执行角色中添加 kms:Decrypt。此外，要创建和更新 Lambda 函数的用户必须拥有使用 KMS 密钥的权限。有关 KMS 密钥的更多信息，请参阅 [AWS KMS 中的使用密钥策略](#)。

存储敏感信息

如上一节所述，当您部署 Lambda 函数时，您指定的所有环境变量都会在部署过程中（而不是之后）默认加密。然后当调用相应函数时，AWS Lambda 会自动解密这些环境变量。如果您需要在环境变量中存储敏感信息，我们强烈建议您先加密此类信息，然后再部署您的 Lambda 函数。

幸运的是，Lambda 控制台提供了加密帮助程序，这些程序可利用 [AWS Key Management Service](#) 将敏感信息存储为 Ciphertext，从而让您更轻松地完成这一操作。此外，Lambda 控制台还提供解密帮助程序代码，用于解密上述信息，以便您在 Lambda 函数代码中使用。有关更多信息，请参阅 [使用环境变量创建 Lambda 函数以存储敏感信息 \(p. 360\)](#)。

错误情形

如果您的函数配置超过 4KB，或您使用 AWS Lambda 预留的环境变量密钥，您的更新或创建操作将因配置错误而无法执行。在执行期间，环境变量的加密/解密可能会失败。如果 AWS Lambda 因 AWS KMS 服务异

常而无法解密环境变量，AWS KMS 将返回一条异常消息，说明错误状况以及可以采取哪些补救措施来解决此问题。这些都会记录到您在 Amazon CloudWatch 日志的函数日志流中。例如，如果系统禁用了您用于访问环境变量的 KMS 密钥，您将看到以下错误：

```
Lambda was unable to configure access to your environment variables because the KMS key used is disabled.  
Please check your KMS key settings.
```

下一步

[使用环境变量创建 Lambda 函数 \(p. 358\)](#)

使用环境变量创建 Lambda 函数

本节将详细说明如何在无需更改 Lambda 函数代码的情况下通过配置更改来修改 Lambda 函数的行为。

在本教程中，您将执行以下操作：

- 使用指定 Amazon S3 存储桶名称的环境变量的值返回的示例代码创建部署程序包。
- 调用一个 Lambda 函数并确认返回的 Amazon S3 存储桶的名称与相应环境变量设置的值一致。
- 通过更改相应环境变量指定的 Amazon S3 存储桶的名称更新 Lambda 函数。
- 再次调用 Lambda 函数并确认返回的 Amazon S3 存储桶的名称与更新后的值一致。

步骤 1：准备

确保您已完成以下步骤：

- 注册 AWS 账户并在该账户中创建管理员用户（名为 adminuser）。有关说明，请参阅 [设置 AWS 账户 \(p. 4\)](#)
- 安装并设置 AWS CLI。有关说明，请参阅 [设置 AWS Command Line Interface \(AWS CLI\) \(p. 6\)](#)

步骤 2：设置 Lambda 环境

请在此部分中执行以下操作：

- 使用提供的示例代码创建 Lambda 函数部署程序包。
- 创建 Lambda 执行角色。
- 通过上传部署程序包创建 Lambda 函数，然后手动调用该函数以便对其进行测试。

步骤 2.1：创建部署程序包

以下代码示例将读取返回 Amazon S3 存储桶名称的 Lambda 函数的环境变量。

1. 打开文本编辑器，并复制以下代码：

```
var AWS = require('aws-sdk');

exports.handler = function(event, context, callback) {

    var bucketName = process.env.S3_BUCKET;
    callback(null, bucketName);
}
```

2. 将该文件保存为 index.js。
3. 将 index.js 文件压缩为 Test_Environment_Variables.zip。

步骤 2.2：创建执行角色

创建您在创建 Lambda 函数时指定的 IAM 角色（执行角色）。

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 按照 IAM 用户指南 中 [IAM 角色](#) 的步骤操作，创建 IAM 角色（执行角色）。遵循步骤创建角色时，请注意以下事项：
 - 在 Select Role Type 中，选择 AWS Service Roles，然后选择 AWS Lambda。
 - 在 Attach Policy 中，选择名为 AWSLambdaBasicExecutionRole 的策略。
3. 写下 IAM 角色的 Amazon 资源名称 (ARN)。在下一步中创建 Lambda 函数时，您将需要此值。

步骤 2.3：创建 Lambda 函数并对其进行测试

在本节中，您将创建一个 Lambda 函数，其中包含指定名称为 Test 的 Amazon S3 存储桶的环境变量。调用此函数时，它将直接返回 Amazon S3 存储桶的名称。然后，通过将 Amazon S3 存储桶的名称更改为 Prod 更新相应配置，这样一来，再次调用此函数时，它会返回 Amazon S3 存储桶更新后的名称。

要创建 Lambda 函数，请打开一个命令提示符窗口并运行以下 Lambda AWS CLI create-function 命令。您需要提供 .zip 文件路径和执行角色 ARN。请注意，Runtime 参数使用 nodejs6.10，但您还可以指定 nodejs4.3。

```
aws lambda create-function \
--region us-east-1 \
--function-name ReturnBucketName \
--zip-file file:///file-path/Test_Environment_Variables.zip \
--role role-arn \
--environment Variables={S3_BUCKET=Test} \
--handler index.handler \
--runtime nodejs6.10 \
--version version \
--profile default
```

Note

或者，您也可以将 .zip 文件上传到同一 AWS 区域中的 Amazon S3 存储桶，然后在之前的命令中指定该存储桶和对象名称。您需要将 --zip-file 参数替换为 --code 参数。例如：

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

然后运行以下 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 \
--invocation-type Event \
--function-name ReturnBucketName \
--region us-east-1 \
--profile default \
outputfile.txt
```

Lambda 函数会将 Amazon S3 存储桶的名称返回为“Test”。

然后运行以下 Lambda CLI `update-function-configuration` 命令，通过将 Amazon S3 环境变量指向 `Prod` 存储桶对其进行更新。

```
aws lambda update-function-configuration  
--function-name ReturnBucketName \  
--region us-east-1 \  
--environment Variables={S3_BUCKET=Prod}
```

再次使用相同参数运行 `aws lambda invoke` 命令。这一次，Lambda 函数会将 Amazon S3 存储桶的名称返回为 `Prod`。

使用环境变量创建 Lambda 函数以存储敏感信息

借助 [AWS Key Management Service](#) 和 Lambda 控制台的加密帮助程序，您不仅可以为您的 Lambda 函数指定配置设置，还可以使用环境变量存储数据库密码等敏感信息。有关更多信息，请参阅 [环境变量加密 \(p. 357\)](#)。以下示例将向您展示如何完成这一操作，以及如何使用 KMS 解密敏感信息。

本教程将演示如何使用 Lambda 控制台来加密包含敏感信息的环境变量。

步骤 1：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 a Lambda function。
3. 在 Select blueprint 中，选择 Author from scratch 按钮。
4. 在 Basic information 中，执行以下操作：
 - 在文件名*中，指定您的 Lambda 函数的名称。
 - 在角色*中，选择选择现有角色。
 - 在现有角色*中，选择lambda_basic_execution。

Note

如果执行角色的策略没有 `decrypt` 权限，则您需要添加这一权限。

- 选择 Create function。

步骤 2：配置 Lambda 函数

1. 在 Configuration 下面的 Runtime 中，指定 nodejs6.10 或 nodejs4.3。
2. 在 Lambda function code 部分下面，您可以利用 Edit code inline 选项将 Lambda 函数处理程序代码替换为您的自定义代码。
3. 选择 Triggers 选项卡。在 Triggers 页面下面，(可选) 您可以通过选择 Add trigger 按钮，然后选择带有省略号 (...) 的灰色框显示可用服务列表，来自动触发您的 Lambda 函数。对于此示例，不要配置触发器，并且选择 Configuration。
4. 选择 Monitoring 选项卡。此页面将为您的 Lambda 函数调用提供即时 CloudWatch 指标，以及指向其他有用指南的链接，包括 [使用 AWS X-Ray \(p. 302\)](#)。
5. 展开 Environment variables 部分。
6. 输入您的密钥-值对。展开 Encryption configuration 部分。请注意，Lambda 在 KMS key to encrypt at rest 下面提供了默认服务密钥，可用于在您的信息上传后对其进行加密。如果您提供的信息是敏感信息，您还可以选中 Enable helpers for encryption in transit 复选框并提供自定义密钥。这一操作将掩蔽您输入的值，并且会发起对 AWS KMS 的调用以加密该值并将其返回为 Ciphertext。如果您尚未为您的账户创建 KMS 密钥，系统会为您提供指向 AWS IAM 控制台的链接，以创建密钥。您的账户必须拥有针

对该密钥的 `encrypt` 和 `decrypt` 权限。请注意，您选择加密按钮后，它将切换为解密。这样一来，您便可以更新此类信息。完成此操作后，选择加密按钮。

代码按钮提供特定于可以与您的应用程序配合使用的 Lambda 函数的运行时的示例解密代码。

Note

您不能使用默认 Lambda 服务密钥加密客户端的敏感信息。有关更多信息，请参阅 [环境变量加密 \(p. 357\)](#)。

死信队列

默认情况下，系统在丢弃事件之前将重试两次异步调用失败的 Lambda 函数。使用死信队列 (DLQ)，您可以向 Lambda 指明，应向 Amazon SQS 队列或 Amazon SNS 主题发送未处理的事件，然后再在其中采取进一步的操作。

要配置 DLQ，您可以在要向其发送事件负载的 Amazon SNS 主题或 Amazon SQS 队列的 Lambda 函数的 `DeadLetterConfig` 参数上指定目标 Amazon 资源名称 (ARN)，如以下代码所示。有关创建 Amazon SNS 主题的更多信息，请参阅[创建 SNS 主题](#)。有关创建 Amazon SQS 队列的更多信息，请参阅[教程：创建 Amazon SQS 队列](#)。

```
{  
    "Code": {  
        "ZipFile": blob,  
        "S3Bucket": "string",  
        "S3Key": "string",  
        "S3ObjectVersion": "string"  
    },  
    "Description": "string",  
    "FunctionName": "string",  
    "Handler": "string",  
    "MemorySize": number,  
    "Role": "string",  
    "Runtime": "string",  
    "Timeout": number  
    "Publish": bool,  
    "DeadLetterConfig": {  
        "TargetArn": "string"  
    }  
}
```

Lambda 将把无法处理的事件定向到您为 Lambda 函数配置的 Amazon SNS 主题或 Amazon SQS 队列。没有关联 DLQ 的函数会丢弃用尽重试次数的事件。有关重试策略的更多信息，请参阅[了解重试行为 \(p. 146\)](#)。您需要明确提供到 DLQ 资源的 `receive/delete/sendMessage` 访问权限，使其作为 Lambda 函数执行角色的部分权限。写入 DLQ 目标 ARN 的负载将成为未修改消息正文的原始事件负载。如下所述，消息的属性包含有助于您了解事件未经处理的原因信息：

名称	类型	值
RequestID	字符串	唯一的请求标识符
ErrorCode	数字	HTTP 3 位数错误代码
ErrorMessage	字符串	错误消息 (被截断为 1 KB)

如果出于某些原因，事件负载始终不能到达目标 ARN，Lambda 将增加名为 `DeadLetterErrors` 的 CloudWatch 指标的值，然后删除事件负载。



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 \(p. 362\)](#)
- [函数配置 \(p. 363\)](#)
- [警报与指标 \(p. 363\)](#)
- [流事件调用 \(p. 363\)](#)
- [Async 调用 \(p. 364\)](#)
- [Lambda VPC \(p. 364\)](#)

函数代码

- 从核心逻辑中分离 Lambda 处理程序 (入口点)。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。确保您的代码检索到的外部化配置或依赖关系在初次执行后在本地存储和引用。限制变量/对象在每次调用时的重新初始化，而是使用静态初始化/构造函数、全局/静态变量以及单例。保持运行并重复使用连接 (HTTP, 数据库等)，它们在上次调用时建立。
- 使用 [环境变量 \(p. 354\)](#) 将操作参数传递到您的函数。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：[Lambda 执行环境和可用库 \(p. 366\)](#))。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关系，建议在部署程序包中包装所有依赖关系。
- 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 内核编写的函数，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。

- 将依赖关系的复杂性降至最低。首选[执行上下文](#)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-754ddd98c7be3 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 的身份验证和访问控制 \(p. 319\)](#)。
- 熟悉[AWS Lambda 限制 \(p. 369\)](#)。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。

警报与指标

- 使用[AWS Lambda 指标 \(p. 300\)](#) 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](#) 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。

Note

如果没有足够数量的记录需要处理，较少数量的记录也可调用流处理函数，而不会等待。

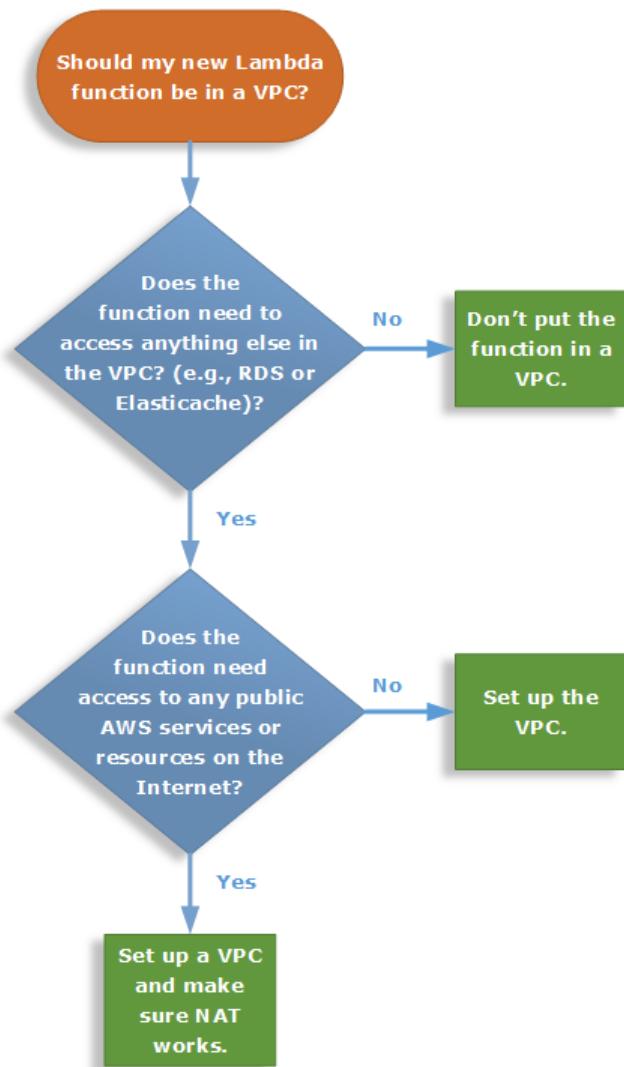
- 通过增加分片提高 Kinesis 流处理吞吐量。一个 Kinesis 流由一个或多个分片组成。Lambda 轮询每个分片时最多会使用一个并发调用。例如，如果您的流有 100 个活跃分片，则最多可以并发运行 100 个 Lambda 函数调用。增加分片数量会直接增加 Lambda 函数并发调用的最大数量，还可增加 Kinesis 流处理的吞吐量。如果您增加 Kinesis 流中的分片数量，请确保您已为数据选择了合适的分区键（请参阅[分区键](#)），这样相关记录将会位于同一分片中，而且您的数据也可合理分配。
- 在 IteratorAge 上使用[Amazon CloudWatch](#)，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 300000 (30 秒)。

Async 调用

- 创建并使用 [死信队列 \(p. 361\)](#)，解决并重放 `async` 函数错误。

Lambda VPC

- 下图指导您通过决策树了解是否应该使用 VPC (Virtual Private Cloud)：



- 除非迫不得已，否则不要将您的 Lambda 函数置于 VPC 中。使用它只是为了访问您无法公开的资源，如私有 [Amazon 关系数据库](#) 实例，除此之外没有其他好处。诸如 Amazon Elasticsearch Service 之类的服务可通过 IAM 的访问策略确保其安全，所以将终端节点公开是安全的，您不需要在 VPC 中运行函数以确保其安全。
- Lambda 在您的 VPC 中创建弹性网络接口 (ENI) 以访问内部资源。在请求增加并发前，请确保您拥有足够的 ENI 容量 (公式可在此处找到：[配置 Lambda 函数以访问 Amazon VPC 中的资源 \(p. 129\)](#)) 和 IP 地址空间。如果没有足够的 ENI 容量，需要请求增加。如果没有足够的 IP 地址空间，您可能需要创建更大的子网。
- 在您的 VPC 中创建专用 Lambda 子网：

- 这样可以更轻松地为 NAT 网关流量应用自定义路由表，而无需更改您的其他公有/私有子网。有关更多信息，请参阅 [配置 Lambda 函数以访问 Amazon VPC 中的资源 \(p. 129\)](#)。
- 这样您还可以为 Lambda 留出专门的地址空间，无需与其他资源共享。

运行时支持策略

AWS Lambda 仅弃用在维护时段结束时标记为 EOL (终止使用) 的运行时版本。标记为已弃用的版本将停止支持创建新的函数以及更新在弃用的运行时中编写的现有函数 (除非重新配置为使用支持的运行时版本)。AWS Lambda 也不会为弃用的运行时提供安全更新、技术支持或修补程序，并保留权利随时禁止调用配置为在弃用的运行时上运行的函数。您可以在下面找到标记为已弃用的运行时版本列表：

Runtime	版本
Node.js	0.10

Lambda 执行环境和可用库

底层 AWS Lambda 执行环境建立在以下各项的基础之上：

- 公用 Amazon Linux AMI 版本 (AMI 名称：amzn-ami-hvm-2017.03.1.20170812-x86_64-gp2)，可以在[此处](#)访问该版本。
有关使用 AMI 的信息，请参阅Amazon EC2 用户指南（适用于 Linux 实例）中的 [Amazon 系统映像 \(AMI\)](#)。
- Linux 内核版本 – 4.9.75-25.55.amzn1.x86_64

如果您使用代码中的任何本机二进制文件，请确保在此环境编译这些文件。注意，在 AWS Lambda 上仅支持 64 位二进制文件。

AWS Lambda 支持以下运行时版本：

- Node.js – v4.3.2 和 6.10.3
- Java - Java 8
- Python – Python 3.6 和 2.7
- .NET Core - .NET Core 1.0.1 和 .NET Core 2.0
- Go - Go 1.x

Note

并不是所有运行时都可以在公用 Amazon Linux AMI 版本或其 yum 存储库中使用。您可能需要从其各自的公共站点手动下载并安装它们。

无论您使用哪个支持的运行时，以下库在 AWS Lambda 执行环境中都可用，因此您无需包含它们：

- AWS 开发工具包 - [适用于 JavaScript 的 AWS 开发工具包](#)版本 2.190.00
- 适用于 Python 2.7 的 AWS 开发工具包 (Boto 3) 版本 3-1.5.25 botocore-1.8.39
适用于 Python 3.6 的 AWS 开发工具包 (Boto 3) 版本 3-1.5.25 botocore-1.8.39
- 适用于 Java 的 `java-1.8.0-openjdk` 的 Amazon Linux 构建。

有关在 Lambda 函数中使用 `boto` 库的示例，请参阅[从 Lambda 函数访问资源 \(p. 128\)](#)。

适用于 Lambda 函数的环境变量

下面是 AWS Lambda 执行环境中适用于 Lambda 函数的环境变量列表。下表表明由 AWS Lambda 预留、无法更改，以及可在创建 Lambda 函数时设置的环境变量。有关将环境变量用于 Lambda 函数的更多信息，请参阅[环境变量 \(p. 354\)](#)。

Lambda 环境变量

键	预留	值
LAMBDA_TASK_ROOT	是	包含到 Lambda 函数代码的路径。
AWS_EXECUTION_ENV	是	环境变量设置为以下选项之一，具体取决于 Lambda 函数的运行时：

键	预留	值
		<ul style="list-style-type: none"> AWS_Lambda_java8 AWS_Lambda_nodejs4.3 AWS_Lambda_nodejs6.10 AWS_Lambda_python2.7 AWS_Lambda_python3.6 AWS_Lambda_dotnetcore1.0 AWS_Lambda_dotnetcore2.0
LAMBDA_RUNTIME_DIR	是	仅限与 Lambda 运行时相关的项目。例如，可在此路径下找到适用于 Node.js 的 AWS 开发工具包和适用于 Python 的 boto3。
AWS_REGION	是	执行 Lambda 函数的 AWS 区域。
AWS_DEFAULT_REGION	是	执行 Lambda 函数的 AWS 区域。
AWS_LAMBDA_LOG_GROUP_NAME	是	Amazon CloudWatch Logs 组的名称，系统将在其中创建包含您的 Lambda 函数日志的日志流。
AWS_LAMBDA_LOG_STREAM_NAME	是	Amazon CloudWatch Logs 流包含 Lambda 函数日志。
AWS_LAMBDA_FUNCTION_NAME	是	Lambda 函数的名称。
AWS_LAMBDA_FUNCTION_MEMORY_SIZE	是	Lambda 函数的大小 (按 MB 计算)。
AWS_LAMBDA_FUNCTION_VERSION	是	Lambda 函数的版本。
AWS_ACCESS_KEY AWS_ACCESS_KEY_ID AWS_SECRET_KEY AWS_SECRET_ACCESS_KEY AWS_SESSION_TOKEN AWS_SECURITY_TOKEN	是	安全证书要求根据采用的运行时执行 Lambda 函数。不同的运行时将使用这些密钥不同的子集。这些子集可以通过为函数指定的 IAM 执行角色生成。
路径	否	包含 /usr/local/bin、/usr/bin 或 /bin 以运行可执行文件。
LANG	否	设置为 en_US.UTF-8。这是运行时的区域设置。
LD_LIBRARY_PATH	否	包含 /lib64、/usr/lib64、LAMBDA_TASK_ROOT 和 LAMBDA_TASK_ROOT/lib。用于存储帮助程序库和函数代码。
NODE_PATH	否	设置 Node.js 的运行时。包含 LAMBDA_RUNTIME_DIR、LAMBDA_RUNTIME_DIR/node_modules，以及 LAMBDA_TASK_ROOT。

键	预留	值
PYTHONPATH	否	设置 Python 的运行时。该运行时包含 LAMBDA_RUNTIME_DIR。
TZ	是	当前本地时间。默认为 UTC 。

AWS Lambda 限制

如 [Lambda 函数 \(p. 15\)](#) 中所述，在您打包自定义代码（包括任何依赖项）并将其上传到 AWS Lambda 中之后，您就创建了 Lambda 函数。但是，AWS Lambda 会强加一些限制，例如，您的部署程序包的大小或 Lambda 函数在每次调用中分得的内存量。本章节讨论这些 AWS Lambda 限制。

主题

- [AWS Lambda 限制 \(p. 369\)](#)
- [AWS Lambda 限制错误 \(p. 370\)](#)

AWS Lambda 限制

每个调用的 AWS Lambda 资源限制

资源	限制
内存分配范围	最小值 = 128 MB/最大值 = 3008 MB (增量为 64 MB)。如果超过最大内存使用量，则函数调用将会终止。
临时磁盘容量（“/tmp”空间）	512MB
文件描述符数	1024
过程和线程数（合并总数量）	1024
每个请求的最大执行时长	300 秒
Invoke (p. 425) 请求正文负载大小 (RequestResponse/同步调用)	6MB
Invoke (p. 425) 请求正文负载大小 (Event/异步调用)	128 K

每个区域的 AWS Lambda 账户限制

资源	默认限制
并发执行数（请参阅 管理并发 (p. 350) ）	1000

请求提高并发执行数限制

1. 打开 [AWS Support Center](#) 页面，登录（如有必要），然后单击 Create case。
2. 在 Regarding (关于) 下，选择 Service Limit Increase (提高服务限制)。
3. 在 Limit Type 下，选择 Lambda，在表单中填写所有必填字段，然后单击页面底部对应于首选联系方法的按钮。

Note

AWS 可以代您自动提高并发执行数限制，以使您的函数能够匹配传入事件率（例如由 Amazon S3 存储桶触发函数时）。

下表列出了部署 Lambda 函数的服务限制。

AWS Lambda 部署限制

项目	默认限制
Lambda 函数部署程序包大小 (压缩的 .zip/.jar 文件)	50 MB
每个区域可以上传的所有部署程序包的总大小	75GB
可压缩到部署程序包中的代码/依赖项的大小 (未压缩的 .zip/.jar 大小)。 Note 每个 Lambda 函数都会在其的 /tmp 目录中接收到额外的 500 MB 的非持久性磁盘空间。该 /tmp 目录可用于在函数初始化期间加载额外的资源，如依赖关系库或数据集。	250MB
环境变量集的总大小	4 KB

AWS Lambda 限制错误

超出前面限制表格中列出的任意限制的函数都会失败并引发 `exceeded limits` 异常。这些限制是固定的，目前无法更改。例如，如果收到 `CodeStorageExceededException` 异常或来自 AWS Lambda 的类似 "Code storage limit exceeded" 的错误消息，则需要减小代码存储的大小。

减小代码存储的大小

1. 删除不再使用的函数。
2. 减小不想删除的函数的代码大小。可以借助 AWS Lambda 控制台、AWS Command Line Interface 或 AWS 开发工具包找到 Lambda 函数的代码大小。

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。

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：

<listitem>

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。
</listitem>

- 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](#)。

主题

- [Actions \(p. 371\)](#)
- [Data Types \(p. 484\)](#)

Actions

The following actions are supported:

- [AddPermission \(p. 373\)](#)
- [CreateAlias \(p. 378\)](#)

- [CreateEventSourceMapping \(p. 382\)](#)
- [CreateFunction \(p. 387\)](#)
- [DeleteAlias \(p. 395\)](#)
- [DeleteEventSourceMapping \(p. 397\)](#)
- [DeleteFunction \(p. 400\)](#)
- [DeleteFunctionConcurrency \(p. 403\)](#)
- [GetAccountSettings \(p. 405\)](#)
- [GetAlias \(p. 407\)](#)
- [GetEventSourceMapping \(p. 410\)](#)
- [GetFunction \(p. 413\)](#)
- [GetFunctionConfiguration \(p. 417\)](#)
- [GetPolicy \(p. 422\)](#)
- [Invoke \(p. 425\)](#)
- [InvokeAsync \(p. 430\)](#)
- [ListAliases \(p. 433\)](#)
- [ListEventSourceMappings \(p. 436\)](#)
- [ListFunctions \(p. 439\)](#)
- [ListTags \(p. 442\)](#)
- [ListVersionsByFunction \(p. 444\)](#)
- [PublishVersion \(p. 447\)](#)
- [PutFunctionConcurrency \(p. 453\)](#)
- [RemovePermission \(p. 455\)](#)
- [TagResource \(p. 458\)](#)
- [UntagResource \(p. 460\)](#)
- [UpdateAlias \(p. 462\)](#)
- [UpdateEventSourceMapping \(p. 466\)](#)
- [UpdateFunctionCode \(p. 470\)](#)
- [UpdateFunctionConfiguration \(p. 477\)](#)

AddPermission

Adds a permission to the resource policy associated with the specified AWS Lambda function. You use resource policies to grant permissions to event sources that use push model. In a push model, event sources (such as Amazon S3 and custom applications) invoke your Lambda function. Each permission you add to the resource policy allows an event source, permission to invoke the Lambda function.

For information about the push model, see [Lambda Functions](#).

If you are using versioning, the permissions you add are specific to the Lambda function version or alias you specify in the `AddPermission` request via the `Qualifier` parameter. For more information about versioning, see [AWS Lambda Function Versioning and Aliases](#).

This operation requires permission for the `lambda:AddPermission` action.

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. 373)

Name of the Lambda function whose resource policy you are updating by adding a new permission.

You can specify a function name (for example, `Thumbnail`) or you can specify Amazon Resource Name (ARN) of the function (for example, `arn:aws:lambda:us-west-2:account-id:function:Thumbnail`). AWS Lambda also allows you to specify partial ARN (for example, `account-id:Thumbnail`). Note that the length constraint applies only to the 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:lambda:`)?([`a-z`]{2}-[`a-z`]+-\d{1}:)?(\d{12}:)?(`function:`)?([`a-zA-Z0-9-_`]+)(:(`\$LATEST`|[`a-zA-Z0-9-_`+]))?

[Qualifier](#) (p. 373)

You can use this optional query parameter to describe a qualified ARN using a function version or an alias name. The permission will then apply to the specific qualified ARN. For example, if you specify function version 2 as the qualifier, then permission applies only when request is made using qualified function ARN:

`arn:aws:lambda:aws-region:acct-id:function:function-name:2`

If you specify an alias name, for example `PROD`, then the permission is valid only for requests made using the alias ARN:

`arn:aws:lambda:aws-region:acct-id:function:function-name:PROD`

If the qualifier is not specified, the permission is valid only when requests are made using unqualified function ARN.

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

Action (p. 373)

The AWS Lambda action you want to allow in this statement. Each Lambda action is a string starting with `lambda:` followed by the API name (see [Actions](#)) . For example, `lambda:CreateFunction`. You can use wildcard (`lambda:*`) to grant permission for all AWS Lambda actions.

Type: String

Pattern: `(lambda:[*]|lambda:[a-zA-Z]+|[*])`

Required: Yes

EventSourceToken (p. 373)

A unique token that must be supplied by the principal invoking the function. This is currently only used for Alexa Smart Home functions.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Pattern: `[a-zA-Z0-9._\-\-]+`

Required: No

Principal (p. 373)

The principal who is getting this permission. It can be Amazon S3 service Principal (`s3.amazonaws.com`) if you want Amazon S3 to invoke the function, an AWS account ID if you are granting cross-account permission, or any valid AWS service principal such as `sns.amazonaws.com`. For example, you might want to allow a custom application in another AWS account to push events to AWS Lambda by invoking your function.

Type: String

Pattern: `.*`

Required: Yes

RevisionId (p. 373)

An optional value you can use to ensure you are updating the latest update of the function version or alias. If the `RevisionID` you pass doesn't match the latest `RevisionID` of the function or alias, it will fail with an error message, advising you to retrieve the latest function version or alias `RevisionID` using either [GetFunction \(p. 413\)](#) or [GetAlias \(p. 407\)](#).

Type: String

Required: No

[SourceAccount \(p. 373\)](#)

This parameter is used for S3 and SES. The AWS account ID (without a hyphen) of the source owner. For example, if the `SourceArn` identifies a bucket, then this is the bucket owner's account ID. You can use this additional condition to ensure the bucket you specify is owned by a specific account (it is possible the bucket owner deleted the bucket and some other AWS account created the bucket). You can also use this condition to specify all sources (that is, you don't specify the `SourceArn`) owned by a specific account.

Type: String

Pattern: \d{12}

Required: No

[SourceArn \(p. 373\)](#)

This is optional; however, when granting permission to invoke your function, you should specify this field with the Amazon Resource Name (ARN) as its value. This ensures that only events generated from the specified source can invoke the function.

Important

If you add a permission without providing the source ARN, any AWS account that creates a mapping to your function ARN can send events to invoke your Lambda function.

Type: String

Pattern: arn:aws:([a-zA-Z0-9\-_])+:([a-z]{2}-[a-z]+\-\d{1})?:(\d{12})?:(.*)

Required: No

[StatementId \(p. 373\)](#)

A unique statement identifier.

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 \(p. 375\)](#)

The permission statement you specified in the request. The response returns the same as a string using a backslash ("\\") as an escape character in the JSON.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

PolicyLengthExceededException

Lambda function access policy is limited to 20 KB.

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.

HTTP Status Code: 409

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) 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

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 V2

CreateAlias

Creates an alias that points to the specified Lambda function version. For more information, see [Introduction to AWS Lambda Aliases](#).

Alias names are unique for a given function. This requires permission for the `lambda:CreateAlias` action.

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 \(p. 378\)](#)

Name of the Lambda function for which you want to create an alias. Note that the length constraint applies only to the 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:lambda:[a-z]{2}-[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 \(p. 378\)](#)

Description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[FunctionVersion \(p. 378\)](#)

Lambda function version for which you are creating the alias.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (`\$LATEST|[0-9]+`)

Required: Yes

[Name \(p. 378\)](#)

Name for the alias you are creating.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\$)([a-zA-Z0-9-_]+)

Required: Yes

[RoutingConfig \(p. 378\)](#)

Specifies an additional version your alias can point to, allowing you to dictate what percentage of traffic will invoke each version. For more information, see [使用别名的流量转移 \(p. 280\)](#).

Type: [AliasRoutingConfiguration \(p. 490\)](#) 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 \(p. 379\)](#)

Lambda function ARN that is qualified using the alias name as the suffix. For example, if you create an alias called `BETA` that points to a `helloworld` function version, the ARN is `arn:aws:lambda:aws-regions:acct-id:function:helloworld:BETA`.

Type: String

Pattern: arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$\#LATEST|[a-zA-Z0-9-_]+))?

[Description \(p. 379\)](#)

Alias description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 379\)](#)

Function version to which the alias points.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST | [0-9]+)

[Name \(p. 379\)](#)

Alias name.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\$)([a-zA-Z0-9-_]+)

[RevisionId \(p. 379\)](#)

Represents the latest updated revision of the function or alias.

Type: String

[RoutingConfig \(p. 379\)](#)

Specifies an additional function versions the alias points to, allowing you to dictate what percentage of traffic will invoke each version. For more information, see [使用别名的流量转移 \(p. 280\)](#).

Type: [AliasRoutingConfiguration \(p. 490\)](#) object

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceConflictException

The resource already exists.

HTTP Status Code: 409

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) 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

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 V2](#)

CreateEventSourceMapping

Identifies a stream as an event source for a Lambda function. It can be either an Amazon Kinesis stream or an Amazon DynamoDB stream. AWS Lambda invokes the specified function when records are posted to the stream.

This association between a stream source and a Lambda function is called the event source mapping.

You provide mapping information (for example, which stream to read from and which Lambda function to invoke) in the request body.

Each event source, such as an Amazon Kinesis or a DynamoDB stream, can be associated with multiple AWS Lambda functions. A given Lambda function can be associated with multiple AWS event sources.

If you are using versioning, you can specify a specific function version or an alias via the function name parameter. For more information about versioning, see [AWS Lambda Function Versioning and Aliases](#).

This operation requires permission for the `lambda:CreateEventSourceMapping` action.

Request Syntax

```
POST /2015-03-31/event-source-mappings/ HTTP/1.1
Content-type: application/json

{
    "BatchSize": number,
    "Enabled": boolean,
    "EventSourceArn": "string",
    "FunctionName": "string",
    "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 \(p. 382\)](#)

The largest number of records that AWS Lambda will retrieve from your event source at the time of invoking your function. Your function receives an event with all the retrieved records. The default is 100 records.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

Required: No

[Enabled \(p. 382\)](#)

Indicates whether AWS Lambda should begin polling the event source. By default, `Enabled` is true.

Type: Boolean

Required: No

[EventSourceArn \(p. 382\)](#)

The Amazon Resource Name (ARN) of the Amazon Kinesis or the Amazon DynamoDB stream that is the event source. Any record added to this stream could cause AWS Lambda to invoke your Lambda function, it depends on the `BatchSize`. AWS Lambda POSTs the Amazon Kinesis event, containing records, to your Lambda function as JSON.

Type: String

Pattern: `arn:aws:([a-zA-Z0-9\-])+:(a-z{2}-[a-z]+\-\d{1})?:(\d{12})?:(.*)`

Required: Yes

[FunctionName \(p. 382\)](#)

The Lambda function to invoke when AWS Lambda detects an event on the stream.

You can specify the function name (for example, `Thumbnail1`) or you can specify Amazon Resource Name (ARN) of the function (for example, `arn:aws:lambda:us-west-2:account-id:function:Thumbnail`).

If you are using versioning, you can also provide a qualified function ARN (ARN that is qualified with function version or alias name as suffix). For more information about versioning, see [AWS Lambda Function Versioning and Aliases](#)

AWS Lambda also allows you to specify only the function name with the account ID qualifier (for example, `account-id:Thumbnail`).

Note that the length constraint applies only to the 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:lambda:)?([a-z]{2}-[a-z]+\-\d{1}):(?\d{12}:(?)function:(?)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Required: Yes

[StartingPosition \(p. 382\)](#)

The position in the DynamoDB or Kinesis stream where AWS Lambda should start reading. For more information, see [GetShardIterator](#) in the Amazon Kinesis API Reference Guide or [GetShardIterator](#) in the Amazon DynamoDB API Reference Guide. The `AT_TIMESTAMP` value is supported only for [Kinesis streams](#).

Type: String

Valid Values: `TRIM_HORIZON` | `LATEST` | `AT_TIMESTAMP`

Required: Yes

[StartingPositionTimestamp \(p. 382\)](#)

The timestamp of the data record from which to start reading. Used with [shard iterator type AT_TIMESTAMP](#). If a record with this exact timestamp does not exist, the iterator returned is for the next (later) record. If the timestamp is older than the current trim horizon, the iterator returned is for the oldest untrimmed data record (`TRIM_HORIZON`). Valid only for [Kinesis streams](#).

Type: Timestamp

Required: No

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
    "BatchSize": number,
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "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 \(p. 384\)](#)

The largest number of records that AWS Lambda will retrieve from your event source at the time of invoking your function. Your function receives an event with all the retrieved records.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[EventSourceArn \(p. 384\)](#)

The Amazon Resource Name (ARN) of the Amazon Kinesis stream that is the source of events.

Type: String

Pattern: `arn:aws:([a-zA-Z0-9\-])+:(\w{2}-\w{1})?:(\d{12})?:(\.**)`

[FunctionArn \(p. 384\)](#)

The Lambda function to invoke when AWS Lambda detects an event on the stream.

Type: String

Pattern: `arn:aws:lambda:\w{2}-\w{1}:\d{12}:function:[a-zA-Z0-9-_]+\:(\$\$LATEST|\w{2}-\w{1}-\w{1}-\w{1})?`

[LastModified \(p. 384\)](#)

The UTC time string indicating the last time the event mapping was updated.

Type: Timestamp

[LastProcessingResult \(p. 384\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 384\)](#)

The state of the event source mapping. It can be `Creating`, `Enabled`, `Disabled`, `Enabling`, `Disabling`, `Updating`, or `Deleting`.

Type: String

[StateTransitionReason \(p. 384\)](#)

The reason the event source mapping is in its current state. It is either user-requested or an AWS Lambda-initiated state transition.

Type: String

[UUID \(p. 384\)](#)

The AWS Lambda assigned opaque identifier for the mapping.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceConflictException

The resource already exists.

HTTP Status Code: 409

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) 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

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 V2

CreateFunction

Creates a new Lambda function. The function metadata is created from the request parameters, and the code for the function is provided by a .zip file in the request body. If the function name already exists, the operation will fail. Note that the function name is case-sensitive.

If you are using versioning, you can also publish a version of the Lambda function you are creating using the `Publish` parameter. For more information about versioning, see [AWS Lambda Function Versioning and Aliases](#).

This operation requires permission for the `lambda:CreateFunction` action.

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",
  "MemorySize": number,
  "Publish": boolean,
  "Role": "string",
  "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 \(p. 387\)](#)

The code for the Lambda function.

Type: [FunctionCode \(p. 498\)](#) object

Required: Yes

[DeadLetterConfig \(p. 387\)](#)

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic. For more information, see [死信队列 \(p. 361\)](#).

Type: [DeadLetterConfig \(p. 492\)](#) object

Required: No

[Description \(p. 387\)](#)

A short, user-defined function description. Lambda does not use this value. Assign a meaningful description as you see fit.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[Environment \(p. 387\)](#)

The parent object that contains your environment's configuration settings.

Type: [Environment \(p. 493\)](#) object

Required: No

[FunctionName \(p. 387\)](#)

The name you want to assign to the function you are uploading. The function names appear in the console and are returned in the [ListFunctions \(p. 439\)](#) API. Function names are used to specify functions to other AWS Lambda API operations, such as [Invoke \(p. 425\)](#). Note that the length constraint applies only to the 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:lambda:`)?([`a-z`]{2}-[`a-z`]+-\d{1}:)?(\d{12}:)?(function:)?([`a-zA-Z0-9-_`]+)(:(\\$\\$LATEST|[`a-zA-Z0-9-_`]+))?

Required: Yes

[Handler \(p. 387\)](#)

The function within your code that Lambda calls to begin execution. For Node.js, it is the module-name.export value in your function. For Java, it can be package.class-name:handler or package.class-name. For more information, see [Lambda Function Handler \(Java\)](#).

Type: String

Length Constraints: Maximum length of 128.

Pattern: [^\s]+

Required: Yes

[KMSKeyArn \(p. 387\)](#)

The Amazon Resource Name (ARN) of the KMS key used to encrypt your function's environment variables. If not provided, AWS Lambda will use a default service key.

Type: String

Pattern: `(arn:aws:[a-z0-9-.]+::*)|()`

Required: No

[MemorySize \(p. 387\)](#)

The amount of memory, in MB, your Lambda function is given. Lambda uses this memory size to infer the amount of CPU and memory allocated to your function. Your function use-case determines your CPU and memory requirements. For example, a database operation might need less memory compared to an image processing function. 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 \(p. 387\)](#)

This boolean parameter can be used to request AWS Lambda to create the Lambda function and publish a version as an atomic operation.

Type: Boolean

Required: No

[Role \(p. 387\)](#)

The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources. For more information, see [AWS Lambda: How it Works](#).

Type: String

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_\/]+`

Required: Yes

[Runtime \(p. 387\)](#)

The runtime environment for the Lambda function you are uploading.

To use the Python runtime v3.6, set the value to "python3.6". To use the Python runtime v2.7, set the value to "python2.7". To use the Node.js runtime v6.10, set the value to "nodejs6.10". To use the Node.js runtime v4.3, set the value to "nodejs4.3". To use the .NET Core runtime v1.0, set the value to "dotnetcore1.0". To use the .NET Core runtime v2.0, set the value to "dotnetcore2.0".

Note

Node v0.10.42 is currently marked as deprecated. You must migrate existing functions to the newer Node.js runtime versions available on AWS Lambda (nodejs4.3 or nodejs6.10) as soon as possible. Failure to do so will result in an invalid parameter error being returned. Note that you will have to follow this procedure for each region that contains functions written in the Node v0.10.42 runtime.

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | java8 | python2.7 | python3.6 | dotnetcore1.0 | dotnetcore2.0 | nodejs4.3-edge | go1.x

Required: Yes

[Tags \(p. 387\)](#)

The list of tags (key-value pairs) assigned to the new function. For more information, see [Tagging Lambda Functions](#) in the AWS Lambda Developer Guide.

Type: String to string map

Required: No

[Timeout \(p. 387\)](#)

The function execution time at which Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

[TracingConfig \(p. 387\)](#)

The parent object that contains your function's tracing settings.

Type: [TracingConfig \(p. 504\)](#) object

Required: No

[VpcConfig \(p. 387\)](#)

If your Lambda function accesses resources in a VPC, you provide this parameter identifying the list of security group IDs and subnet IDs. These must belong to the same VPC. You must provide at least one security group and one subnet ID.

Type: [VpcConfig \(p. 506\)](#) 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",
        "MasterArn": "string",
        "MemorySize": number,
        "RevisionId": "string",
        "Role": "string",
        "Runtime": "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 \(p. 390\)](#)

It is the SHA256 hash of your function deployment package.

Type: String

[CodeSize \(p. 390\)](#)

The size, in bytes, of the function .zip file you uploaded.

Type: Long

[DeadLetterConfig \(p. 390\)](#)

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic. For more information, see [死信队列 \(p. 361\)](#).

Type: [DeadLetterConfig \(p. 492\)](#) object

[Description \(p. 390\)](#)

The user-provided description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 390\)](#)

The parent object that contains your environment's configuration settings.

Type: [EnvironmentResponse \(p. 495\)](#) object

[FunctionArn \(p. 390\)](#)

The Amazon Resource Name (ARN) assigned to the function.

Type: String

Pattern: `arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:($LATEST|[a-zA-Z0-9-_]+))?`

[FunctionName \(p. 390\)](#)

The name of the function. Note that the length constraint applies only to the 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 170.

Pattern: `(arn:aws:lambda:)?([a-z]{2}-[a-z]+\d{1}:(:\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+(:($LATEST|[a-zA-Z0-9-_]+))?)`

[Handler \(p. 390\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 390\)](#)

The Amazon Resource Name (ARN) of the KMS key used to encrypt your function's environment variables. If empty, it means you are using the AWS Lambda default service key.

Type: String

Pattern: `(arn:aws:[a-zA-Z0-9-_\.]+:*)|()`

[LastModified \(p. 390\)](#)

The time stamp of the last time you updated the function. The time stamp is conveyed as a string complying with ISO-8601 in this way YYYY-MM-DDThh:mm:ssTZD (e.g., 1997-07-16T19:20:30+01:00). For more information, see [Date and Time Formats](#).

Type: String

[MasterArn \(p. 390\)](#)

Returns the ARN (Amazon Resource Name) of the master function.

Type: String

Pattern: `arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:($LATEST|[a-zA-Z0-9-_]+))?`

[MemorySize \(p. 390\)](#)

The memory size, in MB, you configured for the function. Must be a multiple of 64 MB.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 390\)](#)

Represents the latest updated revision of the function or alias.

Type: String

[Role \(p. 390\)](#)

The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources.

Type: String

Pattern: arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/\]+

[Runtime \(p. 390\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | java8 | python2.7 | python3.6 | dotnetcore1.0 | dotnetcore2.0 | nodejs4.3-edge | go1.x

[Timeout \(p. 390\)](#)

The function execution time at which Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 390\)](#)

The parent object that contains your function's tracing settings.

Type: [TracingConfigResponse \(p. 505\)](#) object

[Version \(p. 390\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[VpcConfig \(p. 390\)](#)

VPC configuration associated with your Lambda function.

Type: [VpcConfigResponse \(p. 507\)](#) object

Errors

CodeStorageExceededException

You have exceeded your maximum total code size per account. [Limits](#)

HTTP Status Code: 400

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceConflictException

The resource already exists.

HTTP Status Code: 409

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) 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

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 V2](#)

DeleteAlias

Deletes the specified Lambda function alias. For more information, see [Introduction to AWS Lambda Aliases](#).

This requires permission for the `lambda:DeleteAlias` action.

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. 395)

The Lambda function name for which the alias is created. Deleting an alias does not delete the function version to which it is pointing. Note that the length constraint applies only to the 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:lambda:`)?([`a-z`]{2}-[`a-z`]+-\d{1}:)?(\d{12}:)?(function:)?([`a-zA-Z0-9-_`]+)(:(`\$LATEST`|[`a-zA-Z0-9-_`]+))?

Name (p. 395)

Name of the alias to delete.

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. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

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 V2](#)

DeleteEventSourceMapping

Removes an event source mapping. This means AWS Lambda will no longer invoke the function for events in the associated source.

This operation requires permission for the `lambda:DeleteEventSourceMapping` action.

Request Syntax

```
DELETE /2015-03-31/event-source-mappings/UUID HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[UUID \(p. 397\)](#)

The event source mapping ID.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
    "BatchSize": number,
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "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 \(p. 397\)](#)

The largest number of records that AWS Lambda will retrieve from your event source at the time of invoking your function. Your function receives an event with all the retrieved records.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[EventSourceArn \(p. 397\)](#)

The Amazon Resource Name (ARN) of the Amazon Kinesis stream that is the source of events.

Type: String

Pattern: arn:aws:([a-zA-Z0-9\-\-]+)([a-z]{2}\-[a-z]\+\-\d{1})?:(\d{12})?:(.*)

[FunctionArn \(p. 397\)](#)

The Lambda function to invoke when AWS Lambda detects an event on the stream.

Type: String

Pattern: arn:aws:lambda:[a-z]{2}\-[a-z]\+\-\d{1}:\d{12}:function:[a-zA-Z0-9\-_]+\:(\\$_LATEST|[a-zA-Z0-9\-_]+\+)?

[LastModified \(p. 397\)](#)

The UTC time string indicating the last time the event mapping was updated.

Type: Timestamp

[LastProcessingResult \(p. 397\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 397\)](#)

The state of the event source mapping. It can be Creating, Enabled, Disabled, Enabling, Disabling, Updating, or Deleting.

Type: String

[StateTransitionReason \(p. 397\)](#)

The reason the event source mapping is in its current state. It is either user-requested or an AWS Lambda-initiated state transition.

Type: String

[UUID \(p. 397\)](#)

The AWS Lambda assigned opaque identifier for the mapping.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) 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

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 V2](#)

DeleteFunction

Deletes the specified Lambda function code and configuration.

If you are using the versioning feature and you don't specify a function version in your `DeleteFunction` request, AWS Lambda will delete the function, including all its versions, and any aliases pointing to the function versions. To delete a specific function version, you must provide the function version via the `Qualifier` parameter. For information about function versioning, see [AWS Lambda Function Versioning and Aliases](#).

When you delete a function the associated resource policy is also deleted. You will need to delete the event source mappings explicitly.

This operation requires permission for the `lambda:DeleteFunction` action.

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. 400)

The Lambda function to delete.

You can specify the function name (for example, `Thumbnail`) or you can specify Amazon Resource Name (ARN) of the function (for example, `arn:aws:lambda:us-west-2:account-id:function:Thumbnail`). If you are using versioning, you can also provide a qualified function ARN (ARN that is qualified with function version or alias name as suffix). AWS Lambda also allows you to specify only the function name with the account ID qualifier (for example, `account-id:Thumbnail`). Note that the length constraint applies only to the 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:lambda:[a-z]{2}-[a-z]+\-\d{1}:\)?(\d{12}:\)?(function:\)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Qualifier (p. 400)

Using this optional parameter you can specify a function version (but not the `\$LATEST` version) to direct AWS Lambda to delete a specific function version. If the function version has one or more aliases pointing to it, you will get an error because you cannot have aliases pointing to it. You can delete any function version but not the `\$LATEST`, that is, you cannot specify `\$LATEST` as the value of this parameter. The `\$LATEST` version can be deleted only when you want to delete all the function versions and aliases.

You can only specify a function version, not an alias name, using this parameter. You cannot delete a function version using its alias.

If you don't specify this parameter, AWS Lambda will delete the function, including all of its versions and aliases.

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. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceConflictException

The resource already exists.

HTTP Status Code: 409

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) 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

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 V2

DeleteFunctionConcurrency

Removes concurrent execution limits from this function. For more information, see [管理并发 \(p. 350\)](#).

Request Syntax

```
DELETE /2017-10-31/functions/FunctionName/concurrency HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 403\)](#)

The name of the function you are removing concurrent execution limits from. For more information, see [管理并发 \(p. 350\)](#).

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (`arn:aws:lambda:`)?([`a-z`]{2}-[`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. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) 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

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 V2](#)

GetAccountSettings

Returns a customer's account settings.

You can use this operation to retrieve Lambda limits information, such as code size and concurrency limits. For more information about limits, see [AWS Lambda Limits](#). You can also retrieve resource usage statistics, such as code storage usage and function count.

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 \(p. 405\)](#)

Provides limits of code size and concurrency associated with the current account and region.

Type: [AccountLimit \(p. 485\)](#) object

[AccountUsage \(p. 405\)](#)

Provides code size usage and function count associated with the current account and region.

Type: [AccountUsage \(p. 487\)](#) object

Errors

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

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 V2](#)

GetAlias

Returns the specified alias information such as the alias ARN, description, and function version it is pointing to. For more information, see [Introduction to AWS Lambda Aliases](#).

This requires permission for the `lambda:GetAlias` action.

Request Syntax

```
GET /2015-03-31/functions/FunctionName/aliases/Name HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 407\)](#)

Function name for which the alias is created. An alias is a subresource that exists only in the context of an existing Lambda function so you must specify the function name. Note that the length constraint applies only to the 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:lambda:[a-z]{2}-[a-z]+\d{1}:(\d{12}):)(function:[a-zA-Z0-9-_]+)(:$LATEST|[a-zA-Z0-9-_]+)`

[Name \(p. 407\)](#)

Name of the alias for which you want to retrieve information.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(?!^[\d-]+$)[a-zA-Z0-9-_]+`

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 \(p. 407\)](#)

Lambda function ARN that is qualified using the alias name as the suffix. For example, if you create an alias called **BETA** that points to a helloworld function version, the ARN is `arn:aws:lambda:aws-regions:acct-id:function:helloworld:BETA`.

Type: String

Pattern: `arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Description \(p. 407\)](#)

Alias description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 407\)](#)

Function version to which the alias points.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[Name \(p. 407\)](#)

Alias name.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(?!^[\0-9]+\$)([a-zA-Z0-9-_]+)`

[RevisionId \(p. 407\)](#)

Represents the latest updated revision of the function or alias.

Type: String

[RoutingConfig \(p. 407\)](#)

Specifies an additional function versions the alias points to, allowing you to dictate what percentage of traffic will invoke each version. For more information, see [使用别名的流量转移 \(p. 280\)](#).

Type: [AliasRoutingConfiguration \(p. 490\)](#) object

Errors

InvalidArgumentException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) 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

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 V2](#)

GetEventSourceMapping

Returns configuration information for the specified event source mapping (see [CreateEventSourceMapping \(p. 382\)](#)).

This operation requires permission for the `lambda:GetEventSourceMapping` action.

Request Syntax

```
GET /2015-03-31/event-source-mappings/UUID HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[UUID \(p. 410\)](#)

The AWS Lambda assigned ID 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,
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "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 \(p. 410\)](#)

The largest number of records that AWS Lambda will retrieve from your event source at the time of invoking your function. Your function receives an event with all the retrieved records.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[EventSourceArn \(p. 410\)](#)

The Amazon Resource Name (ARN) of the Amazon Kinesis stream that is the source of events.

Type: String

Pattern: arn:aws:([a-zA-Z0-9\-\-]+)([a-z]{2}\-[a-z]\+\-\d{1})?:(\d{12})?:(\.*)

[FunctionArn \(p. 410\)](#)

The Lambda function to invoke when AWS Lambda detects an event on the stream.

Type: String

Pattern: arn:aws:lambda:[a-z]{2}\-[a-z]\+\-\d{1}:\d{12}:function:[a-zA-Z0-9\-_]+\:(\\$_LATEST|[a-zA-Z0-9\-_]+)?

[LastModified \(p. 410\)](#)

The UTC time string indicating the last time the event mapping was updated.

Type: Timestamp

[LastProcessingResult \(p. 410\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 410\)](#)

The state of the event source mapping. It can be Creating, Enabled, Disabled, Enabling, Disabling, Updating, or Deleting.

Type: String

[StateTransitionReason \(p. 410\)](#)

The reason the event source mapping is in its current state. It is either user-requested or an AWS Lambda-initiated state transition.

Type: String

[UUID \(p. 410\)](#)

The AWS Lambda assigned opaque identifier for the mapping.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) 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

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 V2](#)

GetFunction

Returns the configuration information of the Lambda function and a presigned URL link to the .zip file you uploaded with [CreateFunction \(p. 387\)](#) so you can download the .zip file. Note that the URL is valid for up to 10 minutes. The configuration information is the same information you provided as parameters when uploading the function.

Using the optional `Qualifier` parameter, you can specify a specific function version for which you want this information. If you don't specify this parameter, the API uses unqualified function ARN which return information about the `$LATEST` version of the Lambda function. For more information, see [AWS Lambda Function Versioning and Aliases](#).

This operation requires permission for the `lambda:GetFunction` action.

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. 413)

The Lambda function name.

You can specify a function name (for example, `Thumbnail`) or you can specify Amazon Resource Name (ARN) of the function (for example, `arn:aws:lambda:us-west-2:account-id:function:Thumbnail`). AWS Lambda also allows you to specify a partial ARN (for example, `account-id:Thumbnail`). Note that the length constraint applies only to the 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:lambda:`)?([`a-z`]{2}-[`a-z`]+-\d{1}:)?(\d{12}:)?(function:)?([`a-zA-Z0-9-_`.]+)(:(`\$LATEST`|[`a-zA-Z0-9-_`+]))?

Qualifier (p. 413)

Use this optional parameter to specify a function version or an alias name. If you specify function version, the API uses qualified function ARN for the request and returns information about the specific Lambda function version. If you specify an alias name, the API uses the alias ARN and returns information about the function version to which the alias points. If you don't provide this parameter, the API uses unqualified function ARN and returns information about the `$LATEST` version of the Lambda 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",
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "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 \(p. 413\)](#)

The object for the Lambda function location.

Type: [FunctionCodeLocation \(p. 499\)](#) object

[Concurrency \(p. 413\)](#)

The concurrent execution limit set for this function. For more information, see [管理并发 \(p. 350\)](#).

Type: [Concurrency \(p. 491\)](#) object

[Configuration \(p. 413\)](#)

A complex type that describes function metadata.

Type: [FunctionConfiguration \(p. 500\)](#) object

[Tags \(p. 413\)](#)

Returns the list of tags associated with the function. For more information, see [Tagging Lambda Functions](#) in the AWS Lambda Developer Guide.

Type: String to string map

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) 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](#)

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 V2](#)

GetFunctionConfiguration

Returns the configuration information of the Lambda function. This is the same information you provided as parameters when uploading the function by using [CreateFunction \(p. 387\)](#).

If you are using the versioning feature, you can retrieve this information for a specific function version by using the optional `Qualifier` parameter and specifying the function version or alias that points to it. If you don't provide it, the API returns information about the `$LATEST` version of the function. For more information about versioning, see [AWS Lambda Function Versioning and Aliases](#).

This operation requires permission for the `lambda:GetFunctionConfiguration` operation.

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 \(p. 417\)](#)

The name of the Lambda function for which you want to retrieve the configuration information.

You can specify a function name (for example, `Thumbnail`) or you can specify Amazon Resource Name (ARN) of the function (for example, `arn:aws:lambda:us-west-2:account-id:function:Thumbnail`). AWS Lambda also allows you to specify a partial ARN (for example, `account-id:Thumbnail`). Note that the length constraint applies only to the 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:lambda:)?([a-z]{2}-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\$\text{LATEST}|[a-zA-Z0-9-_]+))?`

[Qualifier \(p. 417\)](#)

Using this optional parameter you can specify a function version or an alias name. If you specify function version, the API uses qualified function ARN and returns information about the specific function version. If you specify an alias name, the API uses the alias ARN and returns information about the function version to which the alias points.

If you don't specify this parameter, the API uses unqualified function ARN, and returns information about the `$LATEST` function version.

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",
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "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. 417)

It is the SHA256 hash of your function deployment package.

Type: String

CodeSize (p. 417)

The size, in bytes, of the function .zip file you uploaded.

Type: Long

DeadLetterConfig (p. 417)

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic. For more information, see [死信队列 \(p. 361\)](#).

Type: [DeadLetterConfig \(p. 492\)](#) object

[Description \(p. 417\)](#)

The user-provided description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 417\)](#)

The parent object that contains your environment's configuration settings.

Type: [EnvironmentResponse \(p. 495\)](#) object

[FunctionArn \(p. 417\)](#)

The Amazon Resource Name (ARN) assigned to the function.

Type: String

Pattern: `arn:aws:lambda:[a-z]{2}-[a-z]+\-\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[FunctionName \(p. 417\)](#)

The name of the function. Note that the length constraint applies only to the 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 170.

Pattern: `(arn:aws:lambda:)?([a-z]{2}-[a-z]+\-\d{1}:(?:(\d{12}:(?:function:)?)?([a-zA-Z0-9-_\.]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?)`

[Handler \(p. 417\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 417\)](#)

The Amazon Resource Name (ARN) of the KMS key used to encrypt your function's environment variables. If empty, it means you are using the AWS Lambda default service key.

Type: String

Pattern: `(arn:aws:[a-zA-Z0-9-_\.]+::.*|())`

[LastModified \(p. 417\)](#)

The time stamp of the last time you updated the function. The time stamp is conveyed as a string complying with ISO-8601 in this way YYYY-MM-DDThh:mm:ssTZD (e.g., 1997-07-16T19:20:30+01:00). For more information, see [Date and Time Formats](#).

Type: String

[MasterArn \(p. 417\)](#)

Returns the ARN (Amazon Resource Name) of the master function.

Type: String

Pattern: arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:\\$_LATEST|[a-zA-Z0-9-_]+))?

[MemorySize \(p. 417\)](#)

The memory size, in MB, you configured for the function. Must be a multiple of 64 MB.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 417\)](#)

Represents the latest updated revision of the function or alias.

Type: String

[Role \(p. 417\)](#)

The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources.

Type: String

Pattern: arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@-_/.]+

[Runtime \(p. 417\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | java8 | python2.7 | python3.6 | dotnetcore1.0 | dotnetcore2.0 | nodejs4.3-edge | go1.x

[Timeout \(p. 417\)](#)

The function execution time at which Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 417\)](#)

The parent object that contains your function's tracing settings.

Type: [TracingConfigResponse \(p. 505\)](#) object

[Version \(p. 417\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$_LATEST|[0-9]+)

[VpcConfig \(p. 417\)](#)

VPC configuration associated with your Lambda function.

Type: [VpcConfigResponse \(p. 507\)](#) object

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) 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

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 V2](#)

GetPolicy

Returns the resource policy associated with the specified Lambda function.

If you are using the versioning feature, you can get the resource policy associated with the specific Lambda function version or alias by specifying the version or alias name using the `Qualifier` parameter. For more information about versioning, see [AWS Lambda Function Versioning and Aliases](#).

You need permission for the `lambda:GetPolicy` action.

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. 422)

Function name whose resource policy you want to retrieve.

You can specify the function name (for example, `Thumbnail`) or you can specify Amazon Resource Name (ARN) of the function (for example, `arn:aws:lambda:us-west-2:account-id:function:Thumbnail`). If you are using versioning, you can also provide a qualified function ARN (ARN that is qualified with function version or alias name as suffix). AWS Lambda also allows you to specify only the function name with the account ID qualifier (for example, `account-id:Thumbnail`). Note that the length constraint applies only to the 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:lambda:`)?([`a-z`]{2}-[`a-z`]+-\d{1}:)?(\d{12}:)?(function:)?([`a-zA-Z0-9-_.`]+)(:(\\$\\$LATEST|[`a-zA-Z0-9-_`+]))?

Qualifier (p. 422)

You can specify this optional query parameter to specify a function version or an alias name in which case this API will return all permissions associated with the specific qualified ARN. If you don't provide this parameter, the API will return permissions that apply to the unqualified function ARN.

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 \(p. 422\)](#)

The resource policy associated with the specified function. The response returns the same as a string using a backslash ("\\") as an escape character in the JSON.

Type: String

[RevisionId \(p. 422\)](#)

Represents the latest updated revision of the function or alias.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) 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](#)

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 V2](#)

Invoke

Invokes a specific Lambda function. For an example, see [Create the Lambda Function and Test It Manually](#).

If you are using the versioning feature, you can invoke the specific function version by providing function version or alias name that is pointing to the function version using the `Qualifier` parameter in the request. If you don't provide the `Qualifier` parameter, the `$LATEST` version of the Lambda function is invoked. Invocations occur at least once in response to an event and functions must be idempotent to handle this. For information about the versioning feature, see [AWS Lambda Function Versioning and Aliases](#).

This operation requires permission for the `lambda:InvokeFunction` action.

Note

The `TooManyRequestsException` noted below will return the following:

`ConcurrentInvocationLimitExceeded` will be returned if you have no functions with reserved concurrency and have exceeded your account concurrent limit or if a function without reserved concurrency exceeds the account's unreserved concurrency limit.

`ReservedFunctionConcurrentInvocationLimitExceeded` will be returned when a function with reserved concurrency exceeds its configured concurrency limit.

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. 425)

Using the `ClientContext` you can pass client-specific information to the Lambda function you are invoking. You can then process the client information in your Lambda function as you choose through the `context` variable. For an example of a `ClientContext` JSON, see [PutEvents](#) in the Amazon Mobile Analytics API Reference and User Guide.

The `ClientContext` JSON must be base64-encoded and has a maximum size of 3583 bytes.

`FunctionName` (p. 425)

The Lambda function name.

You can specify a function name (for example, `Thumbnail`) or you can specify Amazon Resource Name (ARN) of the function (for example, `arn:aws:lambda:us-west-2:account-id:function:Thumbnail`). AWS Lambda also allows you to specify a partial ARN (for example, `account-id:Thumbnail`). Note that the length constraint applies only to the 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:lambda:`)?([`a-z`]{2}-[`a-z`]+\-\d{1}:)?(\d{12}:)?(`function:`)?([`a-zA-Z0-9-_`.]+)(:(`\$LATEST`|[`a-zA-Z0-9-_`+]))?

[InvocationType \(p. 425\)](#)

By default, the `Invoke` API assumes `RequestResponse` invocation type. You can optionally request asynchronous execution by specifying `Event` as the `InvocationType`. You can also use this parameter to request AWS Lambda to not execute the function but do some verification, such as if the caller is authorized to invoke the function and if the inputs are valid. You request this by specifying `DryRun` as the `InvocationType`. This is useful in a cross-account scenario when you want to verify access to a function without running it.

Valid Values: `Event` | `RequestResponse` | `DryRun`

[LogType \(p. 425\)](#)

You can set this optional parameter to `Tail` in the request only if you specify the `InvocationType` parameter with value `RequestResponse`. In this case, AWS Lambda returns the base64-encoded last 4 KB of log data produced by your Lambda function in the `x-amz-log-result` header.

Valid Values: `None` | `Tail`

[Qualifier \(p. 425\)](#)

You can use this optional parameter to specify a Lambda function version or alias name. If you specify a function version, the API uses the qualified function ARN to invoke a specific Lambda function. If you specify an alias name, the API uses the alias ARN to invoke the Lambda function version to which the alias points.

If you don't provide this parameter, then the API uses unqualified function ARN which results in invocation of the `$LATEST` version.

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 \(p. 425\)](#)

JSON that you want to provide to your Lambda function as input.

Response Syntax

```
HTTP/1.1 Status Code
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 \(p. 426\)](#)

The HTTP status code will be in the 200 range for successful request. For the `RequestResponse` invocation type this status code will be 200. For the `Event` invocation type this status code will be 202. For the `DryRun` invocation type the status code will be 204.

The response returns the following HTTP headers.

[ExecutedVersion \(p. 426\)](#)

The function version that has been executed. This value is returned only if the invocation type is RequestResponse. For more information, see [使用别名的流量转移 \(p. 280\)](#).

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (`\$LATEST` | [0-9]+)

[FunctionError \(p. 426\)](#)

Indicates whether an error occurred while executing the Lambda function. If an error occurred this field will have one of two values; Handled or Unhandled. Handled errors are errors that are reported by the function while the Unhandled errors are those detected and reported by AWS Lambda. Unhandled errors include out of memory errors and function timeouts. For information about how to report an Handled error, see [Programming Model](#).

[LogResult \(p. 426\)](#)

It is the base64-encoded logs for the Lambda function invocation. This is present only if the invocation type is RequestResponse and the logs were requested.

The response returns the following as the HTTP body.

[Payload \(p. 426\)](#)

It is the JSON representation of the object returned by the Lambda function. This is present only if the invocation type is RequestResponse.

In the event of a function error this field contains a message describing the error. For the Handled errors the Lambda function will report this message. For Unhandled errors AWS Lambda reports the message.

Errors

EC2AccessDeniedException

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 (ENI) 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. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

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 function zip file.

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

KMSInvalidStateException

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

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

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

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 V2](#)

InvokeAsync

Important

This API is deprecated. We recommend you use [Invoke API](#) (see [Invoke \(p. 425\)](#)).

Submits an invocation request to AWS Lambda. Upon receiving the request, Lambda executes the specified function asynchronously. To see the logs generated by the Lambda function execution, see the CloudWatch Logs console.

This operation requires permission for the `lambda:InvokeFunction` action.

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 \(p. 430\)](#)

The Lambda function name. Note that the length constraint applies only to the 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:lambda:`)?([`a-z`]{2}-[`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 \(p. 430\)](#)

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 \(p. 430\)](#)

It will be 202 upon success.

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

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) specified in the request does not exist.

HTTP Status Code: 404

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

Example

Invoke a Lambda function

The following example uses a POST request to invoke a Lambda function.

Sample Request

```
POST /2014-11-13/functions/helloworld/invoke-async/ HTTP/1.1
[ input json ]
```

Sample Response

```
HTTP/1.1 202 Accepted

x-amzn-requestid: f037bc5c-5a08-11e4-b02e-af446c3f9d0d
content-length: 0
connection: keep-alive
date: Wed, 22 Oct 2014 16:31:55 GMT
content-type: application/json
```

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 V2

ListAliases

Returns list of aliases created for a Lambda function. For each alias, the response includes information such as the alias ARN, description, alias name, and the function version to which it points. For more information, see [Introduction to AWS Lambda Aliases](#).

This requires permission for the `lambda>ListAliases` action.

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. 433)

Lambda function name for which the alias is created. Note that the length constraint applies only to the 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:lambda:`)?([a-z]{2}-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_+]):(:(\\$LATEST|[a-zA-Z0-9-_+]))?

[FunctionVersion](#) (p. 433)

If you specify this optional parameter, the API returns only the aliases that are pointing to the specific Lambda function version, otherwise the API returns all of the aliases created for the Lambda function.

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST|[0-9]+)

[Marker](#) (p. 433)

Optional string. An opaque pagination token returned from a previous `ListAliases` operation. If present, indicates where to continue the listing.

[MaxItems](#) (p. 433)

Optional integer. Specifies the maximum number of aliases to return in response. This parameter value must be greater than 0.

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 \(p. 433\)](#)

A list of aliases.

Type: Array of [AliasConfiguration \(p. 488\)](#) objects

[NextMarker \(p. 433\)](#)

A string, present if there are more aliases.

Type: String

Errors

`InvalidParameterValueException`

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

`ResourceNotFoundException`

The resource (for example, a Lambda function or access policy statement) 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`

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 V2](#)

ListEventSourceMappings

Returns a list of event source mappings you created using the [CreateEventSourceMapping](#) (see [CreateEventSourceMapping \(p. 382\)](#)).

For each mapping, the API returns configuration information. You can optionally specify filters to retrieve specific event source mappings.

If you are using the versioning feature, you can get list of event source mappings for a specific Lambda function version or an alias as described in the `FunctionName` parameter. For information about the versioning feature, see [AWS Lambda Function Versioning and Aliases](#).

This operation requires permission for the `lambda:ListEventSourceMappings` action.

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 \(p. 436\)](#)

The Amazon Resource Name (ARN) of the Amazon Kinesis stream. (This parameter is optional.)

Pattern: `arn:aws:([a-zA-Z0-9\-_])+:(a-z{2}-[a-z]+\-\d{1})?:(\d{12})?:(.*)`

[FunctionName \(p. 436\)](#)

The name of the Lambda function.

You can specify the function name (for example, `Thumbnail`) or you can specify Amazon Resource Name (ARN) of the function (for example, `arn:aws:lambda:us-west-2:account-id:function:Thumbnail`). If you are using versioning, you can also provide a qualified function ARN (ARN that is qualified with function version or alias name as suffix). AWS Lambda also allows you to specify only the function name with the account ID qualifier (for example, `account-id:Thumbnail`). Note that the length constraint applies only to the 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:lambda:)?([a-z]{2}-[a-z]+\-\d{1}):(?\d{12}):?(function:)?([a-zA-Z0-9\-_]+)(:(\$LATEST|[a-zA-Z0-9\-_]+))?`

[Marker \(p. 436\)](#)

Optional string. An opaque pagination token returned from a previous `ListEventSourceMappings` operation. If present, specifies to continue the list from where the returning call left off.

[MaxItems \(p. 436\)](#)

Optional integer. Specifies the maximum number of event sources to return in response. This value must be greater than 0.

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,
      "EventSourceArn": "string",
      "FunctionArn": "string",
      "LastModified": number,
      "LastProcessingResult": "string",
      "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 \(p. 437\)](#)

An array of `EventSourceMappingConfiguration` objects.

Type: Array of [EventSourceMappingConfiguration \(p. 496\)](#) objects

[NextMarker \(p. 437\)](#)

A string, present if there are more event source mappings.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) 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

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 V2](#)

ListFunctions

Returns a list of your Lambda functions. For each function, the response includes the function configuration information. You must use [GetFunction \(p. 413\)](#) to retrieve the code for your function.

This operation requires permission for the `lambda>ListFunctions` action.

If you are using the versioning feature, you can list all of your functions or only `$LATEST` versions. For information about the versioning feature, see [AWS Lambda Function Versioning and Aliases](#).

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 \(p. 439\)](#)

Optional string. If not specified, only the unqualified functions ARNs (Amazon Resource Names) will be returned.

Valid value:

`ALL`: Will return all versions, including `$LATEST` which will have fully qualified ARNs (Amazon Resource Names).

Valid Values: `ALL`

[Marker \(p. 439\)](#)

Optional string. An opaque pagination token returned from a previous `ListFunctions` operation. If present, indicates where to continue the listing.

[MasterRegion \(p. 439\)](#)

Optional string. If not specified, will return only regular function versions (i.e., non-replicated versions).

Valid values are:

The region from which the functions are replicated. For example, if you specify `us-east-1`, only functions replicated from that region will be returned.

`ALL`: Will return all functions from any region. If specified, you also must specify a valid `FunctionVersion` parameter.

Pattern: `ALL | [a-z]{2}(-gov)?-[a-z]+-\d{1}`

[MaxItems \(p. 439\)](#)

Optional integer. Specifies the maximum number of AWS Lambda functions to return in response. This parameter value must be greater than 0.

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",
      "Environment": {
        "Error": {
          "ErrorCode": "string",
          "Message": "string"
        },
        "Variables": {
          "string" : "string"
        }
      },
      "FunctionArn": "string",
      "FunctionName": "string",
      "Handler": "string",
      "KMSKeyArn": "string",
      "LastModified": "string",
      "MasterArn": "string",
      "MemorySize": number,
      "RevisionId": "string",
      "Role": "string",
      "Runtime": "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 \(p. 440\)](#)

A list of Lambda functions.

Type: Array of [FunctionConfiguration \(p. 500\)](#) objects

[NextMarker \(p. 440\)](#)

A string, present if there are more functions.

Type: String

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ServiceException

The AWS Lambda service encountered an internal error.

HTTP Status Code: 500

TooManyRequestsException

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 V2](#)

ListTags

Returns a list of tags assigned to a function when supplied the function ARN (Amazon Resource Name). For more information on Tagging, see [Tagging Lambda Functions](#) in the AWS Lambda Developer Guide.

Request Syntax

```
GET /2017-03-31/tags/ARN HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[Resource \(p. 442\)](#)

The ARN (Amazon Resource Name) of the function. For more information, see [Tagging Lambda Functions](#) in the AWS Lambda Developer Guide.

Pattern: arn:aws:lambda:[a-z]{2}-[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 \(p. 442\)](#)

The list of tags assigned to the function. For more information, see [Tagging Lambda Functions](#) in the AWS Lambda Developer Guide.

Type: String to string map

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) 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

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 V2](#)

ListVersionsByFunction

List all versions of a function. For information about the versioning feature, see [AWS Lambda Function Versioning and Aliases](#).

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. 444)

Function name whose versions to list. You can specify a function name (for example, `Thumbnail`) or you can specify Amazon Resource Name (ARN) of the function (for example, `arn:aws:lambda:us-west-2:account-id:function:Thumbnail`). AWS Lambda also allows you to specify a partial ARN (for example, `account-id:Thumbnail`). Note that the length constraint applies only to the 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:lambda:`)?([`a-z`]{2}-[`a-z`]+-\d{1}:)?(\d{12}:)?(function:)?([`a-zA-Z0-9-_`.]+)(:(\\$LATEST|[`a-zA-Z0-9-_`+]))?

[Marker](#) (p. 444)

Optional string. An opaque pagination token returned from a previous `ListVersionsByFunction` operation. If present, indicates where to continue the listing.

[MaxItems](#) (p. 444)

Optional integer. Specifies the maximum number of AWS Lambda function versions to return in response. This parameter value must be greater than 0.

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",
"MasterArn": "string",
"MemorySize": number,
"RevisionId": "string",
"Role": "string",
"Runtime": "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 \(p. 444\)](#)

A string, present if there are more function versions.

Type: String

[Versions \(p. 444\)](#)

A list of Lambda function versions.

Type: Array of [FunctionConfiguration \(p. 500\)](#) objects

Errors

InvalidArgumentException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) 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

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 V2](#)

PublishVersion

Publishes a version of your function from the current snapshot of \$LATEST. That is, AWS Lambda takes a snapshot of the function code and configuration information from \$LATEST and publishes a new version. The code and configuration cannot be modified after publication. For information about the versioning feature, see [AWS Lambda Function Versioning and Aliases](#).

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 \(p. 447\)](#)

The Lambda function name. You can specify a function name (for example, `Thumbnail`) or you can specify Amazon Resource Name (ARN) of the function (for example, `arn:aws:lambda:us-west-2:account-id:function:ThumbNail`). AWS Lambda also allows you to specify a partial ARN (for example, `account-id:Thumbnail`). Note that the length constraint applies only to the 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:lambda:`)?([`a-z`]{2}-[`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 \(p. 447\)](#)

The SHA256 hash of the deployment package you want to publish. This provides validation on the code you are publishing. If you provide this parameter, the value must match the SHA256 of the \$LATEST version for the publication to succeed. You can use the DryRun parameter of [UpdateFunctionCode \(p. 470\)](#) to verify the hash value that will be returned before publishing your new version.

Type: String

Required: No

[Description \(p. 447\)](#)

The description for the version you are publishing. If not provided, AWS Lambda copies the description from the \$LATEST version.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[RevisionId \(p. 447\)](#)

An optional value you can use to ensure you are updating the latest update of the function version or alias. If the RevisionID you pass doesn't match the latest RevisionID of the function or alias, it will fail with an error message, advising you to retrieve the latest function version or alias RevisionID using either [GetFunction \(p. 413\)](#) or [GetAlias \(p. 407\)](#).

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",
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "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 \(p. 448\)](#)

It is the SHA256 hash of your function deployment package.

Type: String

[CodeSize \(p. 448\)](#)

The size, in bytes, of the function .zip file you uploaded.

Type: Long

[DeadLetterConfig \(p. 448\)](#)

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic. For more information, see [死信队列 \(p. 361\)](#).

Type: [DeadLetterConfig \(p. 492\)](#) object

[Description \(p. 448\)](#)

The user-provided description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 448\)](#)

The parent object that contains your environment's configuration settings.

Type: [EnvironmentResponse \(p. 495\)](#) object

[FunctionArn \(p. 448\)](#)

The Amazon Resource Name (ARN) assigned to the function.

Type: String

Pattern: `arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[FunctionName \(p. 448\)](#)

The name of the function. Note that the length constraint applies only to the 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 170.

Pattern: `(arn:aws:lambda:)?([a-z]{2}-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Handler \(p. 448\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 448\)](#)

The Amazon Resource Name (ARN) of the KMS key used to encrypt your function's environment variables. If empty, it means you are using the AWS Lambda default service key.

Type: String

Pattern: `(arn:aws:[a-z0-9-.]+::*)|()`

[LastModified \(p. 448\)](#)

The time stamp of the last time you updated the function. The time stamp is conveyed as a string complying with ISO-8601 in this way YYYY-MM-DDThh:mm:ssTZD (e.g., 1997-07-16T19:20:30+01:00). For more information, see [Date and Time Formats](#).

Type: String

[MasterArn \(p. 448\)](#)

Returns the ARN (Amazon Resource Name) of the master function.

Type: String

Pattern: `arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[MemorySize \(p. 448\)](#)

The memory size, in MB, you configured for the function. Must be a multiple of 64 MB.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 448\)](#)

Represents the latest updated revision of the function or alias.

Type: String

[Role \(p. 448\)](#)

The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources.

Type: String

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-\/_]+`

[Runtime \(p. 448\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | java8 | python2.7 | python3.6 | dotnetcore1.0 | dotnetcore2.0 | nodejs4.3-edge | go1.x

[Timeout \(p. 448\)](#)

The function execution time at which Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 448\)](#)

The parent object that contains your function's tracing settings.

Type: [TracingConfigResponse \(p. 505\)](#) object

[Version \(p. 448\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST | [0-9]+)

[VpcConfig \(p. 448\)](#)

VPC configuration associated with your Lambda function.

Type: [VpcConfigResponse \(p. 507\)](#) object

Errors

[CodeStorageExceededException](#)

You have exceeded your maximum total code size per account. [Limits](#)

HTTP Status Code: 400

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

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 (for example, a Lambda function or access policy statement) 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](#)

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 V2](#)

PutFunctionConcurrency

Sets a limit on the number of concurrent executions available to this function. It is a subset of your account's total concurrent execution limit per region. Note that Lambda automatically reserves a buffer of 100 concurrent executions for functions without any reserved concurrency limit. This means if your account limit is 1000, you have a total of 900 available to allocate to individual functions. For more information, see [管理并发 \(p. 350\)](#).

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 \(p. 453\)](#)

The name of the function you are setting concurrent execution limits on. For more information, see [管理并发 \(p. 350\)](#).

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (`arn:aws:lambda:`)?([a-z]{2}-[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 \(p. 453\)](#)

The concurrent execution limit reserved for this function. For more information, see [管理并发 \(p. 350\)](#).

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 \(p. 453\)](#)

The number of concurrent executions reserved for this function. For more information, see [管理并发 \(p. 350\)](#).

Type: Integer

Valid Range: Minimum value of 0.

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) 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](#)

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 V2](#)

RemovePermission

You can remove individual permissions from an resource policy associated with a Lambda function by providing a statement ID that you provided when you added the permission.

If you are using versioning, the permissions you remove are specific to the Lambda function version or alias you specify in the `AddPermission` request via the `Qualifier` parameter. For more information about versioning, see [AWS Lambda Function Versioning and Aliases](#).

Note that removal of a permission will cause an active event source to lose permission to the function.

You need permission for the `lambda:RemovePermission` action.

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. 455)

Lambda function whose resource policy you want to remove a permission from.

You can specify a function name (for example, `Thumbnail`) or you can specify Amazon Resource Name (ARN) of the function (for example, `arn:aws:lambda:us-west-2:account-id:function:Thumbnail`). AWS Lambda also allows you to specify a partial ARN (for example, `account-id:Thumbnail`). Note that the length constraint applies only to the 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:lambda:`)?([a-zA-Z]{2}-[a-zA-Z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_+]):(:(\$LATEST|[a-zA-Z0-9-_+]))?

`Qualifier` (p. 455)

You can specify this optional parameter to remove permission associated with a specific function version or function alias. If you don't specify this parameter, the API removes permission associated with the unqualified function ARN.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: ([a-zA-Z0-9\$-_]+)

`RevisionId` (p. 455)

An optional value you can use to ensure you are updating the latest update of the function version or alias. If the `RevisionID` you pass doesn't match the latest `RevisionID` of the function or alias, it will fail with an error message, advising you to retrieve the latest function version or alias `RevisionID` using either [GetFunction](#) (p. 413) or [GetAlias](#) (p. 407).

`StatementId` (p. 455)

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. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

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 (for example, a Lambda function or access policy statement) 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

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 V2

TagResource

Creates a list of tags (key-value pairs) on the Lambda function. Requires the Lambda function ARN (Amazon Resource Name). If a key is specified without a value, Lambda creates a tag with the specified key and a value of null. For more information, see [Tagging Lambda Functions](#) in the AWS Lambda Developer Guide.

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.

[Resource \(p. 458\)](#)

The ARN (Amazon Resource Name) of the Lambda function. For more information, see [Tagging Lambda Functions](#) in the AWS Lambda Developer Guide.

Pattern: `arn:aws:lambda:[a-z]{2}-[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 \(p. 458\)](#)

The list of tags (key-value pairs) you are assigning to the Lambda function. For more information, see [Tagging Lambda Functions](#) in the AWS Lambda Developer Guide.

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. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) 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

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 V2](#)

UntagResource

Removes tags from a Lambda function. Requires the function ARN (Amazon Resource Name). For more information, see [Tagging Lambda Functions](#) in the AWS Lambda Developer Guide.

Request Syntax

```
DELETE /2017-03-31/tags/ARN?tagKeys=TagKeys HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

Resource (p. 460)

The ARN (Amazon Resource Name) of the function. For more information, see [Tagging Lambda Functions](#) in the AWS Lambda Developer Guide.

Pattern: `arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

TagKeys (p. 460)

The list of tag keys to be deleted from the function. For more information, see [Tagging Lambda Functions](#) in the AWS Lambda Developer Guide.

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. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) 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

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 V2](#)

UpdateAlias

Using this API you can update the function version to which the alias points and the alias description. For more information, see [Introduction to AWS Lambda Aliases](#).

This requires permission for the `lambda:UpdateAlias` action.

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. 462)

The function name for which the alias is created. Note that the length constraint applies only to the 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:lambda:)?([a-zA-Z]{2}-[a-zA-Z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Name](#) (p. 462)

The alias name.

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. 462)

You can change the description of the alias using this parameter.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[FunctionVersion \(p. 462\)](#)

Using this parameter you can change the Lambda function version to which the alias points.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST | [0-9]+)

Required: No

[RevisionId \(p. 462\)](#)

An optional value you can use to ensure you are updating the latest update of the function version or alias. If the RevisionID you pass doesn't match the latest RevisionID of the function or alias, it will fail with an error message, advising you to retrieve the latest function version or alias RevisionID using either [GetFunction \(p. 413\)](#) or [GetAlias \(p. 407\)](#).

Type: String

Required: No

[RoutingConfig \(p. 462\)](#)

Specifies an additional version your alias can point to, allowing you to dictate what percentage of traffic will invoke each version. For more information, see [使用别名的流量转移 \(p. 280\)](#).

Type: [AliasRoutingConfiguration \(p. 490\)](#) 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 \(p. 463\)](#)

Lambda function ARN that is qualified using the alias name as the suffix. For example, if you create an alias called `BETA` that points to a helloworld function version, the ARN is `arn:aws:lambda:aws-regions:acct-id:function:helloworld:BETA`.

Type: String

Pattern: arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:\\$_LATEST|[a-zA-Z0-9-_]+))?

[Description \(p. 463\)](#)

Alias description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 463\)](#)

Function version to which the alias points.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$_LATEST|[0-9]+)

[Name \(p. 463\)](#)

Alias name.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?!\^ [0-9]+\\$)([a-zA-Z0-9-_]+)

[RevisionId \(p. 463\)](#)

Represents the latest updated revision of the function or alias.

Type: String

[RoutingConfig \(p. 463\)](#)

Specifies an additional function versions the alias points to, allowing you to dictate what percentage of traffic will invoke each version. For more information, see [使用别名的流量转移 \(p. 280\)](#).

Type: [AliasRoutingConfiguration \(p. 490\)](#) object

Errors

`InvalidParameterValueException`

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

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 (for example, a Lambda function or access policy statement) 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

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 V2](#)

UpdateEventSourceMapping

You can update an event source mapping. This is useful if you want to change the parameters of the existing mapping without losing your position in the stream. You can change which function will receive the stream records, but to change the stream itself, you must create a new mapping.

If you are using the versioning feature, you can update the event source mapping to map to a specific Lambda function version or alias as described in the `FunctionName` parameter. For information about the versioning feature, see [AWS Lambda Function Versioning and Aliases](#).

If you disable the event source mapping, AWS Lambda stops polling. If you enable again, it will resume polling from the time it had stopped polling, so you don't lose processing of any records. However, if you delete event source mapping and create it again, it will reset.

This operation requires permission for the `lambda:UpdateEventSourceMapping` action.

Request Syntax

```
PUT /2015-03-31/event-source-mappings/UUID HTTP/1.1
Content-type: application/json

{
  "BatchSize": number,
  "Enabled": boolean,
  "FunctionName": "string"
}
```

URI Request Parameters

The request requires the following URI parameters.

[UUID \(p. 466\)](#)

The event source mapping identifier.

Request Body

The request accepts the following data in JSON format.

[BatchSize \(p. 466\)](#)

The maximum number of stream records that can be sent to your Lambda function for a single invocation.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

Required: No

[Enabled \(p. 466\)](#)

Specifies whether AWS Lambda should actively poll the stream or not. If disabled, AWS Lambda will not poll the stream.

Type: Boolean

Required: No

[FunctionName \(p. 466\)](#)

The Lambda function to which you want the stream records sent.

You can specify a function name (for example, `Thumbnail`) or you can specify Amazon Resource Name (ARN) of the function (for example, `arn:aws:lambda:us-west-2:account-id:function:Thumbnail`). AWS Lambda also allows you to specify a partial ARN (for example, `account-id:Thumbnail`). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 characters in length.

If you are using versioning, you can also provide a qualified function ARN (ARN that is qualified with function version or alias name as suffix). For more information about versioning, see [AWS Lambda Function Versioning and Aliases](#)

Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: `(arn:aws:lambda:)?([a-z]{2}-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Required: No

Response Syntax

```
HTTP/1.1 202
Content-type: application/json

{
    "BatchSize": number,
    "EventSourceArn": "string",
    "FunctionArn": "string",
    "LastModified": number,
    "LastProcessingResult": "string",
    "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 \(p. 467\)](#)

The largest number of records that AWS Lambda will retrieve from your event source at the time of invoking your function. Your function receives an event with all the retrieved records.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[EventSourceArn \(p. 467\)](#)

The Amazon Resource Name (ARN) of the Amazon Kinesis stream that is the source of events.

Type: String

Pattern: arn:aws:([a-zA-Z0-9\-\-]+)([a-z]{2}\-[a-z]\+\-\d{1})?:(\d{12})?:(\.*)

[FunctionArn \(p. 467\)](#)

The Lambda function to invoke when AWS Lambda detects an event on the stream.

Type: String

Pattern: arn:aws:lambda:[a-z]{2}-[a-z]+\-\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\\$\\$LATEST|[a-zA-Z0-9-_]+))?

[LastModified \(p. 467\)](#)

The UTC time string indicating the last time the event mapping was updated.

Type: Timestamp

[LastProcessingResult \(p. 467\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 467\)](#)

The state of the event source mapping. It can be Creating, Enabled, Disabled, Enabling, Disabling, Updating, or Deleting.

Type: String

[StateTransitionReason \(p. 467\)](#)

The reason the event source mapping is in its current state. It is either user-requested or an AWS Lambda-initiated state transition.

Type: String

[UUID \(p. 467\)](#)

The AWS Lambda assigned opaque identifier for the mapping.

Type: String

Errors

[InvalidParameterValueException](#)

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

HTTP Status Code: 400

[ResourceConflictException](#)

The resource already exists.

HTTP Status Code: 409

[ResourceNotFoundException](#)

The resource (for example, a Lambda function or access policy statement) 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

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 V2](#)

UpdateFunctionCode

Updates the code for the specified Lambda function. This operation must only be used on an existing Lambda function and cannot be used to update the function configuration.

If you are using the versioning feature, note this API will always update the \$LATEST version of your Lambda function. For information about the versioning feature, see [AWS Lambda Function Versioning and Aliases](#).

This operation requires permission for the `lambda:UpdateFunctionCode` action.

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. 470)

The existing Lambda function name whose code you want to replace.

You can specify a function name (for example, `Thumbnail`) or you can specify Amazon Resource Name (ARN) of the function (for example, `arn:aws:lambda:us-west-2:account-id:function:Thumbnail`). AWS Lambda also allows you to specify a partial ARN (for example, `account-id:Thumbnail`). Note that the length constraint applies only to the 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:lambda:)?([a-z]{2}-[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. 470)

This boolean parameter can be used to test your request to AWS Lambda to update the Lambda function and publish a version as an atomic operation. It will do all necessary computation and validation of your code but will not upload it or publish a version. Each time this operation is invoked, the `CodeSha256` hash value of the provided code will also be computed and returned in the response.

Type: Boolean

Required: No

[Publish \(p. 470\)](#)

This boolean parameter can be used to request AWS Lambda to update the Lambda function and publish a version as an atomic operation.

Type: Boolean

Required: No

[RevisionId \(p. 470\)](#)

An optional value you can use to ensure you are updating the latest update of the function version or alias. If the `RevisionId` you pass doesn't match the latest `RevisionId` of the function or alias, it will fail with an error message, advising you to retrieve the latest function version or alias `RevisionID` using either [GetFunction \(p. 413\)](#) or [GetAlias \(p. 407\)](#).

Type: String

Required: No

[S3Bucket \(p. 470\)](#)

Amazon S3 bucket name where the .zip file containing your deployment package is stored. This bucket must reside in the same AWS Region where you are creating the Lambda function.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 63.

Pattern: ^[0-9A-Za-z\.\-_]*(\?!\.)\$

Required: No

[S3Key \(p. 470\)](#)

The Amazon S3 object (the deployment package) key name you want to upload.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

[S3ObjectVersion \(p. 470\)](#)

The Amazon S3 object (the deployment package) version you want to upload.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

[ZipFile \(p. 470\)](#)

The contents of your zip file containing your deployment package. If you are using the web API directly, the contents of the zip file must be base64-encoded. If you are using the AWS SDKs or the AWS CLI, the SDKs or CLI will do the encoding for you. For more information about creating a .zip file, see [Execution Permissions](#).

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",
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "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 \(p. 472\)](#)

It is the SHA256 hash of your function deployment package.

Type: String

[CodeSize \(p. 472\)](#)

The size, in bytes, of the function .zip file you uploaded.

Type: Long

[DeadLetterConfig \(p. 472\)](#)

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic. For more information, see [死信队列 \(p. 361\)](#).

Type: [DeadLetterConfig \(p. 492\)](#) object

[Description \(p. 472\)](#)

The user-provided description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 472\)](#)

The parent object that contains your environment's configuration settings.

Type: [EnvironmentResponse \(p. 495\)](#) object

[FunctionArn \(p. 472\)](#)

The Amazon Resource Name (ARN) assigned to the function.

Type: String

Pattern: `arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[FunctionName \(p. 472\)](#)

The name of the function. Note that the length constraint applies only to the 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 170.

Pattern: `(arn:aws:lambda:)?([a-z]{2}-[a-z]+\d{1}:)?(\d{12}:)?(function:)?([a-zA-Z0-9-_\.]+)(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[Handler \(p. 472\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 472\)](#)

The Amazon Resource Name (ARN) of the KMS key used to encrypt your function's environment variables. If empty, it means you are using the AWS Lambda default service key.

Type: String

Pattern: `(arn:aws:[a-zA-Z0-9-_\.]+:*)|()`

[LastModified \(p. 472\)](#)

The time stamp of the last time you updated the function. The time stamp is conveyed as a string complying with ISO-8601 in this way YYYY-MM-DDThh:mm:ssTZD (e.g., 1997-07-16T19:20:30+01:00). For more information, see [Date and Time Formats](#).

Type: String

[MasterArn \(p. 472\)](#)

Returns the ARN (Amazon Resource Name) of the master function.

Type: String

Pattern: `arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$\$LATEST|[a-zA-Z0-9-_]+))?`

[MemorySize \(p. 472\)](#)

The memory size, in MB, you configured for the function. Must be a multiple of 64 MB.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 472\)](#)

Represents the latest updated revision of the function or alias.

Type: String

[Role \(p. 472\)](#)

The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources.

Type: String

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/.]+`

[Runtime \(p. 472\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | java8 | python2.7 | python3.6 | dotnetcore1.0 | dotnetcore2.0 | nodejs4.3-edge | go1.x

[Timeout \(p. 472\)](#)

The function execution time at which Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 472\)](#)

The parent object that contains your function's tracing settings.

Type: [TracingConfigResponse \(p. 505\)](#) object

[Version \(p. 472\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[VpcConfig \(p. 472\)](#)

VPC configuration associated with your Lambda function.

Type: [VpcConfigResponse \(p. 507\)](#) object

Errors

CodeStorageExceededException

You have exceeded your maximum total code size per account. [Limits](#)

HTTP Status Code: 400

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

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 (for example, a Lambda function or access policy statement) 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

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 V2](#)

UpdateFunctionConfiguration

Updates the configuration parameters for the specified Lambda function by using the values provided in the request. You provide only the parameters you want to change. This operation must only be used on an existing Lambda function and cannot be used to update the function's code.

If you are using the versioning feature, note this API will always update the \$LATEST version of your Lambda function. For information about the versioning feature, see [AWS Lambda Function Versioning and Aliases](#).

This operation requires permission for the `lambda:UpdateFunctionConfiguration` action.

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",
    "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. 477)

The name of the Lambda function.

You can specify a function name (for example, `Thumbnail`) or you can specify Amazon Resource Name (ARN) of the function (for example, `arn:aws:lambda:us-west-2:account-id:function:Thumbnail`). AWS Lambda also allows you to specify a partial ARN (for example, `account-id:Thumbnail`). Note that the length constraint applies only to the ARN. If you specify only the function name, it is limited to 64 character in length.

Length Constraints: Minimum length of 1. Maximum length of 140.

Pattern: (`arn:aws:lambda:`)?([*a-z*]{2}-[*a-z*]+\-\d{1}:)?(\d{12}:)?(function:)?([*a-zA-Z0-9-_*]+)(:(\\$\text{LATEST}|[*a-zA-Z0-9-_*+]))?

Request Body

The request accepts the following data in JSON format.

[DeadLetterConfig \(p. 477\)](#)

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic. For more information, see [死信队列 \(p. 361\)](#).

Type: [DeadLetterConfig \(p. 492\)](#) object

Required: No

[Description \(p. 477\)](#)

A short user-defined function description. AWS Lambda does not use this value. Assign a meaningful description as you see fit.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[Environment \(p. 477\)](#)

The parent object that contains your environment's configuration settings.

Type: [Environment \(p. 493\)](#) object

Required: No

[Handler \(p. 477\)](#)

The function that Lambda calls to begin executing your function. For Node.js, it is the `module-name.export` value in your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: [^\s]+

Required: No

[KMSKeyArn \(p. 477\)](#)

The Amazon Resource Name (ARN) of the KMS key used to encrypt your function's environment variables. If you elect to use the AWS Lambda default service key, pass in an empty string ("") for this parameter.

Type: String

Pattern: (`arn:aws:[a-z0-9-.]+:[.*]`)|()

Required: No

[MemorySize \(p. 477\)](#)

The amount of memory, in MB, your Lambda function is given. AWS Lambda uses this memory size to infer the amount of CPU allocated to your function. Your function use-case determines your CPU and memory requirements. For example, a database operation might need less memory compared to an image processing function. 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 \(p. 477\)](#)

An optional value you can use to ensure you are updating the latest update of the function version or alias. If the RevisionID you pass doesn't match the latest RevisionId of the function or alias, it will fail with an error message, advising you to retrieve the latest function version or alias RevisionID using either [GetFunction \(p. 413\)](#) or [GetAlias \(p. 407\)](#).

Type: String

Required: No

[Role \(p. 477\)](#)

The Amazon Resource Name (ARN) of the IAM role that Lambda will assume when it executes your function.

Type: String

Pattern: arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/_]+

Required: No

[Runtime \(p. 477\)](#)

The runtime environment for the Lambda function.

To use the Python runtime v3.6, set the value to "python3.6". To use the Python runtime v2.7, set the value to "python2.7". To use the Node.js runtime v6.10, set the value to "nodejs6.10". To use the Node.js runtime v4.3, set the value to "nodejs4.3". To use the .NET Core runtime v1.0, set the value to "dotnetcore1.0". To use the .NET Core runtime v2.0, set the value to "dotnetcore2.0".

Note

Node v0.10.42 is currently marked as deprecated. You must migrate existing functions to the newer Node.js runtime versions available on AWS Lambda (nodejs4.3 or nodejs6.10) as soon as possible. Failure to do so will result in an invalid parameter error being returned. Note that you will have to follow this procedure for each region that contains functions written in the Node v0.10.42 runtime.

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | java8 | python2.7 | python3.6 | dotnetcore1.0 | dotnetcore2.0 | nodejs4.3-edge | go1.x

Required: No

[Timeout \(p. 477\)](#)

The function execution time at which AWS Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

[TracingConfig \(p. 477\)](#)

The parent object that contains your function's tracing settings.

Type: [TracingConfig \(p. 504\)](#) object

Required: No

[VpcConfig \(p. 477\)](#)

If your Lambda function accesses resources in a VPC, you provide this parameter identifying the list of security group IDs and subnet IDs. These must belong to the same VPC. You must provide at least one security group and one subnet ID.

Type: [VpcConfig \(p. 506\)](#) 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",
    "MasterArn": "string",
    "MemorySize": number,
    "RevisionId": "string",
    "Role": "string",
    "Runtime": "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 \(p. 480\)](#)

It is the SHA256 hash of your function deployment package.

Type: String

[CodeSize \(p. 480\)](#)

The size, in bytes, of the function .zip file you uploaded.

Type: Long

[DeadLetterConfig \(p. 480\)](#)

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic. For more information, see [死信队列 \(p. 361\)](#).

Type: [DeadLetterConfig \(p. 492\)](#) object

[Description \(p. 480\)](#)

The user-provided description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 480\)](#)

The parent object that contains your environment's configuration settings.

Type: [EnvironmentResponse \(p. 495\)](#) object

[FunctionArn \(p. 480\)](#)

The Amazon Resource Name (ARN) assigned to the function.

Type: String

Pattern: `arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[FunctionName \(p. 480\)](#)

The name of the function. Note that the length constraint applies only to the 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 170.

Pattern: `(arn:aws:lambda:)?([a-z]{2}-[a-z]+\d{1}:(:\d{12}:(function:)?)?([a-zA-Z0-9-_\.]+(:(\$LATEST|[a-zA-Z0-9-_]+))?)`

[Handler \(p. 480\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 480\)](#)

The Amazon Resource Name (ARN) of the KMS key used to encrypt your function's environment variables. If empty, it means you are using the AWS Lambda default service key.

Type: String

Pattern: `(arn:aws:[a-z0-9-.]+::*)|()`

[LastModified \(p. 480\)](#)

The time stamp of the last time you updated the function. The time stamp is conveyed as a string complying with ISO-8601 in this way YYYY-MM-DDThh:mm:ssTZD (e.g., 1997-07-16T19:20:30+01:00). For more information, see [Date and Time Formats](#).

Type: String

[MasterArn \(p. 480\)](#)

Returns the ARN (Amazon Resource Name) of the master function.

Type: String

Pattern: `arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[MemorySize \(p. 480\)](#)

The memory size, in MB, you configured for the function. Must be a multiple of 64 MB.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

[RevisionId \(p. 480\)](#)

Represents the latest updated revision of the function or alias.

Type: String

[Role \(p. 480\)](#)

The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources.

Type: String

Pattern: `arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-/]+`

[Runtime \(p. 480\)](#)

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | java8 | python2.7 | python3.6 | dotnetcore1.0 | dotnetcore2.0 | nodejs4.3-edge | go1.x

[Timeout \(p. 480\)](#)

The function execution time at which Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Integer

Valid Range: Minimum value of 1.

[TracingConfig \(p. 480\)](#)

The parent object that contains your function's tracing settings.

Type: [TracingConfigResponse \(p. 505\)](#) object

[Version \(p. 480\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (`\$LATEST` | [0-9]+)

[VpcConfig \(p. 480\)](#)

VPC configuration associated with your Lambda function.

Type: [VpcConfigResponse \(p. 507\)](#) object

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. For example, if you provided an IAM role for AWS Lambda to assume in the `CreateFunction` or the `UpdateFunctionConfiguration` API, that AWS Lambda is unable to assume you will get this exception.

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.

HTTP Status Code: 409

ResourceNotFoundException

The resource (for example, a Lambda function or access policy statement) 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

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 V2

Data Types

The following data types are supported:

- AccountLimit (p. 485)
- AccountUsage (p. 487)
- AliasConfiguration (p. 488)
- AliasRoutingConfiguration (p. 490)
- Concurrency (p. 491)
- DeadLetterConfig (p. 492)
- Environment (p. 493)
- EnvironmentError (p. 494)
- EnvironmentResponse (p. 495)
- EventSourceMappingConfiguration (p. 496)
- FunctionCode (p. 498)
- FunctionCodeLocation (p. 499)
- FunctionConfiguration (p. 500)
- TracingConfig (p. 504)
- TracingConfigResponse (p. 505)
- VpcConfig (p. 506)
- VpcConfigResponse (p. 507)

AccountLimit

Provides limits of code size and concurrency associated with the current account and region.

Contents

CodeSizeUnzipped

Size, in bytes, of code/dependencies that you can zip into a deployment package (uncompressed zip/jar size) for uploading. The default limit is 250 MB.

Type: Long

Required: No

CodeSizeZipped

Size, in bytes, of a single zipped code/dependencies package you can upload for your Lambda function(.zip/.jar file). Try using Amazon S3 for uploading larger files. Default limit is 50 MB.

Type: Long

Required: No

ConcurrentExecutions

Number of simultaneous executions of your function per region. For more information or to request a limit increase for concurrent executions, see [Lambda Function Concurrent Executions](#). The default limit is 1000.

Type: Integer

Required: No

TotalCodeSize

Maximum size, in bytes, of a code package you can upload per region. The default size is 75 GB.

Type: Long

Required: No

UnreservedConcurrentExecutions

The number of concurrent executions available to functions that do not have concurrency limits set. For more information, see [管理并发 \(p. 350\)](#).

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 V2](#)

AccountUsage

Provides code size usage and function count associated with the current account and region.

Contents

FunctionCount

The number of your account's existing functions per region.

Type: Long

Required: No

TotalCodeSize

Total size, in bytes, of the account's deployment packages per region.

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 V2](#)

AliasConfiguration

Provides configuration information about a Lambda function version alias.

Contents

AliasArn

Lambda function ARN that is qualified using the alias name as the suffix. For example, if you create an alias called `BETA` that points to a `helloworld` function version, the ARN is `arn:aws:lambda:aws-regions:acct-id:function:helloworld:BETA`.

Type: String

Pattern: `arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

Required: No

Description

Alias description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

FunctionVersion

Function version to which the alias points.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

Required: No

Name

Alias name.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(?!^[\0-9]+$)([a-zA-Z0-9-_]+)`

Required: No

RevisionId

Represents the latest updated revision of the function or alias.

Type: String

Required: No

RoutingConfig

Specifies an additional function versions the alias points to, allowing you to dictate what percentage of traffic will invoke each version. For more information, see [使用别名的流量转移 \(p. 280\)](#).

Type: [AliasRoutingConfiguration \(p. 490\)](#) 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 V2](#)

AliasRoutingConfiguration

The parent object that implements what percentage of traffic will invoke each function version. For more information, see [使用别名的流量转移 \(p. 280\)](#).

Contents

AdditionalVersionWeights

Set this value to dictate what percentage of traffic will invoke the updated function version. If set to an empty string, 100 percent of traffic will invoke `function-version`. For more information, see [使用别名的流量转移 \(p. 280\)](#).

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 V2](#)

Concurrency

Contents

ReservedConcurrentExecutions

The number of concurrent executions reserved for this function. For more information, see [管理并发 \(p. 350\)](#).

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 V2](#)

DeadLetterConfig

The Amazon Resource Name (ARN) of an Amazon SQS queue or Amazon SNS topic you specify as your Dead Letter Queue (DLQ). For more information, see [死信队列 \(p. 361\)](#).

Contents

TargetArn

The Amazon Resource Name (ARN) of an Amazon SQS queue or Amazon SNS topic you specify as your Dead Letter Queue (DLQ). [死信队列 \(p. 361\)](#). For more information, see [死信队列 \(p. 361\)](#).

Type: String

Pattern: `(arn:aws:[a-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 V2](#)

Environment

The parent object that contains your environment's configuration settings.

Contents

Variables

The key-value pairs that represent your environment's configuration settings.

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 V2](#)

EnvironmentError

The parent object that contains error information associated with your configuration settings.

Contents

ErrorCode

The error code returned by the environment error object.

Type: String

Required: No

Message

The message returned by the environment error object.

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 V2](#)

EnvironmentResponse

The parent object returned that contains your environment's configuration settings or any error information associated with your configuration settings.

Contents

Error

The parent object that contains error information associated with your configuration settings.

Type: [EnvironmentError \(p. 494\)](#) object

Required: No

Variables

The key-value pairs returned that represent your environment's configuration settings or error information.

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 V2](#)

EventSourceMappingConfiguration

Describes mapping between an Amazon Kinesis stream and a Lambda function.

Contents

BatchSize

The largest number of records that AWS Lambda will retrieve from your event source at the time of invoking your function. Your function receives an event with all the retrieved records.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

Required: No

EventSourceArn

The Amazon Resource Name (ARN) of the Amazon Kinesis stream that is the source of events.

Type: String

Pattern: arn:aws:([a-zA-Z0-9\-_]\+)([a-z]{2}\-[a-z]\+\-\d{1})?\:(\d{12})?\:(\.*)

Required: No

FunctionArn

The Lambda function to invoke when AWS Lambda detects an event on the stream.

Type: String

Pattern: arn:aws:lambda:[a-z]{2}\-[a-z]\+\-\d{1}:\d{12}:function:[a-zA-Z0-9\-_]\+(:(\\$\\$LATEST|[a-zA-Z0-9\-_]\+))?

Required: No

LastModified

The UTC time string indicating the last time the event mapping was updated.

Type: Timestamp

Required: No

LastProcessingResult

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

Required: No

State

The state of the event source mapping. It can be `Creating`, `Enabled`, `Disabled`, `Enabling`, `Disabling`, `Updating`, or `Deleting`.

Type: String

Required: No

StateTransitionReason

The reason the event source mapping is in its current state. It is either user-requested or an AWS Lambda-initiated state transition.

Type: String

Required: No

UUID

The AWS Lambda assigned opaque identifier for the 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 V2](#)

FunctionCode

The code for the Lambda function.

Contents

S3Bucket

Amazon S3 bucket name where the .zip file containing your deployment package is stored. This bucket must reside in the same AWS region where you are creating the Lambda function.

Type: String

Length Constraints: Minimum length of 3. Maximum length of 63.

Pattern: ^[0-9A-Za-z\.\-_]*(\?!\.)\$

Required: No

S3Key

The Amazon S3 object (the deployment package) key name you want to upload.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

S3ObjectVersion

The Amazon S3 object (the deployment package) version you want to upload.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Required: No

ZipFile

The contents of your zip file containing your deployment package. If you are using the web API directly, the contents of the zip file must be base64-encoded. If you are using the AWS SDKs or the AWS CLI, the SDKs or CLI will do the encoding for you. For more information about creating a .zip file, see [Execution Permissions](#) in the AWS Lambda Developer Guide.

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 V2](#)

FunctionCodeLocation

The object for the Lambda function location.

Contents

Location

The presigned URL you can use to download the function's .zip file that you previously uploaded. The URL is valid for up to 10 minutes.

Type: String

Required: No

RepositoryType

The repository from which you can download the function.

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 V2](#)

FunctionConfiguration

A complex type that describes function metadata.

Contents

CodeSha256

It is the SHA256 hash of your function deployment package.

Type: String

Required: No

CodeSize

The size, in bytes, of the function .zip file you uploaded.

Type: Long

Required: No

DeadLetterConfig

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic. For more information, see [死信队列 \(p. 361\)](#).

Type: [DeadLetterConfig \(p. 492\)](#) object

Required: No

Description

The user-provided description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

Environment

The parent object that contains your environment's configuration settings.

Type: [EnvironmentResponse \(p. 495\)](#) object

Required: No

FunctionArn

The Amazon Resource Name (ARN) assigned to the function.

Type: String

Pattern: arn:aws:lambda:[a-zA-Z]{2}-[a-zA-Z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+\.(?:\$LATEST|[a-zA-Z0-9-_]+))?

Required: No

FunctionName

The name of the function. Note that the length constraint applies only to the 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 170.

Pattern: `(arn:aws:lambda:[a-zA-Z]{2}-[a-zA-Z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+)(:$LATEST|[a-zA-Z0-9-_\.]+)`?

Required: No

Handler

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

Required: No

KMSKeyArn

The Amazon Resource Name (ARN) of the KMS key used to encrypt your function's environment variables. If empty, it means you are using the AWS Lambda default service key.

Type: String

Pattern: `(arn:aws:[a-zA-Z0-9-_\.]+:*)|()`

Required: No

LastModified

The time stamp of the last time you updated the function. The time stamp is conveyed as a string complying with ISO-8601 in this way YYYY-MM-DDThh:mm:ssTZD (e.g., 1997-07-16T19:20:30+01:00). For more information, see [Date and Time Formats](#).

Type: String

Required: No

MasterArn

Returns the ARN (Amazon Resource Name) of the master function.

Type: String

Pattern: `arn:aws:lambda:[a-zA-Z]{2}-[a-zA-Z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_\.]+(:|$LATEST|[a-zA-Z0-9-_\.]+)?`

Required: No

MemorySize

The memory size, in MB, you configured for the function. Must be a multiple of 64 MB.

Type: Integer

Valid Range: Minimum value of 128. Maximum value of 3008.

Required: No

RevisionId

Represents the latest updated revision of the function or alias.

Type: String

Required: No

Role

The Amazon Resource Name (ARN) of the IAM role that Lambda assumes when it executes your function to access any other Amazon Web Services (AWS) resources.

Type: String

Pattern: arn:aws:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\\-_/.]+

Required: No

Runtime

The runtime environment for the Lambda function.

Type: String

Valid Values: nodejs | nodejs4.3 | nodejs6.10 | java8 | python2.7 | python3.6 | dotnetcore1.0 | dotnetcore2.0 | nodejs4.3-edge | go1.x

Required: No

Timeout

The function execution time at which Lambda should terminate the function. Because the execution time has cost implications, we recommend you set this value based on your expected execution time. The default is 3 seconds.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

TracingConfig

The parent object that contains your function's tracing settings.

Type: [TracingConfigResponse \(p. 505\)](#) 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

VPC configuration associated with your Lambda function.

Type: [VpcConfigResponse \(p. 507\)](#) 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 V2](#)

TracingConfig

The parent object that contains your function's tracing settings.

Contents

Mode

Can be either PassThrough or Active. If PassThrough, Lambda will only trace the request from an upstream service if it contains a tracing header with "sampled=1". If Active, Lambda will respect any tracing header it receives from an upstream service. If no tracing header is received, Lambda will call X-Ray for a tracing decision.

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 V2](#)

TracingConfigResponse

Parent object of the tracing information associated with your Lambda function.

Contents

Mode

The tracing mode associated with your Lambda function.

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 V2](#)

VpcConfig

If your Lambda function accesses resources in a VPC, you provide this parameter identifying the list of security group IDs and subnet IDs. These must belong to the same VPC. You must provide at least one security group and one subnet ID.

Contents

SecurityGroupIds

A list of one or more security groups IDs in your VPC.

Type: Array of strings

Array Members: Maximum number of 5 items.

Required: No

SubnetIds

A list of one or more subnet IDs in your VPC.

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 V2](#)

VpcConfigResponse

VPC configuration associated with your Lambda function.

Contents

SecurityGroupIds

A list of security group IDs associated with the Lambda function.

Type: Array of strings

Array Members: Maximum number of 5 items.

Required: No

SubnetIds

A list of subnet IDs associated with the Lambda function.

Type: Array of strings

Array Members: Maximum number of 16 items.

Required: No

VpcId

The VPC ID associated with your Lambda function.

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 V2](#)

文档历史记录

下表介绍了对 AWS Lambda 开发人员指南的一些重要更改。

此历史记录的相关日期：

- 当前产品版本：2015-03-31
- 文档上次更新时间：2018 年 1 月 25 日

更改	描述	日期
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 添加了对 Go 和 .NET 2.0 的运行时支持。有关更多信息，请参阅 使用 Go 编写 Lambda 函数的编程模型 (p. 60) 和 使用 C# 编写 Lambda 函数的编程模型 (p. 68) 。	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. 103) 。	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 管理并发 (p. 350) 。	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 使用别名的流量转移 (p. 280) 。	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 (p. 293) 。	2017 年 11 月 28 日
中国 (北京) 区域	AWS Lambda 目前在中国 (北京) 区域中提供。有关 Lambda 区域和端点的更多信息，请参阅 AWS General Reference 中的 区域和端点 。	2017 年 11 月 9 日
推出 SAM Local	AWS Lambda 推出 SAM Local，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 使用 SAM Local 在本地测试您的无服务应用程序 (公开测试版) (p. 96) 。	2017 年 8 月 11 日
加拿大 (中部) 区域	AWS Lambda 目前已在加拿大 (中部) 区域可用。有关 Lambda 区域和端点的更多信息，请参阅 AWS General Reference 中的 区域和端点 。	2017 年 6 月 22 日
南美洲 (圣保罗) 区域	AWS Lambda 目前已在南美洲 (圣保罗) 区域可用。有关 Lambda 区域和端点的更多信息，请参阅 AWS General Reference 中的 区域和端点 。	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. 302) 。	2017 年 4 月 19 日

更改	描述	日期
亚太地区（孟买）区域	AWS Lambda 目前已在 亚太地区（孟买）区域 可用。有关 Lambda 区域和端点的更多信息，请参阅 AWS General Reference 中的 区域和端点 。	2017 年 3 月 28 日
AWS Lambda 现在支持 Node.js 运行时 v6.10	AWS Lambda 添加了对 Node.js 运行时 v6.10 的支持。有关更多信息，请参阅 编程模型 (Node.js) (p. 18) 。	2017 年 3 月 22 日
欧洲 (伦敦) 区域	AWS Lambda 目前已在 欧洲 (伦敦) 区域 可用。有关 Lambda 区域和端点的更多信息，请参阅 AWS General Reference 中的 区域和端点 。	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 推出了以下功能：</p> <ul style="list-style-type: none"> AWS Lambda 增加对 C# 的支持。有关更多信息，请参阅使用 C# 编写 Lambda 函数的编程模型 (p. 68)。 Lambda@Edge (预览版) 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AWS Lambda@Edge (p. 261)。 添加了使用 AWS CodePipeline、AWS CodeBuild 和 AWS CloudFormation 自动部署无服务器应用程序的教程。有关更多信息，请参阅基于 Lambda 应用程序的自动化部署 (p. 288)。 更新使用 Amazon CloudWatch (p. 296)以包含死信队列 (p. 361)的相关内容，您可以配置后者，以便检索有关失败的 Lambda 函数异步调用的信息。 	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 Amazon Lex (p. 153) 。	2016 年 11 月 30 日
美国西部 (加利福利亚北部) 区域	AWS Lambda 目前在 美国西部 (加利福利亚北部) 区域 中可用。有关 Lambda 区域和端点的更多信息，请参阅 AWS General Reference 中的 区域和端点 。	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 Lambda 在此版本中引入了以下功能。</p> <ul style="list-style-type: none"> 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，请参阅部署基于 Lambda 的应用程序 (p. 264)。 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，请参阅环境变量 (p. 354)。 	2016 年 11 月 18 日
在 入门 (p. 3) 下添加了一个教程，讲述如何使用 Lambda 控制台创建 Amazon API Gateway 终端节点	本指南说明如何通过 为代理资源配置代理集成 中介绍的新功能无缝集成 Lambda 函数与 API。有关更多信息，请参阅 步骤 3：使用 Lambda 和 API 网关 创建简单的微服务 (p. 235) 。	2016 年 8 月 29 日
亚太区域 (首尔)	AWS Lambda 目前在亚太区域 (首尔) 中可用。有关 Lambda 区域和端点的更多信息，请参阅 AWS General Reference 中的 区域和端点 。	2016 年 8 月 29 日

更改	描述	日期
亚太区域（悉尼）	Lambda 目前在亚太区域（悉尼）中可用。有关 Lambda 区域和端点的更多信息，请参阅 AWS General Reference 中的 区域和端点 。	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 创建简单的 Lambda 函数 (p. 8) 。	2016 年 6 月 23 日
AWS Lambda 现在支持 Node.js 运行时 v4.3	AWS Lambda 添加了对 Node.js 运行时 v4.3 的支持。有关更多信息，请参阅 编程模型 (Node.js) (p. 18) 。	2016 年 4 月 7 日
欧洲（法兰克福）区域	Lambda 目前在欧洲（法兰克福）区域可用。有关 Lambda 区域和端点的更多信息，请参阅 AWS General Reference 中的 区域和端点 。	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 Amazon VPC 中的资源 (p. 129) 。有关演练示例，请参阅 教程：配置 Lambda 函数以访问 Amazon VPC 中的资源 (p. 131) 。	2016 年 2 月 11 日
内容重新组织	<p>重新组织过的内容现在提供了以下项：</p> <ul style="list-style-type: none"> • 入门 (p. 3) - 包含基于控制台的练习，您可在其中创建 Hello World Lambda 函数。您将了解 AWS Lambda 控制台功能，包括使您只需单击几下即可创建 Lambda 函数的蓝图。 • 使用案例 (p. 165) - 提供如何执行以下操作的示例：将 AWS Lambda 和其他 AWS 服务结合使用或将自定义应用程序用作事件源、通过 HTTPS 进行调用以及设置 AWS Lambda 以按照计划的时间间隔调用 Lambda 函数。 • 编程模型 (p. 18) - 介绍编程模型核心概念并给出特定于语言的详细信息。无论您选择哪种语言，都有一个为 Lambda 函数编写代码的通用模式。 • 创建部署程序包 (p. 77) - 介绍如何为采用 AWS Lambda 支持的语言（Python、Java 和 Node.js）编写的 Lambda 函数代码创建部署程序包。 	2015 年 12 月 9 日
已更新 AWS Lambda 运行时。	<p>已在此版本中使用以下软件开发工具包和 Linux 内核版本来更新 AWS Lambda 运行时：</p> <ul style="list-style-type: none"> • 适用于 JavaScript 的 AWS 开发工具包：2.2.12 • Boto 软件开发工具包：1.2.1 • Linux 内核版本：4.9.62-21.56.amzn1.x86_64 <p>有关更多信息，请参阅Lambda 执行环境和可用库 (p. 366)。</p>	2015 年 11 月 4 日

更改	描述	日期
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>AWS Lambda 在此版本引入了以下功能。</p> <ul style="list-style-type: none"> • Python：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 编程模型 (p. 18)。 • 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 函数版本控制和别名 (p. 264)。 • 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 将 AWS Lambda 用于计划的事件 (p. 249)。 • 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。 	2015 年 10 月 8 日
两项新的演练	<p>添加了以下新演练。它们都使用了 Java Lambda 函数。</p> <p>教程：将 AWS Lambda 与 Amazon DynamoDB 结合使用 (p. 192)</p> <p>使用 AWS Lambda 作为移动应用程序后端（自定义事件源：Android）(p. 237)</p>	2015 年 8 月 27 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，请参阅 教程：将 AWS Lambda 与 Amazon DynamoDB 结合使用 (p. 192) 。	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 软件开发工具包（例如：适用于 Java 的 AWS 开发工具包、适用于 Android 的 AWS 软件开发工具包或适用于 iOS 的 AWS 软件开发工具包）。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 将 AWS Lambda 与 Amazon API Gateway 结合使用（按需并通过 HTTPS）(p. 222)。</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 9 日
AWS Lambda 控制台现可提供蓝图，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。每个蓝图为您 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 入门 (p. 3) 。	在此发行版本中
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 编程模型 (p. 18) 。	2015 年 6 月 15 日

更改	描述	日期
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"> 入门 (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

AWS 词汇表

有关最新 AWS 术语，请参阅 AWS General Reference 中的 [AWS 词汇表](#)。