
AWS Lambda

Guía para desarrolladores



AWS Lambda: Guía para desarrolladores

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

| | |
|--|-----|
| ¿Qué es AWS Lambda? | 1 |
| ¿Cuándo debo utilizar Lambda? | 1 |
| ¿Utiliza AWS Lambda por primera vez? | 2 |
| Funciones de Lambda | 3 |
| Requisitos informáticos: configuración de una función de Lambda | 3 |
| Tipos de invocación | 4 |
| Creación de funciones de Lambda | 5 |
| Creación del código de la función de Lambda | 5 |
| Implementación de código y creación de una función de Lambda | 6 |
| Monitorización y solución de problemas | 7 |
| Ejemplos de aplicaciones basadas en AWS Lambda | 8 |
| Temas relacionados | 8 |
| Modelo de programación | 8 |
| Creación de un paquete de implementación | 64 |
| Control de versiones y alias | 80 |
| Variables de entorno | 97 |
| Etiquetado de funciones de Lambda | 105 |
| Compatibilidad con VPC | 108 |
| Configuración de una función de Lambda para el acceso a Amazon VPC | 109 |
| Acceso a Internet para funciones de Lambda | 110 |
| Directrices para la configuración de funciones de Lambda habilitadas para VPC | 110 |
| Tutoriales: Acceso a los recursos de una Amazon VPC | 111 |
| Solución de problemas y monitorización | 120 |
| Escenarios de solución de problemas | 121 |
| Acceso a las métricas de CloudWatch | 122 |
| Acceso a los logs de CloudWatch | 124 |
| Métricas | 125 |
| Colas de mensajes fallidos | 128 |
| Creación de aplicaciones con AWS Lambda | 130 |
| Ejemplo 1 | 131 |
| Ejemplo 2 | 132 |
| Ejemplo 3: Una aplicación personalizada publica eventos e invoca una función de Lambda | 132 |
| Lectura recomendada | 134 |
| Mapeo de orígenes de eventos | 134 |
| Mapeo de orígenes de eventos para los servicios de AWS | 135 |
| Mapeo de orígenes de eventos para los servicios de AWS basados en flujos | 136 |
| Mapeo de orígenes de eventos para las aplicaciones personalizadas | 137 |
| Orígenes de eventos admitidos | 138 |
| Amazon S3 | 139 |
| Amazon DynamoDB | 140 |
| Amazon Kinesis Data Streams | 140 |
| Amazon Simple Notification Service | 140 |
| Amazon Simple Email Service | 141 |
| Amazon Cognito | 141 |
| AWS CloudFormation | 142 |
| Amazon CloudWatch Logs | 142 |
| Amazon CloudWatch Events | 142 |
| AWS CodeCommit | 143 |
| Eventos programados (basados en Amazon CloudWatch Events) | 143 |
| AWS Config | 143 |
| Amazon Alexa | 144 |
| Amazon Lex | 144 |
| Amazon API Gateway | 144 |
| Botón de AWS IoT | 145 |

| | |
|--|-----|
| Amazon CloudFront | 145 |
| Amazon Kinesis Data Firehose | 145 |
| Otros orígenes de eventos: invocación de una función de Lambda bajo demanda | 146 |
| Datos de los eventos de muestra | 146 |
| Implementación de aplicaciones basadas en Lambda | 156 |
| Implementación de aplicaciones sin servidor mediante AWS CloudFormation | 156 |
| Uso de AWS Serverless Application Model (AWS SAM) | 156 |
| Creación de su propia aplicación sin servidor | 161 |
| Automatización de la implementación de aplicaciones basadas en Lambda | 164 |
| Prueba local de aplicaciones sin servidor mediante SAM Local (en versión beta pública) | 169 |
| Tiempos de ejecución admitidos | 170 |
| Requisitos para usar SAM Local | 171 |
| Introducción al uso de SAM Local | 171 |
| Solución de problemas de aplicaciones basadas en Lambda | 176 |
| Rastreo de aplicaciones basadas en Lambda con AWS X-Ray | 177 |
| Configuración de AWS X-Ray con Lambda | 177 |
| Mapa de servicio de Lambda en AWS X-Ray | 179 |
| Lambda como rastro de AWS X-Ray | 179 |
| Emisión de segmentos de rastro desde una función de Lambda | 180 |
| El demonio de AWS X-Ray en el entorno de Lambda | 185 |
| Uso de variables de entorno para comunicarse con AWS X-Ray | 185 |
| Ejemplos de rastros de Lambda en la consola de AWS X-Ray | 186 |
| Funcionamiento | 188 |
| ¿Cómo ejecuta AWS Lambda mi código? El modelo de contenedores | 188 |
| Ejecuciones simultáneas | 189 |
| Velocidad de solicitudes de ejecución simultánea | 189 |
| Límite de ejecuciones simultáneas | 189 |
| Escalado | 190 |
| Reintentos si se producen errores | 192 |
| Modelo de permisos | 193 |
| Administración de permisos mediante un rol de IAM (rol de ejecución) | 193 |
| Administración de permisos mediante una política de función de Lambda | 194 |
| Lectura recomendada | 195 |
| Entorno de ejecución | 195 |
| Variables de entorno disponibles para las funciones de Lambda | 196 |
| Lectura recomendada | 198 |
| Introducción | 199 |
| Paso 1: Configuración de una cuenta de AWS y de la AWS CLI | 199 |
| Paso 1.1: Configuración de una cuenta | 199 |
| Paso 1.2: Configuración de la AWS CLI | 202 |
| Paso 2: Creación de una función HelloWorld de Lambda y exploración de la consola | 202 |
| Preparación para la introducción | 203 |
| Paso 2.1: Creación de una función Hello World de Lambda | 203 |
| Paso 2.2: Invocación de la función de Lambda | 206 |
| Paso 2.3: Creación de una función de Lambda escrita en Java (opcional) | 209 |
| Paso 2.4: Creación de una función de Lambda escrita en C# (opcional) | 210 |
| Paso 3: Creación de un microservicio sencillo utilizando Lambda y API Gateway | 211 |
| Paso siguiente | 211 |
| Paso 3.1: Creación de una API utilizando Amazon API Gateway | 211 |
| Paso 3.2: Prueba de envío de una solicitud HTTPS | 212 |
| Paso 3.3: Uso de otros proyectos (opcional) | 212 |
| Siguientes pasos | 213 |
| Casos de uso | 214 |
| Amazon S3 | 214 |
| Tutorial | 215 |
| Kinesis | 232 |
| Tutorial | 233 |

| | |
|--|-----|
| Amazon DynamoDB | 243 |
| Tutorial | 244 |
| AWS CloudTrail | 254 |
| Tutorial | 255 |
| Amazon SNS | 270 |
| Tutorial | 270 |
| Amazon API Gateway | 275 |
| Uso de AWS Lambda con Amazon API Gateway (bajo demanda a través de HTTPS) | 276 |
| Backend móvil (Android) | 289 |
| Tutorial | 291 |
| Eventos programados | 302 |
| Tutorial | 303 |
| Aplicaciones de usuario personalizadas | 309 |
| Tutorial | 309 |
| AWS Lambda@Edge | 317 |
| Cómo crear funciones de Lambda para Lambda@Edge | 318 |
| Configuración de permisos y roles de IAM para Lambda@Edge | 319 |
| Creación de una función de Lambda@Edge y de un disparador para un evento de CloudFront | 321 |
| Adición de disparadores para una función de Lambda@Edge (consola de AWS Lambda) | 322 |
| Creación de funciones para Lambda@Edge | 323 |
| Ejemplo: Prueba A/B | 324 |
| Ejemplo: Redirección HTTP | 325 |
| Edición de una función de Lambda para Lambda@Edge | 325 |
| Comprobación y depuración | 326 |
| Límites de Lambda@Edge | 327 |
| Registro de llamadas a la API con AWS CloudTrail | 328 |
| Información de AWS Lambda en CloudTrail | 328 |
| Información sobre las entradas de archivos de log de AWS Lambda | 329 |
| prácticas recomendadas | 331 |
| Código de la función | 331 |
| Función de configuración | 332 |
| Alarms y métricas | 332 |
| Invocaciones con eventos de flujo | 333 |
| Invocaciones asíncronas | 333 |
| VPC de Lambda | 333 |
| Límites | 335 |
| Límites de AWS Lambda | 335 |
| Errores relacionados con los límites de AWS Lambda | 336 |
| Autenticación y control de acceso | 337 |
| Autenticación | 337 |
| Control de acceso | 338 |
| Información general sobre la administración de acceso | 338 |
| Recursos y operaciones de AWS Lambda | 339 |
| Titularidad de los recursos | 339 |
| Administrar el acceso a los recursos | 340 |
| Especificación de elementos de política: acciones, efectos, recursos y entidades principales | 342 |
| Especificar condiciones en una política | 342 |
| Uso de políticas basadas en identidad (políticas de IAM) | 343 |
| Permisos necesarios para usar la consola de AWS Lambda | 344 |
| Políticas administradas (predefinidas) de AWS para AWS Lambda | 344 |
| Ejemplos de políticas administradas por el cliente | 345 |
| Permisos de la consola | 348 |
| Uso de políticas basadas en recursos (políticas de funciones de Lambda) | 355 |
| Permitir que Amazon S3 invoque una función de Lambda | 357 |
| Permitir que Amazon API Gateway invoque una función de Lambda | 358 |
| Escenario con varias cuentas | 358 |
| Recuperar una política de función de Lambda | 359 |

| | |
|---|-----|
| Eliminar permisos | 359 |
| Trabajar con el control de versiones, los alias y los permisos de las funciones de Lambda | 359 |
| Información sobre los permisos de la API de Lambda | 359 |
| Plantillas de política | 362 |
| Basic: "Permisos básicos de Lambda" | 362 |
| VPCAccess: "Permisos de acceso a VPC de Lambda" | 363 |
| Kinesis: "Permisos de sondeador de flujos de Kinesis de Lambda" | 363 |
| DynamoDB: "Permisos de sondeador de flujos de DynamoDB de Lambda" | 364 |
| Edge: "Permisos básicos de Lambda Edge" | 364 |
| RedrivePolicySNS: "Permisos de cola de mensajes fallidos de SNS" | 364 |
| RedrivePolicySQS: "Permisos de cola de mensajes fallidos de SQS" | 365 |
| | 365 |
| CloudFormation: "Permisos de solo lectura de la pila de CloudFormation" | 365 |
| AMI: "Permisos de solo lectura de AMI" | 365 |
| KMS: "Permisos de descifrado de KMS" | 366 |
| S3: "Permisos de solo lectura de objetos de S3" | 366 |
| Elasticsearch: "Permisos de Elasticsearch" | 366 |
| SES: "Permisos de rebote de SES" | 366 |
| TestHarness: "Permisos para pruebas TestHarness" | 367 |
| Microservice: "Permisos para microservicio sencillo" | 367 |
| VPN: "Permisos de monitorización de conexiones VPN" | 367 |
| SQS: "Permisos de sondeador SQS" | 368 |
| IoTButton: "Permisos de botón IoT de AWS" | 368 |
| RekognitionNoDataAccess: "Permisos sin datos de Amazon Rekognition" | 368 |
| RekognitionReadOnlyAccess: "Permisos de solo lectura de Amazon Rekognition" | 369 |
| RekognitionWriteOnlyAccess: "Permisos de solo escritura de Amazon Rekognition" | 369 |
| Referencia de la API | 370 |
| Actions | 370 |
| AddPermission | 372 |
| CreateAlias | 376 |
| CreateEventSourceMapping | 380 |
| CreateFunction | 385 |
| DeleteAlias | 393 |
| DeleteEventSourceMapping | 395 |
| DeleteFunction | 398 |
| DeleteFunctionConcurrency | 401 |
| GetAccountSettings | 403 |
| GetAlias | 405 |
| GetEventSourceMapping | 408 |
| GetFunction | 411 |
| GetFunctionConfiguration | 415 |
| GetPolicy | 420 |
| Invoke | 423 |
| InvokeAsync | 428 |
| ListAliases | 431 |
| ListEventSourceMappings | 434 |
| ListFunctions | 437 |
| ListTags | 440 |
| ListVersionsByFunction | 442 |
| PublishVersion | 445 |
| PutFunctionConcurrency | 450 |
| RemovePermission | 452 |
| TagResource | 454 |
| UntagResource | 456 |
| UpdateAlias | 458 |
| UpdateEventSourceMapping | 462 |
| UpdateFunctionCode | 466 |

| | |
|---------------------------------------|-----|
| UpdateFunctionConfiguration | 472 |
| Data Types | 478 |
| AccountLimit | 480 |
| AccountUsage | 482 |
| AliasConfiguration | 483 |
| AliasRoutingConfiguration | 485 |
| Concurrency | 486 |
| DeadLetterConfig | 487 |
| Environment | 488 |
| EnvironmentError | 489 |
| EnvironmentResponse | 490 |
| EventSourceMappingConfiguration | 491 |
| FunctionCode | 493 |
| FunctionCodeLocation | 494 |
| FunctionConfiguration | 495 |
| TracingConfig | 499 |
| TracingConfigResponse | 500 |
| VpcConfig | 501 |
| VpcConfigResponse | 502 |
| Historial de revisión | 503 |
| AWS Glossary | 509 |

¿Qué es AWS Lambda?

AWS Lambda es un servicio informático que permite ejecutar código sin aprovisionar ni administrar servidores. AWS Lambda ejecuta el código solo cuando es necesario, y se escala de manera automática, pasando de pocas solicitudes al día a miles por segundo. Solo se paga el tiempo de computación que se consume; no hay ningún cargo mientras el código no se ejecuta. Con AWS Lambda, puede ejecutar código para prácticamente cualquier tipo de aplicación o servicio de backend, y sin que se requiera ningún tipo de administración. AWS Lambda ejecuta el código en una infraestructura informática de alta disponibilidad y realiza todas las tareas de administración de los recursos informáticos, incluido el mantenimiento del servidor y del sistema operativo, el aprovisionamiento de capacidad y el escalado automático, así como la monitorización del código y las funciones de registro. Lo único que tiene que hacer es proporcionar el código en uno de los lenguajes que admite AWS Lambda (actualmente, Node.js, Java, C# y Python).

Puede utilizar AWS Lambda para ejecutar código como respuesta a eventos, por ejemplo, cambios en los datos de un bucket de Amazon S3 o de una tabla de Amazon DynamoDB; para ejecutar código como respuesta a solicitudes HTTP utilizando Amazon API Gateway; o invocar código utilizando las llamadas a las API realizadas a través de los SDK de AWS. Con estas capacidades, puede utilizar Lambda para crear fácilmente disparadores de procesamiento de datos para servicios de AWS como Amazon S3 y Amazon DynamoDB, procesar datos de flujos almacenados en Kinesis o crear su propio backend que opera con el nivel de seguridad, desempeño y escala de AWS.

También puede crear aplicaciones [sin servidor](#) compuestas por funciones activadas por eventos e implementarlas automáticamente utilizando AWS CodePipeline y AWS CodeBuild. Para obtener más información, consulte [Implementación de aplicaciones basadas en Lambda \(p. 156\)](#).

Para obtener más información acerca del entorno de ejecución de AWS Lambda, consulte [Entorno de ejecución de Lambda y bibliotecas disponibles \(p. 195\)](#). Para obtener información acerca de cómo determina AWS Lambda los recursos informáticos necesarios para ejecutar el código, consulte [Requisitos informáticos: configuración de una función de Lambda \(p. 3\)](#).

¿Cuándo debo utilizar AWS Lambda?

AWS Lambda es una plataforma de computación ideal para muchas situaciones, siempre que el código de las aplicaciones pueda escribirse en los lenguajes admitidos por AWS Lambda (es decir, Node.js, Java, C# y Python) y que pueda ejecutarse dentro del entorno de tiempo de ejecución estándar de AWS Lambda y de los recursos proporcionados por Lambda.

Cuando se utiliza AWS Lambda, solo es necesario preocuparse por el código. AWS Lambda administra la flota de computación, que ofrece una combinación equilibrada de memoria, CPU, red y otros recursos. Esto es a cambio de flexibilidad, lo que significa que no se puede iniciar sesión en instancias de computación ni personalizar el sistema operativo ni el tiempo de ejecución del lenguaje. Estas restricciones permiten que AWS Lambda realice actividades operativas y administrativas en su nombre, como son el aprovisionamiento de capacidad, la monitorización del estado de la flota, la aplicación de parches de seguridad, la implementación del código, y la monitorización y el registro de las funciones de Lambda.

Si necesita administrar sus propios recursos informáticos, Amazon Web Services también ofrece otros servicios informáticos para satisfacer sus necesidades.

- El servicio Amazon Elastic Compute Cloud (Amazon EC2) ofrece flexibilidad y una amplia variedad de tipos de instancias de EC2 para elegir. Ofrece la posibilidad de personalizar los sistemas operativos, la pila completa de software y la configuración de red y de seguridad, pero el usuario es responsable de aprovisionar la capacidad, monitorizar el estado y el desempeño de la flota y utilizar las zonas de disponibilidad para aumentar la tolerancia a errores.

- Elastic Beanstalk ofrece un servicio fácil de utilizar para implementar y escalar aplicaciones en Amazon EC2, en el que se mantiene la propiedad y el control total sobre las instancias EC2 subyacentes.

¿Utiliza AWS Lambda por primera vez?

Si utiliza AWS Lambda por primera vez, le recomendamos que lea las siguientes secciones en orden:

1. Lea la información general del producto y vea el vídeo introductorio para conocer los ejemplos de casos de uso. Estos recursos están disponibles en la [página web de AWS Lambda](#).
Lea la sección “Funcionamiento” de esta guía. Esta sección presenta varios componentes de AWS Lambda con los que va a trabajar para crear una experiencia integral. Para obtener más información, consulte [Funcionamiento \(p. 188\)](#).
2. Lea la sección "Funciones de Lambda" de esta guía. Hay conceptos fundamentales con los que debe estar familiarizado para comprender el modelo de programación y las opciones de implementación para una función de Lambda. En esta sección se explican estos conceptos y se proporcionan detalles de cómo funcionan en los distintos lenguajes que puede utilizar para crear el código de una función de Lambda. Para obtener más información, consulte [Funciones de Lambda \(p. 3\)](#).
3. Realice el ejercicio de introducción basado en la consola. El ejercicio ofrece instrucciones para crear y probar su primera función de Lambda mediante la consola. También puede obtener más información sobre los proyectos que proporciona la consola para crear rápidamente funciones de Lambda. Para obtener más información, consulte [Introducción \(p. 199\)](#).
4. Lea la sección “Creación de aplicaciones con AWS Lambda” de esta guía. Esta sección presenta varios componentes de AWS Lambda con los que va a trabajar para crear una experiencia integral. Para obtener más información, consulte [Creación de aplicaciones con AWS Lambda \(p. 130\)](#).

Además del ejercicio de introducción, puede examinar distintos casos de uso, cada uno de los cuales dispone de un tutorial que presenta un escenario de ejemplo. Puede realizar tutoriales específicos acordes con las necesidades de su aplicación (por ejemplo, si desea que la invocación de la función de Lambda sea controlada por eventos o bajo demanda). Para obtener más información, consulte [Casos de uso \(p. 214\)](#).

Los siguientes temas proporcionan información adicional sobre AWS Lambda:

- [Control de versiones y alias de las funciones de AWS Lambda \(p. 80\)](#)
- [Solución de problemas y monitorización de funciones de AWS Lambda con Amazon CloudWatch \(p. 120\)](#)
- [Prácticas recomendadas para trabajar con funciones de AWS Lambda \(p. 331\)](#)
- [Límites de AWS Lambda \(p. 335\)](#)

Funciones de Lambda

Después de empaquetar el código personalizado, incluidas las dependencias, y cargarlo en AWS Lambda, habrá creado una función de Lambda.

Si es la primera vez que utiliza AWS Lambda, puede que le surjan las preguntas siguientes: ¿Qué tipo de código puedo ejecutar como función de Lambda? ¿Cómo ejecuta AWS Lambda mi código? ¿Cómo determina AWS Lambda la cantidad de memoria y los requisitos de CPU necesarios para ejecutar el código de una función de Lambda? En las secciones siguientes se incluye información general acerca de cómo funciona una función de Lambda.

Dependiendo de su situación, puede crear aplicaciones en las que utilice el servicio de AWS Lambda para ejecutar todo el código de la aplicación o parte de él. Para obtener más información, consulte [Creación de funciones de Lambda \(p. 5\)](#). [Funcionamiento \(p. 188\)](#) proporciona ejemplos que ilustran cómo crear una función de Lambda para situaciones específicas.

Las secciones de este tema proporcionan la siguiente información introductoria sobre las funciones de Lambda:

Temas

- [Requisitos informáticos: configuración de una función de Lambda \(p. 3\)](#)
- [Tipos de invocación \(p. 4\)](#)
- [Introducción: Creación de funciones de Lambda \(p. 5\)](#)
- [Configuración de una función de Lambda para acceder recursos en una Amazon VPC \(p. 108\)](#)
- [Solución de problemas y monitorización de funciones de AWS Lambda con Amazon CloudWatch \(p. 120\)](#)

Requisitos informáticos: configuración de una función de Lambda

Una función de Lambda incluye código y puede tener dependencias asociadas. Además, una función de Lambda también tiene asociada información de configuración. Inicialmente, debe especificar la información de configuración al crear una función de Lambda. Lambda proporciona una API que permite actualizar algunos de los datos de configuración. La información de configuración de una función de Lambda incluye los siguientes elementos fundamentales:

- **Recursos informáticos necesarios:** solo debe especificar la cantidad de memoria que desea asignar a la función de Lambda. AWS Lambda asigna potencia de CPU de acuerdo con la memoria, en la misma proporción que un tipo de instancia de Amazon EC2 de uso general, como el tipo M3. Por ejemplo, si se especifican 256 MB de memoria, la función de Lambda recibirá el doble de la capacidad de CPU que si se hubieran especificado 128 MB.

Es posible actualizar la configuración y solicitar más memoria en incrementos de 64 MB, desde 128 MB hasta 1 536 MB. Para obtener información sobre los límites correspondientes, consulte [Límites de AWS Lambda \(p. 335\)](#).

Para cambiar la cantidad de memoria que necesita una función de Lambda, haga lo siguiente:

1. Inicie sesión en la consola de administración de AWS y vaya a la consola de AWS Lambda.
2. Elija la función cuyo tamaño de memoria desea cambiar.
3. Haga clic en la pestaña Configuration y, a continuación, expanda Advanced settings.
4. En la lista Memory (MB), elija la cantidad deseada.

También puede actualizar el tamaño de memoria de las funciones utilizando el siguiente comando de la CLI de AWS (con incrementos válidos de 64 MB):

```
$ aws lambda update-function-configuration \
--function-name your function name \
--region region where your function resides \
--memory-size memory amount \
--profile adminuser
```

Para obtener información acerca de la configuración y el uso de la CLI de AWS, consulte [Paso 1: Configuración de una cuenta de AWS y de la AWS CLI \(p. 199\)](#).

- Tiempo máximo de ejecución (tiempo de espera): se paga por los recursos de AWS que se utilizan para ejecutar la función de Lambda. Para evitar que la función de Lambda se ejecute de forma indefinida, debe especificar un tiempo de espera. Cuando finaliza el tiempo de espera especificado, AWS Lambda termina la función de Lambda.

Important

Evite utilizar código recursivo en la función de Lambda que haga que esta llame a sí misma automáticamente hasta que se cumplan ciertos criterios arbitrarios. Esto podría producir un volumen no intencionado de invocaciones de la función y costos elevados.

- Rol de IAM (rol de ejecución): es el rol que AWS Lambda asume cuando ejecuta la función de Lambda en nombre del usuario.
- Nombre del controlador: el controlador hace referencia al método del código donde AWS Lambda comienza la ejecución. AWS Lambda pasa la información del evento que activó la invocación al método del controlador en forma de parámetro.

Tipos de invocación

AWS Lambda admite la invocación síncrona y asíncrona de funciones de Lambda. Solo es posible controlar el tipo de invocación cuando se invoca manualmente una función de Lambda (lo que se denomina invocación bajo demanda). Los siguientes ejemplos ilustran invocaciones bajo demanda:

- Una aplicación personalizada invoca una función de Lambda.
- Se invoca una función de Lambda manualmente (por ejemplo, mediante la AWS CLI) con fines de prueba.

En ambos casos, es posible invocar la función de Lambda utilizando la operación [Invoke \(p. 423\)](#) y se puede especificar el tipo de invocación síncrona o asíncrona.

Sin embargo, cuando se utilizan los servicios de AWS como orígenes de eventos, el tipo de invocación viene predeterminado para cada uno de estos servicios. No se tiene ningún control sobre el tipo de invocación que utilizan estos orígenes de eventos cuando invocan una función de Lambda. Por ejemplo, Amazon S3 siempre invoca una función de Lambda forma asíncrona y Amazon Cognito siempre invoca una función de Lambda de forma síncrona. En el caso de los servicios de AWS basados en flujos (Amazon Kinesis Streams y Flujos de Amazon DynamoDB), AWS Lambda sondea el flujo e invoca la función de Lambda de forma síncrona.

Introducción: Creación de funciones de Lambda

AWS Lambda es un servicio informático que puede ejecutar código en su nombre con solo cargar en él el código de una aplicación formado por una o varias funciones de Lambda. AWS Lambda se encarga de aprovisionar y administrar los servidores para ejecutar el código tras la invocación.

Normalmente, el ciclo de vida de una aplicación basada en AWS Lambda incluye la creación de código, la implementación del código en AWS Lambda y, por último, la monitorización y la solución de problemas. A continuación se muestran algunas preguntas generales que surgen en cada una de estas fases del ciclo de vida:

- Creación del código de la función de Lambda: ¿Qué lenguajes se admiten? ¿Existe un modelo de programación que debo seguir? ¿Cómo puedo empaquetar el código y las dependencias para cargarlos en AWS Lambda? ¿Qué herramientas están disponibles?
- Carga del código y creación de las funciones de Lambda: ¿Cómo cargo el paquete de código en AWS Lambda? ¿Cómo indico a AWS Lambda dónde debe empezar a ejecutar el código? ¿Cómo puedo especificar requisitos informáticos como la memoria y el tiempo de espera?
- Monitorización y solución de problemas: ¿Qué métricas están disponibles para una función de Lambda que está en producción? Si se producen errores, ¿cómo puedo obtener logs o solucionar problemas?

En las secciones siguientes se proporciona información introductoria, y la sección de ejemplos, situada al final, contiene ejemplos funcionales que puede examinar.

Note

En este tema se ofrece información general introductoria de cómo desarrollar aplicaciones basadas en AWS Lambda. En la sección [Funcionamiento \(p. 188\)](#) se describen los detalles de las funciones de Lambda, los orígenes de eventos y cómo AWS Lambda ejecuta las funciones de Lambda.

Creación del código de la función de Lambda

Puede escribir el código de la función de Lambda en los lenguajes admitidos por AWS Lambda. Para ver una lista de los lenguajes admitidos, consulte [Entorno de ejecución de Lambda y bibliotecas disponibles \(p. 195\)](#). Existen herramientas para la creación de código, como la consola de AWS Lambda el IDE de Eclipse y el IDE de Visual Studio. Sin embargo, las herramientas y las opciones disponibles dependen de lo siguiente:

- El lenguaje que elija para escribir el código de la función de Lambda.
- Las bibliotecas que utilice en el código. El tiempo de ejecución de AWS Lambda proporciona algunas de las bibliotecas, y el usuario debe cargar las bibliotecas adicionales que utilice.

En la siguiente tabla se muestran los lenguajes y las herramientas y opciones disponibles que puede utilizar.

| Lenguaje | Herramientas y opciones para crear código | Más información |
|----------|---|--|
| Node.js | <ul style="list-style-type: none">Consola de AWS LambdaVisual Studio, con el complemento del IDE (consulte AWS Lambda Support in Visual Studio)Su propio entorno de desarrollo | Puede utilizar la consola si los lenguajes que elija no requieren compilación, el código se guarda en un solo archivo y no depende de ninguna biblioteca. |
| Java | <ul style="list-style-type: none">Eclipse, con AWS Toolkit for Eclipse (consulte Uso de AWS Lambda con AWS Toolkit for Eclipse)Su propio entorno de desarrollo | AWS Toolkit también crea el paquete de implementación, que se explica en Implementación de código y creación de una función de Lambda (p. 6) . |
| C# | <ul style="list-style-type: none">Visual Studio, con el complemento del IDE (consulte AWS Lambda Support in Visual Studio).NET Core (consulte la guía de instalación de .NET Core)Su propio entorno de desarrollo | AWS Toolkit también crea el paquete de implementación, que se explica en Implementación de código y creación de una función de Lambda (p. 6) . |
| Python | <ul style="list-style-type: none">Consola de AWS LambdaSu propio entorno de desarrollo | Puede utilizar la consola si los lenguajes que elija no requieren compilación, el código se guarda en un solo archivo y no depende de ninguna biblioteca. |

Además, independientemente del lenguaje que elija, existe un patrón para escribir el código de una función de Lambda. Por ejemplo, cómo escribir el método del controlador de la función de Lambda (es decir, el primer método al que AWS Lambda llama cuando comienza a ejecutar el código), cómo se pasan los eventos al controlador, las instrucciones que se pueden utilizar en el código para generar logs en CloudWatch Logs, cómo interactuar con el tiempo de ejecución de AWS Lambda y cómo obtener información, como por ejemplo, el tiempo que queda hasta que se agote el tiempo de espera y cómo controlar las excepciones. La sección [Modelo de programación \(p. 8\)](#) proporciona información para cada uno de los lenguajes admitidos.

Note

Cuando se familiarice con AWS Lambda, consulte los [Casos de uso \(p. 214\)](#), que proporcionan instrucciones paso a paso para ayudarle a explorar la experiencia integral.

Implementación de código y creación de una función de Lambda

Para crear una función de Lambda, primero tiene que empaquetar el código y las dependencias en un paquete de implementación. A continuación, cargue el paquete de implementación en AWS Lambda para crear la función de Lambda.

Temas

- [Creación de un paquete de implementación: organización del código y las dependencias \(p. 7\)](#)
- [Carga de un paquete de implementación: creación de una función de Lambda \(p. 7\)](#)

- Prueba de una función de Lambda (p. 7)

Creación de un paquete de implementación: organización del código y las dependencias

En primer lugar, debe organizar el código y las dependencias adecuadamente y crear un paquete de implementación. Las instrucciones para crear un paquete de implementación dependen del lenguaje elegido para escribir el código. Por ejemplo, puede utilizar complementos como Jenkins (para Node.js y Python) y Maven (para Java) para crear los paquetes de implementación. Para obtener más información, consulte [Creación de un paquete de implementación \(p. 64\)](#).

Si crea funciones de Lambda mediante la consola, esta crea el paquete de implementación automáticamente y, a continuación, lo carga para crear la función de Lambda.

Carga de un paquete de implementación: creación de una función de Lambda

AWS Lambda proporciona la operación [CreateFunction \(p. 385\)](#), que se utiliza para crear una función de Lambda. Puede utilizar la consola de AWS Lambda, la AWS CLI y los SDK de AWS para crear una función de Lambda. Internamente, todas estas interfaces llaman a la operación [CreateFunction](#).

Además de proporcionar el paquete de implementación, puede proporcionar la información de configuración al crear la función de Lambda, incluidos los requisitos informáticos de la función de Lambda, el nombre del método del controlador de la función de Lambda y el tiempo de ejecución, que depende del lenguaje elegido para escribir el código. Para obtener más información, consulte [Funciones de Lambda \(p. 3\)](#).

Note

En esta sección se ofrece información general introductoria sobre cómo desarrollar aplicaciones basadas en AWS Lambda. En la sección [Funcionamiento \(p. 188\)](#) se describen los detalles de las funciones de Lambda, los orígenes de eventos y cómo AWS Lambda ejecuta las funciones de Lambda.

Prueba de una función de Lambda

Si la función de Lambda está diseñada para procesar eventos de un tipo determinado, puede utilizar los datos del evento de muestra para probar la función de Lambda utilizando uno de los siguientes métodos:

- Probar la función de Lambda en la consola.
- Probar la función de Lambda mediante la AWS CLI. Puede utilizar el método `Invoke` para invocar la función de Lambda y pasarle los datos del evento de muestra.

La consola proporciona los datos del evento de muestra. Los mismos datos también están disponibles en el tema [Eventos de muestra publicados por los orígenes de eventos \(p. 146\)](#), y puede utilizarlos en la AWS CLI para invocar la función de Lambda.

Monitorización y solución de problemas

Cuando la función de Lambda está en producción, AWS Lambda la monitoriza automáticamente en su nombre, e informa sobre las métricas en Amazon CloudWatch. Para obtener más información, consulte [Acceso a las métricas de Amazon CloudWatch para AWS Lambda \(p. 122\)](#).

Para ayudarle a solucionar problemas relacionados con los errores de una función, Lambda registra todas las solicitudes gestionadas por la función, y almacena automáticamente los logs generados por el código

en Amazon CloudWatch Logs. Para obtener más información, consulte [Acceso a los logs de Amazon CloudWatch para AWS Lambda \(p. 124\)](#).

Ejemplos de aplicaciones basadas en AWS Lambda

Esta guía proporciona varios ejemplos con instrucciones paso a paso. Si es la primera vez que utiliza AWS Lambda, le recomendamos que pruebe los siguientes ejercicios:

- [Introducción \(p. 199\)](#): el ejercicio de introducción proporciona una experiencia basada en la consola. El código de muestra está escrito en Python. Puede editar el código en la consola, cargarlo en AWS Lambda y probarlo con los datos del evento de muestra proporcionados en la consola.
- [Casos de uso \(p. 214\)](#): si no sabe escribir el código mediante la consola, debe crear sus propios paquetes de implementación y utilizar la AWS CLI (o los SDK) para crear la función de Lambda. Para obtener más información, consulte [Creación del código de la función de Lambda \(p. 5\)](#). La mayoría de los ejemplos de la sección Casos de uso utilizan la AWS CLI. Si es la primera vez que utiliza AWS Lambda, le recomendamos que pruebe uno de estos ejercicios.

Temas relacionados

Los siguientes temas aportan información adicional.

[Modelo de programación \(p. 8\)](#)

[Creación de un paquete de implementación \(p. 64\)](#)

[Control de versiones y alias de las funciones de AWS Lambda \(p. 80\)](#)

[Solución de problemas y monitorización de funciones de AWS Lambda con Amazon CloudWatch \(p. 120\)](#)

Modelo de programación

Puede escribir código para la función de Lambda en cualquiera de los lenguajes que admite AWS Lambda. Independientemente del lenguaje que elija, hay un patrón común para escribir código para una función de Lambda que incluye los siguientes conceptos clave:

- Controlador: el controlador es la función que AWS Lambda llama para iniciar la ejecución de la función de Lambda. El controlador se identifica al crear la función de Lambda. Cuando se invoca una función de Lambda, AWS Lambda comienza a ejecutar el código llamando a la función de controlador. AWS Lambda pasa los datos de eventos a este controlador como primer parámetro. El controlador debe procesar los datos de eventos entrantes y puede invocar otras funciones o métodos del código.
- El objeto context y cómo interactúa con Lambda durante el tiempo de ejecución: AWS Lambda también pasa un objeto “de contexto” a la función de controlador como segundo parámetro. A través de este objeto context, el código puede interactuar con AWS Lambda. Por ejemplo, el código puede encontrar el tiempo que falta para que AWS Lambda finalice la ejecución de la función de Lambda.

Además, para lenguajes como Node.js, hay una plataforma asíncrona que utiliza devoluciones de llamada. AWS Lambda proporciona métodos adicionales en este objeto context. Puede utilizar estos métodos del objeto context para indicar a AWS Lambda que termine la función de Lambda y, opcionalmente, devuelva valores al intermediario.

- Registro: la función de Lambda puede contener instrucciones de registro. AWS Lambda escribe estos logs en CloudWatch Logs. Las instrucciones de lenguaje específicas generan entradas de log que dependen del lenguaje que se utilice para crear el código de la función de Lambda.
- Excepciones: la función de Lambda debe comunicar el resultado de la ejecución de la función a AWS Lambda. Según el lenguaje que se utilice para crear el código de la función de Lambda, existen diferentes maneras de finalizar una solicitud correctamente o de notificar a AWS Lambda que se ha producido un error durante la ejecución. Si se invoca la función de forma síncrona, AWS Lambda reenvía el resultado de vuelta al cliente.

Note

El código de la función de Lambda debe escribirse sin estado y no debe tener afinidad con la infraestructura informática subyacente. El código debe prever que el acceso al sistema de archivos local, los procesos secundarios y los artefactos similares estarán limitados a la duración de la solicitud. El estado persistente debe almacenarse en Amazon S3, Amazon DynamoDB u otro servicio de almacenamiento en la nube. La obligatoriedad de utilizar funciones sin estado permite que AWS Lambda lance tantas copias de la función como sean necesarias para escalar de acuerdo con la velocidad de los eventos y las solicitudes entrantes. Es posible que estas funciones no siempre se ejecuten en la misma instancia informática con cada solicitud. Una instancia determinada de la función de Lambda puede ser utilizada más de una vez por AWS Lambda.

Los siguientes temas específicos del lenguaje proporcionan información detallada:

- [Modelo de programación \(Node.js\) \(p. 9\)](#)
- [Modelo de programación para crear funciones de Lambda en Java \(p. 25\)](#)
- [Modelo de programación para crear funciones de Lambda en C# \(p. 53\)](#)
- [Modelo de programación para crear funciones de Lambda en Python \(p. 44\)](#)

Modelo de programación (Node.js)

Actualmente, AWS Lambda admite los siguientes tiempos de ejecución de Node.js:

- Tiempo de ejecución v6.10 de Node.js (tiempo de ejecución = nodejs6.10)
- Tiempo de ejecución v4.3 de Node.js (tiempo de ejecución = nodejs4.3)
- Tiempo de ejecución v0.10.42 de Node.js (tiempo de ejecución = nodejs)

Important

Actualmente, Node v0.10.42 está marcado como obsoleto. Debe migrar las funciones existentes a las versiones del tiempo de ejecución de Node.js más recientes disponibles en AWS Lambda (nodejs4.3 o nodejs6.10) tan pronto como sea posible. Puede solicitar una única extensión hasta el 30 de junio de 2017 accediendo a la consola de Lambda y siguiendo las instrucciones proporcionadas. Si no se realiza la migración ni se obtiene una extensión, las invocaciones de las funciones escritas en el tiempo de ejecución v0.10.42 de Node devolverán un error de valor de parámetro no válido. Tenga en cuenta que deberá seguir este procedimiento para cada región que contenga funciones escritas en el tiempo de ejecución v0.10.42 de Node. Para obtener información acerca de las diferencias del modelo de programación en el tiempo de ejecución v0.10.42, consulte [Utilización del tiempo de ejecución anterior v0.10.42 de Node.js \(p. 20\)](#).

Al crear una función de Lambda, puede especificar el tiempo de ejecución que desea utilizar. Para obtener más información, consulte el parámetro `runtime` de [CreateFunction \(p. 385\)](#).

En las secciones siguientes se explica cómo se aplican los [conceptos fundamentales y los patrones de programación comunes](#) al crear código de funciones de Lambda en Node.js. El modelo de programación descrito en las siguientes secciones se aplica a ambas versiones, excepto donde se indique.

Temas

- [Controlador de la función de Lambda \(Node.js\) \(p. 10\)](#)
- [El objeto context \(Node.js\) \(p. 12\)](#)
- [Registro \(Node.js\) \(p. 15\)](#)
- [Errores de funciones \(Node.js\) \(p. 17\)](#)
- [Utilización del tiempo de ejecución anterior v0.10.42 de Node.js \(p. 20\)](#)

Controlador de la función de Lambda (Node.js)

En el momento de crear una función de Lambda, debe especificar un controlador, que es una función en el código, que AWS Lambda puede invocar cuando el servicio ejecuta el código. Utilice la siguiente sintaxis general al crear una función de controlador en Node.js.

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

El parámetro de devolución de llamada es opcional, y depende de si desea devolver información al intermediario.

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

En la sintaxis, tenga en cuenta lo siguiente:

- **event**: AWS Lambda utiliza este parámetro para pasar datos de eventos al controlador.
- **context**: AWS Lambda utiliza este parámetro para proporcionar al controlador la información del tiempo de ejecución de la función de Lambda que se está ejecutando. Para obtener más información, consulte [El objeto context \(Node.js\) \(p. 12\)](#).
- **callback**: puede utilizar la devolución de llamada opcional para devolver información al intermediario. De lo contrario, el valor de retorno es nulo. Para obtener más información, consulte [Uso del parámetro callback \(p. 11\)](#).

Note

La devolución de llamada se admite únicamente en los tiempos de ejecución v6.10 y v4.3 de Node.js. Si utiliza el tiempo de ejecución v0.10.42, tendrá que utilizar los métodos del objeto context (`done`, `succeed` y `fail`) para terminar la función de Lambda correctamente.

Para obtener información, consulte [Utilización del tiempo de ejecución anterior v0.10.42 de Node.js \(p. 20\)](#).

- **myHandler**: este es el nombre de la función que invoca AWS Lambda. Debe exportarlo para que sea visible para AWS Lambda. Suponga que guarda el código como `helloworld.js`. Entonces, `helloworld.myHandler` es el controlador. Para obtener más información, consulte la sección Controlador en [CreateFunction \(p. 385\)](#).
- Si utiliza el tipo de invocación `RequestResponse` (ejecución síncrona), AWS Lambda devuelve el resultado de la llamada a la función de Node.js al cliente que invoca la función de Lambda (en la respuesta HTTP a la solicitud de invocación, serializada en JSON). Por ejemplo, la consola de AWS

Lambda utiliza el tipo de invocación `RequestResponse`, por lo que al invocar la función utilizando la consola, esta mostrará el valor de retorno.

Si el controlador no devuelve nada, AWS Lambda devuelve null.

- Si se utiliza el tipo de invocación `Event` (ejecución asíncrona), el valor se descarta.

Ejemplo

Observe el siguiente código de ejemplo de Node.js.

```
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");  
}
```

Este ejemplo tiene una función, que también es el controlador. En la función, las instrucciones `console.log()` registran algunos de los datos de eventos entrantes en CloudWatch Logs. Cuando se llama a la devolución de llamada, la función de Lambda solo finaliza después de que el bucle de eventos de Node.js esté vacío (el bucle de eventos de Node.js no es el mismo que el evento que se ha pasado como parámetro).

Note

Si utiliza el tiempo de ejecución v0.10.42, debe utilizar los métodos del objeto `context` (`done`, `succeed` y `fail`) para terminar la función de Lambda correctamente. Para obtener más información, consulte [Utilización del tiempo de ejecución anterior v0.10.42 de Node.js \(p. 20\)](#).

Para cargar y probar este código como una función de Lambda (consola)

1. En la consola, cree una función de Lambda utilizando la siguiente información:

- Utilice el proyecto `hello-world`.
- Recomendamos especificar `nodejs6.10` como el tiempo de ejecución, pero también puede seleccionar `nodejs4.3`. El código de muestra proporcionado funciona con cualquiera de las versiones.
- En Handler, sustituya `index.handler` por `exports.myHandler`.

Para obtener instrucciones acerca de cómo crear una función de Lambda utilizando la consola, consulte [Paso 2.1: Creación de una función Hello World de Lambda \(p. 203\)](#).

2. Sustituya el código de la plantilla por el código proporcionado en esta sección y cree la función.
3. Pruebe la función de Lambda utilizando la plantilla Sample event template denominada Hello World proporcionada en la consola de Lambda. Para obtener instrucciones al respecto, consulte [Paso 2.2: Invocación manual de la función de Lambda y verificación de los resultados, los logs y las métricas \(p. 206\)](#).

Uso del parámetro callback

Los tiempos de ejecución v4.3 y v6.10 de Node.js admiten el parámetro opcional `callback`. Puede utilizarlo para devolver información de forma explícita al intermediario. La sintaxis general es:

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

Donde:

- `error`: es un parámetro opcional que puede utilizar para proporcionar los resultados de la ejecución con errores de la función de Lambda. Cuando una función de Lambda se ejecuta satisfactoriamente, puede pasar `null` como primer parámetro.
- `result`: es un parámetro opcional que puede utilizar para proporcionar el resultado de una ejecución satisfactoria de la función. El resultado proporcionado debe ser compatible con `JSON.stringify`. Si se proporciona un error, este parámetro se ignora.

Note

El uso del parámetro `callback` es opcional. Si no utiliza el parámetro opcional `callback`, el comportamiento es el mismo que al llamar a `callback()` sin ningún parámetro. Puede especificar `callback` en el código para devolver información al intermediario.

Si no utiliza `callback` en el código, AWS Lambda lo llamará implícitamente y el valor de retorno será `null`.

Cuando se llama a la devolución de llamada (de forma explícita o implícita), AWS Lambda continúa la invocación de la función de Lambda hasta que el bucle de eventos de Node.js esté vacío.

A continuación se muestran devoluciones de llamada de ejemplo:

```
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 trata cualquier valor distinto de `null` para el parámetro `error` como una excepción gestionada.

Tenga en cuenta lo siguiente:

- Independientemente del tipo de invocación especificado en el momento de la invocación de la función de Lambda (consulte [Invoke \(p. 423\)](#)), el método de devolución de llamada registra automáticamente la representación de cadena de valores de `error` distintos de `null` en el flujo de Amazon CloudWatch Logs asociado a la función de Lambda.
- Si la función de Lambda se invocó de forma síncrona (utilizando el tipo de invocación `RequestResponse`), la devolución de llamada devolverá un cuerpo de la respuesta como se indica a continuación:
 - Si `error` es `null`, el cuerpo de la respuesta se establece en la representación de cadena de `result`.
 - Si `error` es distinto de `null`, el cuerpo de la respuesta contendrá el valor de `error`.

Note

Cuando se llame a `callback(error, null)` (y `callback(error)`), Lambda registrará los primeros 256 KB del objeto de error. Para un objeto de error más grande, AWS Lambda trunca el log y muestra el texto `Truncated by Lambda` junto al objeto de error.

El objeto context (Node.js)

Mientras se está ejecutando una función de Lambda, es posible interactuar con AWS Lambda para obtener información útil sobre el tiempo de ejecución, como por ejemplo:

- Cantidad de tiempo restante para que AWS Lambda termine la función de Lambda (el tiempo de espera es una de las propiedades de configuración de las funciones de Lambda).
- El grupo de logs y el flujo de logs de CloudWatch asociados a la función de Lambda que se está ejecutando.

- El ID de la solicitud de AWS devuelto al cliente que invocó la función de Lambda. Puede utilizar el ID de la solicitud para cualquier consulta de seguimiento con AWS Support.
- Si se invoca la función de Lambda a través de AWS Mobile SDK, se puede obtener más información sobre la aplicación móvil llamando a la función de Lambda.

AWS Lambda proporciona esta información a través del objeto `context`, que el servicio pasa como segundo parámetro al controlador de la función de Lambda. Para obtener más información, consulte [Controlador de la función de Lambda \(Node.js\) \(p. 10\)](#).

Las siguientes secciones proporcionan una función de Lambda de ejemplo que utiliza el objeto `context` y, a continuación, enumera todos los métodos y atributos disponibles.

Ejemplo

Observe el siguiente ejemplo de Node.js. El controlador recibe información del tiempo de ejecución a través un parámetro de `context`.

```
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");
};
```

El código del controlador de este ejemplo registra parte de la información del tiempo de ejecución de la función de Lambda en CloudWatch. Si invoca la función utilizando la consola de Lambda, esta mostrará los logs en la sección Log output. Puede crear una función de Lambda utilizando este código y probarla mediante la consola.

Para probar este código en la consola de AWS Lambda

1. En la consola, cree una función de Lambda mediante el proyecto hello-world. En runtime, elija nodejs6.10. Para obtener instrucciones al respecto, consulte [Paso 2.1: Creación de una función Hello World de Lambda \(p. 203\)](#).
2. Pruebe la función. También puede actualizar el código para obtener más información de contexto.

Métodos del objeto `context` (Node.js)

El objeto `context` proporciona los siguientes métodos.

`context.getRemainingTimeInMillis()`

Devuelve el tiempo aproximado que falta para terminar la ejecución (antes de que se agote el tiempo de espera) de la función de Lambda que se está ejecutando. El tiempo de espera es uno de los parámetros de

configuración de la función de Lambda. Cuando se alcanza el tiempo de espera, AWS Lambda termina la función de Lambda.

Puede utilizar este método para comprobar el tiempo restante en la ejecución de la función y adoptar las medidas correctivas necesarias durante el tiempo de ejecución.

La sintaxis general es:

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

Propiedades del objeto context (Node.js)

El objeto `context` proporciona las siguientes propiedades que se pueden actualizar:

`callbackWaitsForEmptyEventLoop`

El valor predeterminado es `true`. Esta propiedad es útil solo para modificar el comportamiento predeterminado de la devolución de llamada. De forma predeterminada, la devolución de llamada esperará hasta que el bucle de eventos del tiempo de ejecución de Node.js esté vacío antes de congelar el proceso y devolver los resultados al intermediario. Puede establecer esta propiedad en `false` para solicitar a AWS Lambda que congele el proceso poco después de que se llame a `callback`, incluso si hay eventos en el bucle de eventos. AWS Lambda congelará el proceso, los datos de estado y los eventos en el bucle de eventos de Node.js (cualquier evento restante en el bucle de eventos procesado cuando se llame a la función de Lambda la próxima vez y si AWS Lambda elige utilizar el proceso congelado). Para obtener más información acerca de la devolución de llamada, consulte [Uso del parámetro callback \(p. 11\)](#).

Además, el objeto `context` proporciona las siguientes propiedades que puede utilizar para obtener información del tiempo de ejecución:

`functionName`

Nombre de la función de Lambda que se está ejecutando.

`functionVersion`

La versión de la función de Lambda que se está ejecutando. Si se utiliza un alias para invocar la función, `function_version` será la versión a la que apunta el alias.

`invokedFunctionArn`

El ARN utilizado para invocar esta función. Puede ser el ARN de la función o el ARN de un alias. Un ARN incompleto ejecuta la versión `$LATEST` y los alias ejecutan la versión de la función a la que apunta.

`memoryLimitInMB`

Cantidad máxima de memoria, en MB, configurada para la función de Lambda. El límite de memoria se establece en el momento de crear una función de Lambda, pero puede cambiarse más adelante.

`awsRequestId`

El ID de la solicitud de AWS asociado a la solicitud. Este es el ID que se devuelve al cliente que llamó al método `invoke`.

Note

Si AWS Lambda reintenta la invocación (por ejemplo, en una situación en la que la función de Lambda que está procesando registros de Kinesis genera una excepción), el ID de la solicitud sigue siendo el mismo.

logGroupName

El nombre del grupo de logs de CloudWatch donde se encuentran los logs escritos por la función de Lambda.

logStreamName

El nombre del grupo de logs de CloudWatch donde se encuentran los logs escritos por la función de Lambda. El flujo de logs puede cambiar o no para cada invocación de la función de Lambda.

El valor es null si la función de Lambda no puede crear un flujo de logs, lo que puede ocurrir si el rol de ejecución que concede los permisos necesarios a la función de Lambda no incluye permisos para las acciones de CloudWatch.

identity

Información acerca del proveedor de identidad de Amazon Cognito cuando se invoca a través de AWS Mobile SDK. Puede ser null.

- identity.cognitoIdentityId
- identity.cognitoIdentityPoolId

Para obtener más información acerca de los valores exactos para una plataforma de móviles específica, consulte [Contexto de identidad](#) en la AWS Mobile SDK for iOS Developer Guide y [Contexto de identidad](#) en la SDK de AWS Mobile para Android Developer Guide.

clientContext

Información acerca del dispositivo y la aplicación cliente cuando se invocan a través de AWS Mobile SDK. Puede ser null. Con `clientContext` puede obtener la siguiente información:

- 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

Valores personalizados establecidos por la aplicación cliente para móviles.

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

Para obtener más información acerca de los valores exactos para una plataforma de móviles específica, consulte [Contexto de cliente](#) en la AWS Mobile SDK for iOS Developer Guide y [Contexto de cliente](#) en la SDK de AWS Mobile para Android Developer Guide.

Registro (Node.js)

La función de Lambda puede contener instrucciones de registro. AWS Lambda escribe estos logs en CloudWatch. Si utiliza la consola de Lambda para invocar la función de Lambda, la consola muestra los mismos logs.

Las siguientes instrucciones de Node.js generan entradas de log:

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

- `console.info()`

Por ejemplo, fíjese en el siguiente código de ejemplo de Node.js.

```
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
};


```

La captura de pantalla muestra un ejemplo de la sección Log output de la consola de Lambda. También puede encontrar estos logs en CloudWatch. Para obtener más información, consulte [Acceso a los logs de Amazon CloudWatch para AWS Lambda \(p. 124\)](#).

The screenshot shows the AWS Lambda execution result page. At the top, there is a green checkmark icon and the text "Execution result: succeeded (logs)". Below this, there is a "Details" section with the sub-section "Logs". A note states: "The area below shows the result returned by your function execution. [Learn more](#) about returning results from your function." A text box contains the string "value1". The "Summary" section includes fields for "Code SHA-256", "Request ID", "Duration", "Billed duration", and "Resources configured". The "Log output" section displays the following log entries:

| Timestamp | RequestId | Log Message |
|--------------------------|--|---|
| 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 |
| | REPORT RequestId: 9c454e0d-6d78-11e7-b93a-439d5c9ba65d | Duration: 472.47 ms Billed Duration: 500 ms Memory Size: 128 MB Max Memory Used: 18 MB |

La consola utiliza el tipo de invocación `RequestResponse` (invocación síncrona) cuando se invoca la función y, por lo tanto, obtiene el valor de retorno (`value1`) de AWS Lambda, que se muestra en la consola.

Para probar el código de Node.js anterior en la consola de AWS Lambda

1. En la consola, cree una función de Lambda mediante el proyecto hello-world. Asegúrese de seleccionar Node.js como tiempo de ejecución. Para obtener instrucciones al respecto, consulte [Paso 2.1: Creación de una función Hello World de Lambda \(p. 203\)](#).
2. Pruebe la función de Lambda utilizando la plantilla Sample event template denominada Hello World proporcionada en la consola de Lambda. Para obtener instrucciones al respecto, consulte

[Paso 2.2: Invocación manual de la función de Lambda y verificación de los resultados, los logs y las métricas \(p. 206\)](#). También puede actualizar el código y probar otros métodos y propiedades de registro que se describen en esta sección.

Para obtener instrucciones paso a paso, consulte [Introducción \(p. 199\)](#).

Dónde encontrar los logs

Puede encontrar los logs que escribe la función de Lambda como se indica a continuación:

- En la consola de AWS Lambda: la sección Log output de la consola de AWS Lambda muestra los logs.
- En el encabezado de respuesta, cuando se invoca una función de Lambda mediante programación: si se invoca una función de Lambda mediante programación, se puede añadir el parámetro `LogType` para recuperar los últimos 4 KB de datos de log escritos en CloudWatch Logs. AWS Lambda devuelve esta información de log en el encabezado `x-amz-log-results` de la respuesta. Para obtener más información, consulte [Invoke](#).

Si utiliza la CLI de AWS para invocar la función, puede especificar el parámetro `--log-type parameter` con el valor `Tail` para recuperar la misma información.

- En CloudWatch Logs: para encontrar los logs en CloudWatch, necesita conocer el nombre del grupo de logs y el nombre del flujo de logs. Puede obtener dicha información añadiendo los métodos `context.logGroupName` y `context.logStreamName` en el código. Cuando ejecute la función de Lambda, los logs resultantes en la consola o en la CLI le mostrarán el nombre del grupo de logs y el nombre del flujo de logs.

Errores de funciones (Node.js)

Si la función de Lambda notifica a AWS Lambda que no se ha ejecutado correctamente, Lambda intentará convertir el objeto de error en una cadena. Considere el siguiente ejemplo:

```
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);
};
```

Al invocar la función de Lambda, esta notificará a AWS Lambda que la ejecución de la función se completó con un error y pasará la información de error a AWS Lambda. AWS Lambda devuelve la información del error al cliente:

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

Observe que el rastro de stack se devuelve como la matriz JSON `stackTrace` de elementos de rastro de stack.

La forma de obtener la información de error depende del tipo de invocación que el cliente especifique en el momento de invocar la función:

- Si un cliente especifica el tipo de invocación RequestResponse (es decir, ejecución síncrona), se devuelve el resultado al cliente que ha realizado la llamada de invocación.

Por ejemplo, la consola siempre utiliza el tipo de invocación RequestResponse, por lo que mostrará el error en la sección Execution result, tal como se muestra a continuación:

The screenshot shows a red-bordered alert box with the title "Execution result: failed (logs)". Below it is a "Details" section with the text: "The area below shows the result returned by your function execution. [Learn more](#) about returning results from your function." A code block displays the following JSON response:

```
{  
  "errorMessage": "Something went wrong"  
}
```

La misma información también se envía a CloudWatch y la sección Log output muestra los mismos logs.

The screenshot shows the "Log output" section of the Lambda function configuration. It contains a summary of log entries:

| Time | Request ID | Log Message |
|--------------------------|--|--|
| 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 |

- Si un cliente especifica el tipo de invocación Event (es decir, ejecución asíncrona), AWS Lambda no devolverá nada. En cambio, registrará la información de error en CloudWatch Logs. También puede consultar las métricas de errores en CloudWatch Metrics.

En función del origen de eventos, AWS Lambda puede volver a intentar la ejecución de la función de Lambda que ha fallado. Por ejemplo, si Kinesis es el origen de eventos, AWS Lambda reintentará la invocación que produce errores hasta que la función de Lambda se ejecute correctamente o hasta que los registros del flujo caduquen. Para obtener más información acerca de los reintentos, consulte [Reintentos si se producen errores \(p. 192\)](#).

Para probar el código de Node.js anterior (consola)

- En la consola, cree una función de Lambda mediante el proyecto hello-world. En runtime, elija Node.js y en Role, elija Basic execution role. Para obtener instrucciones al respecto, consulte [Paso 2.1: Creación de una función Hello World de Lambda \(p. 203\)](#).
- Sustituya el código de la plantilla por el código proporcionado en esta sección.
- Pruebe la función de Lambda utilizando la plantilla Sample event template denominada Hello World proporcionada en la consola de Lambda. Para obtener instrucciones al respecto, consulte [Paso 2.2: Invocación manual de la función de Lambda y verificación de los resultados, los logs y las métricas \(p. 206\)](#).

Control de errores de las funciones

Puede crear un control de errores personalizado para generar una excepción directamente desde la función de Lambda y gestionarla directamente (Retry o Catch) en una máquina de estado de AWS Step Functions. Para obtener más información, consulte [Administración de las condiciones de error con una máquina de estado](#).

Considere que un `estado CreateAccount` es una `tarea` que escribe los datos de un cliente en una base de datos utilizando una función de Lambda.

- Si la tarea se realiza correctamente, se crea una cuenta y se envía un correo electrónico de bienvenida.
- Si un usuario intenta crear una cuenta para un nombre de usuario que ya existe, la función de Lambda genera un error, lo que hace que la máquina de estado sugiera otro nombre de usuario y vuelva a intentar el proceso de creación de la cuenta.

En los siguientes códigos de muestra se ilustra cómo hacerlo. Tenga en cuenta que los errores personalizados en Node.js deben extender el prototipo de error.

```
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);
};
```

Puede configurar Step Functions para que detecte el error utilizando una regla `Catch`:

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

En tiempo de ejecución, AWS Step Functions detecta el error y realiza una [transición](#) al estado `SuggestAccountName`, según se especifica en la transición `Next`.

Note

La propiedad de nombre del objeto `Error` debe coincidir con el valor de `ErrorEquals`.

El control de errores personalizado facilita la creación de aplicaciones [sin servidor](#). Esta característica se integra con todos los lenguajes admitidos por el [Modelo de programación \(p. 8\)](#) de Lambda, lo que le permite diseñar la aplicación en los lenguajes de programación que elija, combinándolos a medida que avanza.

Para obtener más información acerca de cómo crear sus propias aplicaciones sin servidor utilizando AWS Step Functions y AWS Lambda, consulte [AWS Step Functions](#).

Utilización del tiempo de ejecución anterior v0.10.42 de Node.js

A fecha de mayo de 2017, AWS Lambda es compatible con Node.js 6.10 y Node.js 4.3. Para obtener información acerca cómo especificar este tiempo de ejecución al crear la función de Lambda, consulte el parámetro `--runtime` de [CreateFunction \(p. 385\)](#).

Actualmente, Node v0.10.42 está marcado como obsoleto. Debe migrar las funciones existentes a las versiones del tiempo de ejecución de Node.js más recientes disponibles en AWS Lambda (nodejs4.3 o nodejs6.10) tan pronto como sea posible. Puede solicitar una única extensión hasta el 30 de junio de 2017 accediendo a la consola de Lambda y siguiendo las instrucciones proporcionadas. Si no se realiza la migración ni se obtiene una extensión, las invocaciones de las funciones escritas en Node v0.10.42 devolverán un error de valor de parámetro no válido. Tenga en cuenta que deberá seguir este procedimiento para cada región que contenga funciones escritas en el tiempo de ejecución v0.10.42 de Node. La siguiente sección resalta la política de soporte del tiempo de ejecución de AWS Lambda, junto con el comportamiento exclusivo del tiempo de ejecución v0.10.42 y cómo migrar las funciones existentes a versiones más recientes.

Temas

- [Política de soporte del tiempo de ejecución \(p. 20\)](#)
- [Transición del código de funciones de Lambda a tiempos de ejecución más recientes \(p. 20\)](#)
- [Los métodos del objeto context en el tiempo de ejecución v0.10.42 de Node.js \(p. 22\)](#)

Política de soporte del tiempo de ejecución

AWS Lambda solo descartará los tiempos de ejecución que estén marcados como EOL (fin de vida útil) al final de su periodo de mantenimiento, tal como se especifica en la página [Node LTS working group page](#). Las versiones que están marcadas como EOL (como Node 0.10) dejarán de dar soporte primero a la creación de funciones nuevas. Las funciones existentes continuarán funcionando hasta que los clientes tengan tiempo suficiente para migrar a versiones más nuevas. Trabajaremos con cada cliente según sea necesario. AWS Lambda añadirá soporte para versiones LTS (soporte a largo plazo) de Node adicionales unos meses después de que la versión se marque como LTS.

Transición del código de funciones de Lambda a tiempos de ejecución más recientes

Actualmente, Node v0.10.42 está marcado como obsoleto. Debe migrar las funciones existentes a las versiones del tiempo de ejecución de Node.js más recientes disponibles en AWS Lambda (nodejs4.3 o nodejs6.10) tan pronto como sea posible. Puede solicitar una única extensión hasta el 30 de junio de 2017 accediendo a la consola de Lambda y siguiendo las instrucciones proporcionadas. Si no se realiza la migración ni se obtiene una extensión, las invocaciones de las funciones escritas en el tiempo de ejecución v0.10.42 de Node devolverán un error de valor de parámetro no válido. Tenga en cuenta que deberá seguir este procedimiento para cada región que contenga funciones escritas en el tiempo de ejecución v0.10.42 de Node.

En las secciones siguientes se explica cómo migrar el código de las funciones de Lambda existentes a tiempos de ejecución más recientes:

1. Revise todas sus funciones de Lambda y planifique la migración. Puede obtener la lista de funciones, junto con sus versiones y alias, como se indica a continuación:

Para mostrar una lista de las funciones de Lambda utilizando un proyecto, consulte [Listado de funciones de Lambda y actualización a un tiempo de ejecución más reciente utilizando el proyecto de actualización del tiempo de ejecución \(p. 22\)](#).

Para mostrar una lista de las funciones de Lambda utilizando la consola:

- a. Inicie sesión en la Consola de administración de AWS y abra la consola de Lambda.
 - b. Elija la columna Runtime. Esto ordenará todas las funciones de Lambda para esa región según el valor del tiempo de ejecución.
 - c. Abra cada función de Lambda cuyo valor de tiempo de ejecución sea node.js y, a continuación, elija la pestaña Configuration.
 - d. Elija la lista desplegable Qualifiers.
 - e. Seleccione cada versión y consulte el tiempo de ejecución.
 - f. Seleccione cada alias para ver la versión a la que apunta.
 - g. Repita los pasos anteriores para cada región según sea necesario.
2. Para cada función:
 - a. Actualice primero el tiempo de ejecución manualmente o ejecutando el proyecto nodejs-upgrade-functions en el modo UPDATE (para obtener más información, consulte [Listado de funciones de Lambda y actualización a un tiempo de ejecución más reciente utilizando el proyecto de actualización del tiempo de ejecución \(p. 22\)](#)). Recomendamos actualizar cualquier uso del método context y sustituirlo por el enfoque de devolución de llamada. Para obtener más detalles, consulte [Los métodos del objeto context en el tiempo de ejecución v0.10.42 de Node.js \(p. 22\)](#).
 - b. Pruebe la función de Lambda y verifique que su comportamiento cumple sus criterios de validación interna. Si falla, es posible que deba actualizar el código de Lambda para que funcione en el nuevo tiempo de ejecución:
 - Para obtener una lista de los cambios realizados en Node.js v6.10, consulte [Breaking changes between v5 and v6](#) en GitHub.
 - Para obtener una lista de los cambios realizados en Node.js v4.3, consulte [API changes between v0.10 and v4](#) en GitHub.
 - c. Una vez que la función se ha invocado correctamente, la transición se ha completado.
 3. Revise las funciones existentes para ver las versiones y los alias. Puede obtener una lista de versiones para cada función utilizando [Listado de funciones de Lambda y actualización a un tiempo de ejecución más reciente utilizando la consola de Lambda \(p. 22\)](#) o [Listado de funciones de Lambda y actualización a un tiempo de ejecución más reciente utilizando el proyecto de actualización del tiempo de ejecución \(p. 22\)](#). Para cada una de esas versiones:
 - a. Copie el código en \$LATEST.
 - b. Repita el proceso desde el paso 2 anterior.
 - c. Vuelva a publicar el código cuando esté completo como una nueva versión.
 - d. Actualice cualquier alias que apunte actualmente a la versión anterior a la versión recién publicada.
 - e. Elimine la versión anterior.

Listado de funciones de Lambda y actualización del tiempo de ejecución utilizando la CLI

Puede utilizar el comando [ListFunctions \(p. 437\)](#) para devolver una lista de todas las funciones de Lambda y, a partir de esa lista, obtener las funciones creadas en el tiempo de ejecución v0.10. El siguiente ejemplo de código muestra cómo hacerlo:

```
#!/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."
```

Para cada función de Lambda devuelta que se haya creado utilizando el tiempo de ejecución v0.10, utilice el comando [UpdateFunctionConfiguration \(p. 472\)](#) y establezca el valor de `--runtime` en `nodejs4.3` o `nodejs6.10`.

Listado de funciones de Lambda y actualización a un tiempo de ejecución más reciente utilizando la consola de Lambda

- Inicie sesión en la Consola de administración de AWS y abra la consola de Lambda.
- Elija la pestaña Runtime. Esto ordenará todas las funciones de Lambda para esa región según el valor del tiempo de ejecución.
- Abra cada función de Lambda cuyo valor de tiempo de ejecución sea node.js y, a continuación, elija la pestaña Configuration.
- Establezca el valor de Runtime en Node.js 4.3 o Node.js 6.10.
- Repita este proceso para cada región según sea necesario.

Listado de funciones de Lambda y actualización a un tiempo de ejecución más reciente utilizando el proyecto de actualización del tiempo de ejecución

- Inicie sesión en la Consola de administración de AWS y abra la consola de Lambda.
- Elija Create a Lambda Function.
- Elija el proyecto `nodejs-upgrade-functions` y cree una función con él.
- Tenga en cuenta que la función tiene disponibles las siguientes variables de entorno:
 - `MODE` = List, Backup o Upgrade
 - `TARGET_RUNTIME` = `nodejs4.3` o `nodejs6.10`
 - `EXCLUDED` = una lista separada por comas de nombres de funciones que se deben excluir del procesamiento (no incluya espacios en la lista)
- Para obtener una lista de funciones y versiones, invoque la función desde la consola sin ningún cambio en los valores de las variables.
- Para hacer un backup de las funciones antes de la actualización, cambie el valor de `MODE` a `Backup` e invoque la función desde la consola. Recomendamos que haga esto antes de actualizar las funciones.
- Para actualizar el valor del tiempo de ejecución de las funciones, cambie el valor de `MODE` a `Upgrade` e invoque la función desde la consola.
- Repita este proceso para cada región según sea necesario.
- Tenga en cuenta que:
 - El proyecto guardará la función de Node.js v1.0 existente como una versión y actualizará `\$LATEST` a `nodejs4.3` o `nodejs6.10`, dependiendo de la versión que elija. No se pueden actualizar otras versiones de la función. Puede utilizar esta información de versión para apuntar cualquier aplicación existente a esa versión.
 - El proyecto no modifica los alias. Cualquier alias que haga referencia a esa función tendrá que ser reasignado a la nueva versión. Para obtener más información, consulte [Control de versiones y alias de las funciones de AWS Lambda \(p. 80\)](#).

Los métodos del objeto context en el tiempo de ejecución v0.10.42 de Node.js

El tiempo de ejecución v0.10.42 de Node.js no es compatible con el parámetro de devolución de llamada de la función de Lambda que admiten los tiempos de ejecución v4.3 y v6.10. Al utilizar el tiempo de ejecución v0.10.42, utilice los siguientes métodos del objeto context para terminar correctamente la función

de Lambda. El objeto context admite los métodos `done()`, `succeed()` y `fail()`, que puede utilizar para terminar la función de Lambda. Estos métodos también están presentes en los tiempos de ejecución v4.3 y v6.10 por motivos de compatibilidad con versiones anteriores. Para obtener información acerca de la transición del código para utilizar los tiempos de ejecución v4.3 o v6.10, consulte [Transición del código de funciones de Lambda a tiempos de ejecución más recientes \(p. 20\)](#).

`context.succeed()`

Indica que la ejecución de la función de Lambda y todas las devoluciones de llamada se han completado con éxito. La sintaxis general es:

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

Donde:

`result`: es un parámetro opcional que se puede utilizar para proporcionar el resultado de la ejecución de la función.

El valor proporcionado para `result` debe ser compatible con `JSON.stringify`. Si se produce un error en el método `stringify` o AWS Lambda encuentra otro error, se genera una excepción no gestionada con el encabezado de respuesta `X-Amz-Function-Error` establecido en `Unhandled`.

Puede llamar a este método sin ningún parámetro (`succeed()`) o bien pasarle un valor null (`succeed(null)`).

El comportamiento de este método depende del tipo de invocación especificado en la invocación de la función de Lambda. Para obtener más información acerca de los tipos de invocación, consulte [Invoke \(p. 423\)](#).

- Si se invoca la función de Lambda utilizando el tipo de invocación `Event` (invocación asíncrona), el método devolverá la respuesta HTTP status 202, `request accepted`.
- Si se invoca la función de Lambda utilizando el tipo de invocación `RequestResponse` (invocación síncrona), el método devolverá HTTP status 200 (OK) y establecerá el cuerpo de la respuesta en la representación de cadena de `result`.

`context.fail()`

Indica que la ejecución de la función de Lambda y todas las devoluciones de llamada se han completado sin éxito, lo que resulta en una excepción gestionada. La sintaxis general se muestra a continuación:

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

Donde:

`error`: es un parámetro opcional que puede utilizar para proporcionar el resultado de la ejecución de la función de Lambda.

Si el valor de `error` es distinto de null, el método establecerá el cuerpo de la respuesta en la representación de cadena de `error` y también escribirá los logs correspondientes en CloudWatch. Si se produce un error en el método `stringify` o AWS Lambda encuentra otro error, se generará un error no gestionado con el encabezado `X-Amz-Function-Error` establecido en `Unhandled`.

Note

Para el error de `context.fail(error)` y `context.done(error, null)`, Lambda registra los primeros 256 KB del objeto de error. Para objetos de error más grandes, AWS Lambda trunca el error y muestra el texto: `Truncated by Lambda` junto al objeto de error.

Puede llamar a este método sin ningún parámetro (`fail()`) o bien pasar un valor null (`fail(null)`).

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

Hace que finalice la ejecución de la función de Lambda.

Note

Este método complementa a los métodos `succeed()` y `fail()`, permitiendo el uso del patrón de diseño de devolución de llamada “error first”. No proporciona ninguna funcionalidad adicional.

La sintaxis general es:

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

Donde:

- `error`: es un parámetro opcional que puede utilizar para proporcionar los resultados de la ejecución con errores de la función de Lambda.
- `result`: es un parámetro opcional que puede utilizar para proporcionar el resultado de una ejecución satisfactoria de la función. El resultado proporcionado debe ser compatible con `JSON.stringify`. Si se proporciona un error, este parámetro se ignora.

Puede llamar a este método sin ningún parámetro (`done()`), o bien pasar un valor null (`done(null)`).

AWS Lambda trata cualquier valor distinto de null para el parámetro `error` como una excepción gestionada.

El comportamiento de la función depende del tipo de invocación especificado en el momento de la invocación de la función de Lambda. Para obtener más información acerca de los tipos de invocación, consulte [Invoke \(p. 423\)](#).

- Independientemente del tipo de invocación, el método registra automáticamente la representación de cadena de los valores de `error` distintos de null en el flujo de Amazon CloudWatch Logs asociado a la función de Lambda.
- Si la función de Lambda se invocó utilizando el tipo de invocación `RequestResponse` (síncrona), el método devolverá el cuerpo de la respuesta como se indica a continuación:
 - Si `error` es null, establezca el cuerpo de la respuesta en la representación de JSON de `result`. Esto es similar a `context.succeed()`.
 - Si `error` es un valor distinto de null o se llama a la función con un único argumento de tipo `error`, el cuerpo de la respuesta contendrá el valor de `error`.

Note

Para los errores de `done(error, null)` y `fail(error)`, Lambda registra los primeros 256 KB del objeto de error. En caso de un objeto de error más grande, AWS Lambda truncará el log y mostrará el texto `Truncated by Lambda` al lado del objeto de error.

[Comparación de los métodos del objeto context con el parámetro callback](#)

Si ha creado previamente funciones de Lambda utilizando el tiempo de ejecución v0.10.42 de Node.js, ha utilizado uno de los métodos del objeto `context` (`done()`, `succeed()` y `fail()`) para terminar la función de Lambda. En los tiempos de ejecución v4.3 y v6.10 de Node.js, estos métodos se admiten principalmente por compatibilidad con versiones anteriores. Recomendamos que utilice `callback` (consulte [Uso del parámetro callback \(p. 11\)](#)). A continuación se muestran ejemplos de `callback` equivalentes a los métodos del objeto `context`:

- En el siguiente ejemplo se muestra el método `context.done()` y su equivalente `callback` admitido en el tiempo de ejecución más reciente.

```
// 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

Por motivos de desempeño, AWS Lambda puede reutilizar el mismo proceso de Node.js para múltiples ejecuciones de la función de Lambda. Si esto sucede, AWS Lambda congela el proceso de Node entre ejecuciones, conservando la información de estado que necesita para continuar la ejecución.

Al llamar a los métodos del objeto `context`, AWS Lambda congela el proceso de Node inmediatamente, sin esperar a que se vacíe el bucle de eventos asociado al proceso. El estado del proceso y cualquier evento del bucle de eventos se congelan. Cuando se invoca nuevamente la función, si AWS Lambda reutiliza el proceso congelado, la ejecución de la función continúa con el mismo estado global (por ejemplo, los eventos que permanecieron en el bucle de eventos empezarán a procesarse). Sin embargo, cuando se utiliza la devolución de llamada, AWS Lambda continúa la ejecución de la función de Lambda hasta que el bucle de eventos esté vacío. Después de que se procesen todos los eventos del bucle de eventos, AWS Lambda congela el proceso de Node, incluyendo cualquier variable de estado de la función de Lambda. Por lo tanto, si desea el mismo comportamiento que los métodos del objeto `context`, debe establecer la propiedad `callbackWaitsForEmptyEventLoop` del objeto `context` en `false`.

- En el siguiente ejemplo se muestra el método `context.succeed()` y su equivalente `callback` admitido en el tiempo de ejecución más reciente.

```
// 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');
```

- En el siguiente ejemplo se muestra el método `context.fail()` y su equivalente `callback` admitido en el tiempo de ejecución más reciente.

```
// 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');
```

Modelo de programación para crear funciones de Lambda en Java

En las siguientes secciones se explica cómo se aplican los [conceptos fundamentales y los patrones de programación comunes](#) al crear código de funciones de Lambda en Java.

Temas

- [Controlador de funciones de Lambda \(Java\) \(p. 26\)](#)

- [El objeto context \(Java\) \(p. 36\)](#)
- [Registro \(Java\) \(p. 38\)](#)
- [Errores de funciones \(Java\) \(p. 42\)](#)

Además, tenga en cuenta que AWS Lambda proporciona las siguientes bibliotecas:

- aws-lambda-java-core: esta biblioteca proporciona el objeto Context y las interfaces RequestStreamHandler y RequestHandler. El objeto Context ([El objeto context \(Java\) \(p. 36\)](#)) proporciona información del tiempo de ejecución acerca de la función de Lambda. Las interfaces predefinidas proporcionan una forma de definir el controlador de la función de Lambda. Para obtener más información, consulte [Uso de interfaces predefinidas para la creación de un controlador \(Java\) \(p. 32\)](#).
- aws-lambda-java-events: esta biblioteca proporciona tipos predefinidos que se pueden utilizar a la hora de escribir funciones de Lambda para procesar eventos publicados por Amazon S3, Kinesis, Amazon SNS y Amazon Cognito. Estas clases le ayudan a procesar el evento sin necesidad de escribir su propia lógica de serialización personalizada.
- Custom Appender for Log4j2: puede utilizar el Appender personalizado de Log4j (consulte [Apache Log4j 2](#)), proporcionado por AWS Lambda, para el registro desde sus funciones de Lambda. Para obtener más información, consulte [Registro \(Java\) \(p. 38\)](#).

Estas bibliotecas están disponibles a través del repositorio central de Maven, y también se pueden encontrar en GitHub.

Controlador de funciones de Lambda (Java)

Cuando cree una función de Lambda debe especificar un controlador que AWS Lambda puede invocar cuando el servicio ejecute la función de Lambda en su nombre.

Lambda admite dos enfoques para crear un controlador:

- Cargar el método del controlador directamente sin tener que implementar una interfaz. En esta sección se describe este enfoque.
- Implementar interfaces estándar que se ofrecen como parte de la biblioteca aws-lambda-java-core (enfoque de interfaz). Para obtener más información, consulte [Uso de interfaces predefinidas para la creación de un controlador \(Java\) \(p. 32\)](#).

La sintaxis general para el controlador es la siguiente:

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

Para que AWS Lambda pueda invocar correctamente un controlador, se debe invocar la función con datos de entrada que se puedan serializar en el tipo de datos del parámetro `input`.

En la sintaxis, tenga en cuenta lo siguiente:

- `inputType`: el primer parámetro del controlador es su entrada, que puede estar formada por datos de eventos (publicados por un origen de eventos) o por una entrada personalizada proporcionada por el usuario, como una cadena o cualquier objeto de datos personalizado. Para que AWS Lambda pueda invocar correctamente este controlador, se debe invocar la función con datos de entrada que se puedan serializar en el tipo de datos del parámetro `input`.
- `outputType`: si piensa invocar la función de Lambda de forma síncrona (utilizando el tipo de invocación `RequestResponse`), puede devolver la salida de la función utilizando cualquiera de los tipos de datos

admitidos. Por ejemplo, si utiliza una función de Lambda como backend de aplicaciones móviles, la invocación se realiza de forma síncrona. El tipo de datos de salida se serializará en JSON.

Si piensa invocar la función de Lambda de forma asíncrona (utilizando el tipo de invocación `Event`), `outputType` debería ser `void`. Por ejemplo, si utiliza AWS Lambda con orígenes de eventos como Amazon S3 Kinesis y Amazon SNS, estos orígenes de eventos invocan la función de Lambda mediante el tipo de invocación `Event`.

- `inputType` y `outputType` pueden ser uno de los elementos siguientes:
 - Tipos de Java primitivos (como `String` o `int`).
 - Tipos de eventos predefinidos de AWS que se definen en la biblioteca `aws-lambda-java-events`.

Por ejemplo `S3Event` es una de las clases de objeto Java estándar (POJO) predefinidas en la biblioteca que proporciona métodos destinados a facilitar la lectura de la información del evento entrante de Amazon S3.

- También puede escribir su propia clase POJO. AWS Lambda serializará y deserializará automáticamente los JSON de entrada y salida en función del tipo POJO.

Para obtener más información, consulte [Tipos de entrada y salida del controlador \(Java\) \(p. 28\)](#).

- Puede omitir el objeto `Context` de la firma del método del controlador si no es necesario. Para obtener más información, consulte [El objeto context \(Java\) \(p. 36\)](#).

Por ejemplo, fíjese en el siguiente código de ejemplo de Java.

```
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);
    }
}
```

En este ejemplo, la entrada es de tipo `Integer` y la salida es de tipo `String`. Si empaqueta este código y las dependencias, y crea la función de Lambda, debe especificar `example.Hello::myHandler` (`paquete.clase::referencia-al-método`) como controlador.

En el código Java del ejemplo, el primer parámetro del controlador es la entrada al controlador (`myHandler`), que puede estar formada por datos de eventos (publicados por un origen de eventos como Amazon S3) o por una entrada personalizada proporcionada por el usuario, tal como un objeto `Integer` (como sucede en este ejemplo) o cualquier objeto de datos personalizado.

Para obtener instrucciones acerca de cómo crear una función de Lambda utilizando este código Java, consulte [Paso 2.3: Creación de una función de Lambda escrita en Java \(opcional\) \(p. 209\)](#).

Resolución de la sobrecarga del controlador

Si el código Java contiene varios métodos con el mismo nombre que el `handler`, AWS Lambda utiliza las siguientes reglas para elegir un método que debe invocar:

1. Selecciona el método con el mayor número de parámetros.
2. Si dos o más métodos tienen el mismo número de parámetros, AWS Lambda selecciona el método que tiene `Context` como último parámetro.

Si ninguno de los métodos tiene el parámetro `Context`, o bien lo tienen todos, el comportamiento no está definido.

Información adicional

Los siguientes temas proporcionan más información acerca del controlador.

- Para obtener más información acerca de los tipos de entrada y salida del controlador, consulte [Tipos de entrada y salida del controlador \(Java\) \(p. 28\)](#).
- Para obtener más información acerca de cómo utilizar interfaces predefinidas para crear un controlador, consulte [Uso de interfaces predefinidas para la creación de un controlador \(Java\) \(p. 32\)](#).

Si implementa estas interfaces, puede validar la firma del método del controlador durante la compilación.

- Si la función de Lambda genera una excepción, AWS Lambda registra métricas en CloudWatch indicando que se ha producido un error. Para obtener más información, consulte [Errores de funciones \(Java\) \(p. 42\)](#).

Tipos de entrada y salida del controlador (Java)

Cuando AWS Lambda ejecuta la función de Lambda, invoca el controlador. El primer parámetro es la entrada del controlador, que puede estar formada por datos de eventos (publicados por un origen de eventos) o por una entrada personalizada proporcionada por el usuario, como una cadena o cualquier objeto de datos personalizado.

AWS Lambda admite los siguientes tipos de entrada y salida para un controlador:

- Tipos de Java sencillos (AWS Lambda admite los tipos String, Integer, Boolean, Map y List)
- Tipo POJO (objeto Java estándar)
- Tipo de flujo (si no desea utilizar POJO o si el enfoque de serialización de Lambda no satisface sus necesidades, puede utilizar la implementación de flujo de bytes. Para obtener más información, consulte [Ejemplo: Uso de Stream para la entrada y salida del controlador \(Java\) \(p. 31\)](#)).

Entrada y salida del controlador: tipo String

La siguiente clase de Java muestra un controlador denominado `myHandler` que utiliza el tipo String para la entrada y la salida.

```
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);
    }
}
```

Puede tener funciones de controlador similares para otros tipos de Java sencillos.

Note

Al invocar una función de Lambda de forma asíncrona, los valores devueltos por dicha función se ignorarán. Por lo tanto, es conveniente establecer el tipo de retorno en void para que esto quede claro en el código. Para obtener más información, consulte [Invoke \(p. 423\)](#).

Para probar un ejemplo integral, consulte [Paso 2.3: Creación de una función de Lambda escrita en Java \(opcional\) \(p. 209\)](#).

Entrada y salida del controlador: tipo POJO

La siguiente clase de Java muestra un controlador denominado `myHandler` que utiliza POJO para la entrada y la salida.

```
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 lleva a cabo la serialización basándose en las convenciones de nomenclatura bean estándar (consulte [The Java EE 6 Tutorial](#)). Debe utilizar POJO mutables con métodos getter y setter públicos.

Note

No debe confiar en ninguna otra característica de los marcos de trabajo de serialización, como las anotaciones. Si necesita personalizar el comportamiento de la serialización, puede utilizar el flujo de bytes sin procesar en su propia serialización.

Si utiliza POJO para la entrada y la salida, debe proporcionar la implementación de los tipos `RequestClass` y `ResponseClass`. Para ver un ejemplo, consulte [Ejemplo: Uso de POJO para la entrada y salida del controlador \(Java\) \(p. 29\)](#).

Ejemplo: Uso de POJO para la entrada y salida del controlador (Java)

Suponga que los eventos de una aplicación generan datos que incluyen nombre y apellidos, tal como se muestra a continuación:

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

Para este ejemplo, el controlador recibe este JSON y devuelve la cadena "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);
}
```

Para crear una función de Lambda con este controlador, debe proporcionar la implementación de los tipos de entrada y salida, tal como se muestra en el siguiente ejemplo de Java. La clase `HelloPojo` define el método `handler`.

```
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);
    }
}
```

Para implementar el tipo de entrada, añada el siguiente código a un archivo independiente y asígnele el nombre RequestClass.java. Colóquelo junto a la clase HelloPojo.java en la estructura de directorios:

```
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() {
    }
}
```

Para implementar el tipo de salida, añada el siguiente código a un archivo independiente y asígnele el nombre ResponseClass.java. Colóquelo junto a la clase HelloPojo.java en la estructura de directorios:

```
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

Los métodos `get` y `set` son necesarios para que los POJO funcionen con el serializador JSON integrado en AWS Lambda. Los constructores que no toman argumentos generalmente no son necesarios; sin embargo, en este ejemplo se proporcionan otros constructores y, por lo tanto, es necesario especificar explícitamente los constructores de cero argumentos.

Puede cargar este código como función de Lambda y probarlo como se indica a continuación:

- Cree un paquete de implementación utilizando los archivos de código anteriores.
- Cargue el paquete de implementación en AWS Lambda y cree la función de Lambda. Para ello, puede utilizar la consola o la CLI de AWS.
- Invoque la función de Lambda manualmente utilizando la consola o la CLI. Puede utilizar datos de eventos JSON de muestra cuando invoque manualmente la función de Lambda. Por ejemplo:

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

Para obtener más información, consulte [Paso 2.3: Creación de una función de Lambda escrita en Java \(opcional\) \(p. 209\)](#). Tenga en cuenta las siguientes diferencias:

- Cuando cree un paquete de implementación, no olvide la dependencia de la biblioteca `aws-lambda-java-core`.
- Cuando cree la función de Lambda, especifique `example.HelloPojo::handleRequest` (`paquete.clase::método`) como valor del controlador.

Ejemplo: Uso de Stream para la entrada y salida del controlador (Java)

Si no desea utilizar POJO o si el enfoque de serialización de Lambda no satisface sus necesidades, puede utilizar la implementación de flujo de bytes. En este caso, puede utilizar `InputStream` y `OutputStream` como tipos de entrada y salida para el controlador. A continuación se muestra una función de controlador de ejemplo:

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

Observe que, en este caso, la función de controlador utiliza parámetros para los flujos de solicitud y de respuesta.

A continuación se muestra un ejemplo de función de Lambda que implementa el controlador que utiliza los tipos `InputStream` y `OutputStream` para los parámetros `input` y `output`.

Note

La carga de entrada debe ser JSON válido, pero el flujo de salida no tiene esta restricción. Se admite cualquier tipo de bytes.

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

Puede hacer lo siguiente para probar el código:

- Cree un paquete de implementación utilizando el código anterior.
- Cargue el paquete de implementación en AWS Lambda y cree la función de Lambda. Para ello, puede utilizar la consola o la CLI de AWS.
- Puede invocar manualmente el código proporcionando entrada de muestra. Por ejemplo:

```
test
```

Siga las instrucciones proporcionadas en la Introducción. Para obtener más información, consulte [Paso 2.3: Creación de una función de Lambda escrita en Java \(opcional\) \(p. 209\)](#). Tenga en cuenta las siguientes diferencias:

- Cuando cree un paquete de implementación, no olvide la dependencia de la biblioteca aws-lambda-java-core.
- Cuando cree la función de Lambda, especifique `example.Hello::handler` (`paquete.clase::método`) como valor del controlador.

Uso de interfaces predefinidas para la creación de un controlador (Java)

Puede utilizar una de las interfaces predefinidas que proporciona la biblioteca principal de Java de AWS Lambda (`aws-lambda-java-core`) para crear el controlador de la función de Lambda, en lugar de escribir un método de controlador propio con un nombre y unos parámetros arbitrarios. Para obtener más información acerca de los controladores, consulte [Controlador de funciones de Lambda \(Java\) \(p. 26\)](#).

Puede implementar una de las interfaces predefinidas, `RequestStreamHandler` o `RequestHandler` y proporcionar la implementación para el método `handleRequest` que ofrecen las interfaces. Puede implementar una de estas interfaces dependiendo de si desea utilizar tipos estándar de Java o tipos POJO personalizados para la entrada y salida del controlador (donde AWS Lambda serializa y deserializa automáticamente la entrada y salida para que coincidan con el tipo de datos) o personalizar la serialización utilizando el tipo `Stream`.

Note

Estas interfaces están disponibles en la biblioteca `aws-lambda-java-core`.

Implementar interfaces estándar le ayuda a validar la firma del método durante la compilación.

Si implementa una de las interfaces, especifique `paquete.clase` como controlador en el código Java al crear la función de Lambda. Por ejemplo, a continuación se muestra el comando `create-function` de la CLI modificado con respecto al de la introducción. Observe que el parámetro `--handler` especifica el valor “`example.Hello`”:

```
aws lambda create-function \
--region us-west-2 \
--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
```

Las siguientes secciones proporcionan ejemplos de implementación de estas interfaces.

Ejemplo 1: Creación de un controlador con entrada y salida POJO personalizada (utilizando la interfaz RequestHandler)

La clase de ejemplo Hello de esta sección implementa la interfaz RequestHandler. La interfaz define el método handleRequest() que toma los datos del evento como parámetro de entrada del tipo Request y devuelve un objeto POJO del tipo Response:

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

Se muestra la clase Hello con una implementación de ejemplo del método handleRequest(). Para este ejemplo, suponemos que los datos del evento constan de nombre y apellidos.

```
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);
    }
}
```

Por ejemplo, si los datos del evento en el objeto Request son:

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

El método devuelve el objeto Response siguiente:

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

A continuación, debe implementar las clases Request y Response. Puede utilizar la siguiente implementación para hacer pruebas:

La clase Request:

```
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() {
    }
}
```

La clase Response:

```
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() {
    }
}
```

Puede crear una función de Lambda con este código y probar la experiencia integral como se indica a continuación:

- Cree un paquete de implementación utilizando el código anterior.
- Cargue el paquete de implementación en AWS Lambda y cree la función de Lambda.
- Pruebe la función de Lambda utilizando la consola o la CLI. Puede especificar cualquier muestra de datos JSON que se ajuste a los métodos getter y setter de la clase Request, como por ejemplo:

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

La función de Lambda devolverá el siguiente JSON como respuesta.

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

Siga las instrucciones proporcionadas en la introducción (consulte [Paso 2.3: Creación de una función de Lambda escrita en Java \(opcional\) \(p. 209\)](#)). Tenga en cuenta las siguientes diferencias:

- Cuando cree un paquete de implementación, no olvide la dependencia de la biblioteca aws-lambda-java-core.
- Cuando cree la función de Lambda, especifique example.Hello ([paquete.clase](#)) como valor del controlador.

Ejemplo 2: Creación de un controlador con entrada y salida de flujo (utilizando la interfaz RequestStreamHandler)

La clase Hello de este ejemplo implementa la interfaz RequestStreamHandler. La interfaz define el método handleRequest del modo siguiente:

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

Se muestra la clase Hello con una implementación de ejemplo del controlador handleRequest(). El controlador procesa los datos de los eventos entrantes (por ejemplo, una cadena "hello") simplemente convirtiéndolos en mayúsculas y devolviéndolos.

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

Puede crear una función de Lambda con este código y probar la experiencia integral como se indica a continuación:

- Utilice el código anterior para crear un paquete de implementación.
- Cargue el paquete de implementación en AWS Lambda y cree la función de Lambda.
- Pruebe la función de Lambda utilizando la consola o la CLI. Puede especificar cualquier muestra de datos de cadena, por ejemplo:

```
"test"
```

La función de Lambda devolverá TEST como respuesta.

Siga las instrucciones proporcionadas en la introducción (consulte [Paso 2.3: Creación de una función de Lambda escrita en Java \(opcional\) \(p. 209\)](#)). Tenga en cuenta las siguientes diferencias:

- Cuando cree un paquete de implementación, no olvide la dependencia de la biblioteca aws-lambda-java-core.
- Cuando cree la función de Lambda, especifique example.Hello (*paquete.clase*) como valor del controlador.

El objeto context (Java)

El parámetro context sirve para interactuar con un entorno de ejecución de AWS Lambda. El objeto context le permite acceder a información útil disponible en el entorno de ejecución de Lambda. Por ejemplo, puede utilizar el parámetro context para determinar el flujo de logs de CloudWatch asociado a la función, o utilizar la propiedad clientContext del objeto context para obtener más información acerca de la aplicación que llama a la función de Lambda (cuando se invoca mediante AWS Mobile SDK).

Las propiedades del objeto context son:

- `getMemoryLimitInMB()`: cantidad máxima de memoria, en MB, configurada para la función de Lambda.
- `getFunctionName()`: nombre de la función de Lambda que se está ejecutando.
- `getFunctionVersion()`: la versión de la función de Lambda que se está ejecutando. Si se utiliza un alias para invocar la función, `getFunctionVersion` será la versión a la que apunta el alias.
- `getInvokedFunctionArn()`: el ARN utilizado para invocar esta función. Puede ser el ARN de la función o el ARN del alias. Un ARN incompleto ejecuta la versión \$LATEST y los alias ejecutan la versión de la función a la que apunta.
- `getAwsRequestId()`: el ID de la solicitud de AWS asociado a la solicitud. Este es el ID que se devuelve al cliente que llamó a `invoke()`. Puede utilizar el ID de la solicitud para cualquier consulta de seguimiento con AWS Support. Tenga en cuenta que si AWS Lambda reintenta la función (por ejemplo, en una situación en la que la función de Lambda que está procesando registros de Kinesis genera una excepción), el ID de la solicitud sigue siendo el mismo.
- `getLogStreamName()`: el nombre del flujo de logs de CloudWatch para la ejecución de la función de Lambda en concreto. Puede ser null si el usuario de IAM proporcionado no tiene permiso para realizar acciones de CloudWatch.
- `getLogGroupName()`: el nombre del grupo de logs de CloudWatch asociado a la función de Lambda invocada. Puede ser null si el usuario de IAM proporcionado no tiene permiso para realizar acciones de CloudWatch.
- `getClientContext()`: información acerca del dispositivo y la aplicación cliente cuando se invoca a través de AWS Mobile SDK. Puede ser null. ClientContext proporciona información del cliente, como el ID del cliente, el título de la aplicación, el nombre de la versión, el código de la versión y el nombre del paquete de la aplicación.

- `getIdentity()`: información acerca del proveedor de identidad de Amazon Cognito cuando se invoca a través de AWS Mobile SDK. Puede ser null.
- `getRemainingTimeInMillis()`: tiempo que falta para terminar la ejecución de la función, en milisegundos. En el momento en que se crea la función de Lambda, se establece el límite de tiempo máximo, momento en el que AWS Lambda terminará la ejecución de la función. La información acerca del tiempo restante para la ejecución de la función se puede utilizar para especificar el comportamiento de la función al acercarse al tiempo de espera.
- `getLogger()`: devuelve el registrador de Lambda asociado al objeto context. Para obtener más información, consulte [Registro \(Java\) \(p. 38\)](#).

A continuación se muestra un fragmento de código Java de una función de controlador que imprime parte de la información de contexto.

```
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());  
}
```

Ejemplo: Uso del objeto context (Java)

El siguiente ejemplo de código Java muestra cómo utilizar el objeto Context para recuperar información del tiempo de ejecución de la función de Lambda mientras se está ejecutando.

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

Puede hacer lo siguiente para probar el código:

- Cree un paquete de implementación utilizando el código anterior.
- Cargue el paquete de implementación en AWS Lambda para crear la función de Lambda. Para ello, puede utilizar la consola o AWS CLI.
- Para probar la función de Lambda, utilice el evento de muestra "Hello World" que proporciona la consola de Lambda.

Puede escribir cualquier cadena y la función devolverá la misma cadena en mayúsculas. Además, obtendrá información útil sobre la función proporcionada por el objeto `context`.

Siga las instrucciones proporcionadas en la sección Introducción. Para obtener más información, consulte [Paso 2.3: Creación de una función de Lambda escrita en Java \(opcional\) \(p. 209\)](#). Tenga en cuenta las siguientes diferencias:

- Cuando cree un paquete de implementación, no olvide la dependencia de la biblioteca `aws-lambda-java-core`.
- Cuando cree la función de Lambda, especifique `example.Hello::myHandler` (`package.class::method`) como valor del controlador.

Registro (Java)

La función de Lambda puede contener instrucciones de registro. AWS Lambda escribe estos logs en CloudWatch. Le recomendamos que utilice una de las siguientes herramientas para escribir los logs.

- Custom Appender for Log4j™ 2

AWS Lambda recomienda Log4j 2 para proporcionar un Appender personalizado. Puede utilizar el Appender personalizado de Log4j (consulte [Apache log4j](#)) proporcionado por Lambda para el registro desde sus funciones de Lambda. Cada llamada a los métodos de Log4j, como `log.debug()` o `log.error()`, producirá un evento de CloudWatch Logs. El Appender personalizado se denomina `LambdaAppender` y se debe utilizar en el archivo `log4j.properties`. Debe incluir el artefacto `aws-lambda-java-log4j` (`artifactId:aws-lambda-java-log4j`) en el paquete de implementación (archivo .jar). Para ver un ejemplo, consulte [Ejemplo 1: Escritura de logs mediante Log4J \(Java\) \(p. 39\)](#).

Note

Actualmente, AWS Lambda admite las versiones 1.2 y 2 de Log4j.

- `LambdaLogger.log()`

Cada llamada a `LambdaLogger.log()` da como resultado un evento de CloudWatch Logs, siempre que el tamaño del evento esté dentro de los límites permitidos. Para obtener información acerca de los límites de CloudWatch Logs, consulte [Límites de CloudWatch Logs](#) en la Guía del usuario de Amazon CloudWatch. Para ver un ejemplo, consulte [Ejemplo 2: Escritura de logs mediante LambdaLogger \(Java\) \(p. 41\)](#).

Además, también puede utilizar las siguientes instrucciones en el código de la función de Lambda para generar entradas de log:

- `System.out()`

- `System.err()`

No obstante, tenga en cuenta que AWS Lambda considera cada una de las líneas devueltas por `System.out` y `System.err` como un evento diferente. Esto funciona bien cuando cada línea de salida corresponde a una única entrada de log. Cuando una entrada de log tiene varias líneas de salida, AWS Lambda intenta analizarlas utilizando saltos de línea para identificar los distintos eventos. Por ejemplo, la siguiente instrucción registra las dos palabras ("Hello" y "world") como dos eventos diferentes:

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

Dónde encontrar los logs

Puede encontrar los logs que escribe la función de Lambda como se indica a continuación:

- Busque los logs en CloudWatch Logs. El objeto `context` (de la biblioteca `aws-lambda-java-core`) proporciona los métodos `getLogStreamName()` y `getLogGroupName()`. Utilice estos métodos para encontrar el flujo de logs específico donde se escriben los logs.
- Si invoca una función de Lambda a través de la consola, el tipo de invocación es siempre `RequestResponse` (es decir, ejecución síncrona) y la consola muestra los logs que escribe la función de Lambda mediante el objeto `LambdaLogger`. AWS Lambda también devuelve logs de los métodos `System.out` y `System.err`.
- Si se invoca una función de Lambda mediante programación, se puede añadir el parámetro `LogType` para recuperar los últimos 4 KB de datos de log escritos en CloudWatch Logs. Para obtener más información, consulte [Invoke \(p. 423\)](#). AWS Lambda devuelve esta información de log en el encabezado `x-amz-log-results` de la respuesta. Si utiliza la AWS Command Line Interface para invocar la función, puede especificar el parámetro `--log-type` con el valor `Tail`.

Ejemplos de registro (Java)

En esta sección se proporcionan ejemplos del uso de Custom Appender for Log4j y los objetos `LambdaLogger` para el registro de información.

Ejemplo 1: Escritura de logs mediante Log4J (Java)

El siguiente ejemplo de código Java escribe los logs utilizando los métodos `System` y `Log4j` para ilustrar sus diferencias cuando AWS Lambda registra información en CloudWatch.

```
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");

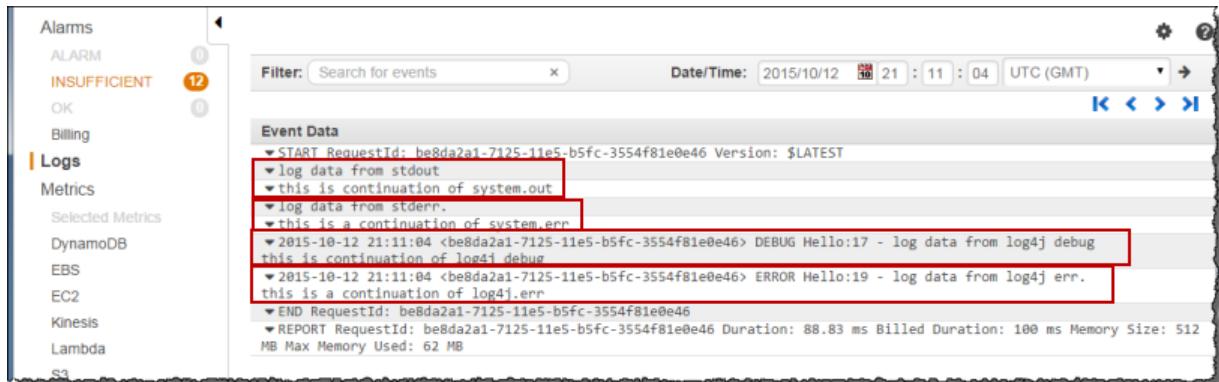
    // Return will include the log stream name so you can look
    // up the log later.
    return String.format("Hello %s. log stream = %s", name,
context.getLogStreamName());
}
```

El ejemplo utiliza el siguiente archivo log4j.properties (directorio [directorio-del-proyecto/src/main/resources/](#)).

```
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
```

A continuación se muestra un ejemplo de entradas de log en CloudWatch Logs.



Nota:

- AWS Lambda analiza la cadena de log de cada uno de los logs de las instrucciones `System.out.println()` y `System.err.println()` como dos eventos independientes (observe las dos flechas hacia abajo en la captura de pantalla) debido al salto de línea.
- Los métodos de Log4j (`log.debug()` y `log.error()`) producen un evento de CloudWatch.
- El tiempo de ejecución de AWS Lambda añade el valor de `AWSRequestId` en el MDC (consulte [Class MDC](#)). Para obtener este valor en el log tal como se muestra, hemos añadido `%X{AWSRequestId}` en el patrón de conversión en el archivo `log4j.properties`.

Puede hacer lo siguiente para probar el código:

- Utilizando el código, cree un paquete de implementación. En el proyecto, no se olvide de añadir el archivo `log4j.properties` al directorio [directorio-del-proyecto/src/main/resources/](#).
- Cargue el paquete de implementación en AWS Lambda para crear la función de Lambda.
- Para probar la función de Lambda utilice una cadena ("esto es una prueba") como evento de muestra. El código del controlador recibe el evento de muestra, pero no hace nada con él. Solo muestra cómo escribir los logs.

Siga las instrucciones proporcionadas en la sección Introducción. Para obtener más información, consulte [Paso 2.3: Creación de una función de Lambda escrita en Java \(opcional\) \(p. 209\)](#). Tenga en cuenta las siguientes diferencias:

- Cuando cree un paquete de implementación, no olvide la dependencia de la biblioteca aws-lambda-java-log4j.
- Cuando cree la función de Lambda, especifique `example.Hello::myHandler` (`package.class::method`) como valor del controlador.

Ejemplo 2: Escritura de logs mediante LambdaLogger (Java)

El siguiente ejemplo de código Java escribe los logs utilizando los métodos `System` y el objeto `LambdaLogger` para ilustrar sus diferencias cuando AWS Lambda registra información en CloudWatch.

```
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",
        context.getLogStreamName());
    }
}
```

A continuación se muestra un ejemplo de entradas de log en CloudWatch Logs.

```
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
```

Nota:

- AWS Lambda analiza la cadena de log de cada uno de los logs de las instrucciones `System.out.println()` y `System.err.println()` como dos eventos independientes (observe las dos flechas hacia abajo en la captura de pantalla) debido al salto de línea.

- `LambdaLogger.log()` produce un evento de CloudWatch.

Puede hacer lo siguiente para probar el código:

- Utilizando el código, cree un paquete de implementación.
- Cargue el paquete de implementación en AWS Lambda para crear la función de Lambda.
- Para probar la función de Lambda utilice una cadena ("esto es una prueba") como evento de muestra. El código del controlador recibe el evento de muestra, pero no hace nada con él. Solo muestra cómo escribir los logs.

Siga las instrucciones proporcionadas en la sección Introducción. Para obtener más información, consulte [Paso 2.3: Creación de una función de Lambda escrita en Java \(opcional\) \(p. 209\)](#). Tenga en cuenta las siguientes diferencias:

- Cuando cree un paquete de implementación, no olvide la dependencia de la biblioteca `aws-lambda-java-core`.
- Cuando cree la función de Lambda, especifique `example.Hello::myHandler` (`package.class::method`) como valor del controlador.

Errores de funciones (Java)

Si la función de Lambda genera una excepción, AWS Lambda reconoce el error y serializa la información de la excepción en JSON y la devuelve. A continuación se muestra un ejemplo de mensaje de error:

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

Observe que el rastro de stack se devuelve como la matriz JSON `stackTrace` de elementos de rastro de stack.

El método en el que se obtiene la información de error depende del tipo de invocación que se haya especificado en el momento de invocar la función:

- Tipo de invocación `RequestResponse` (es decir, ejecución síncrona): en este caso, se obtiene el mensaje de error.

Por ejemplo, si invoca una función de Lambda utilizando la consola de Lambda, `RequestResponse` siempre es el tipo de invocación, y la consola muestra la información de error devuelta por AWS Lambda en la sección Execution result, tal como se muestra en la siguiente imagen.



- Tipo de invocación **Event** (es decir, ejecución asíncrona): en este caso, AWS Lambda no devuelve nada. En su lugar, registra la información de error en las métricas de CloudWatch y CloudWatch Logs.

En función del origen de eventos, AWS Lambda puede volver a intentar la ejecución de la función de Lambda que ha fallado. Por ejemplo, si Kinesis es el origen de eventos de la función de Lambda, AWS Lambda reintentará la función que produce errores hasta que la función de Lambda se ejecute correctamente o hasta que los registros del flujo caduquen.

Control de errores de las funciones

Puede crear un control de errores personalizado para generar una excepción directamente desde la función de Lambda y gestionarla directamente (Retry o Catch) en una máquina de estado de AWS Step Functions. Para obtener más información, consulte [Administración de las condiciones de error con una máquina de estado](#).

Considere que un **estado CreateAccount** es una **tarea** que escribe los datos de un cliente en una base de datos utilizando una función de Lambda.

- Si la tarea se realiza correctamente, se crea una cuenta y se envía un correo electrónico de bienvenida.
- Si un usuario intenta crear una cuenta para un nombre de usuario que ya existe, la función de Lambda genera un error, lo que hace que la máquina de estado sugiera otro nombre de usuario y vuelva a intentar el proceso de creación de la cuenta.

En los siguientes códigos de muestra se ilustra cómo hacerlo. Tenga en cuenta que los errores personalizados en Java deben extender la clase `Exception`.

```
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!");
```

```
}
```

Puede configurar Step Functions para que detecte el error utilizando una regla `Catch`. Lambda establece automáticamente el nombre del error en el nombre de clase completo de la excepción en tiempo de ejecución:

```
{
  "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"
        }
      ],
      ...
    }
  }
}
```

En tiempo de ejecución, AWS Step Functions detecta el error y realiza una [transición](#) al estado `SuggestAccountName`, según se especifica en la transición `Next`.

El control de errores personalizado facilita la creación de aplicaciones [sin servidor](#). Esta característica se integra con todos los lenguajes admitidos por el [Modelo de programación \(p. 8\)](#) de Lambda, lo que le permite diseñar la aplicación en los lenguajes de programación que elija, combinándolos a medida que avanza.

Para obtener más información acerca de cómo crear sus propias aplicaciones sin servidor utilizando AWS Step Functions y AWS Lambda, consulte [AWS Step Functions](#).

Modelo de programación para crear funciones de Lambda en Python

En las siguientes secciones se explica cómo se aplican los [conceptos fundamentales y los patrones de programación comunes](#) al crear código de funciones de Lambda en Python.

Temas

- [Controlador de funciones de Lambda \(Python\) \(p. 44\)](#)
- [El objeto context \(Python\) \(p. 46\)](#)
- [Registro \(Python\) \(p. 48\)](#)
- [Errores de funciones \(Python\) \(p. 51\)](#)

Controlador de funciones de Lambda (Python)

En el momento de crear una función de Lambda, debe especificar un controlador, que es una función en el código, que AWS Lambda puede invocar cuando el servicio ejecuta el código. Utilice la siguiente estructura de sintaxis general al crear una función de controlador en Python.

```
def handler_name(event, context):
```

```
...  
return some_value
```

En la sintaxis, tenga en cuenta lo siguiente:

- **event**: AWS Lambda utiliza este parámetro para pasar datos de eventos al controlador. Este parámetro suele ser del tipo `dict` de Python. También puede ser del tipo `list`, `str`, `int`, `float`, o `NoneType`.
- **context**: AWS Lambda utiliza este parámetro para proporcionar información sobre el tiempo de ejecución al controlador. Este parámetro es del tipo `LambdaContext`.
- Si lo desea, el controlador puede devolver un valor. Lo que sucede con el valor devuelto depende del tipo de invocación que se utilice al invocar la función de Lambda:
 - Si se utiliza el tipo de invocación `RequestResponse` (ejecución síncrona), AWS Lambda devuelve el resultado de la llamada a la función de Python al cliente que invoca la función de Lambda (en la respuesta HTTP a la solicitud de invocación, serializada en JSON). Por ejemplo, la consola de AWS Lambda utiliza el tipo de invocación `RequestResponse`, por lo que al invocar la función utilizando la consola, esta mostrará el valor devuelto.

Si el controlador devuelve `NONE`, AWS Lambda devuelve un valor nulo.

- Si se utiliza el tipo de invocación `Event` (ejecución asíncrona), el valor se descarta.

Por ejemplo, fíjese en el siguiente código de ejemplo de Python.

```
def my_handler(event, context):  
    message = 'Hello {} {}!'.format(event['first_name'],  
                                    event['last_name'])  
    return {  
        'message' : message  
    }
```

Este ejemplo tiene una función denominada `my_handler`. La función devuelve un mensaje que contiene los datos del evento que recibió como entrada.

Para cargar y probar este código como una función de Lambda

1. Guarde el archivo (por ejemplo, como `hello_python.py`).
2. Empaque el archivo y las dependencias en un archivo `.zip`. Al crear el archivo zip, incluya únicamente el código y sus dependencias, no la carpeta donde se encuentran.

Para obtener instrucciones, consulte [Creación de un paquete de implementación \(Python\) \(p. 77\)](#).

3. Cargue el archivo `.zip` utilizando la consola o la AWS CLI para crear una función de Lambda. El nombre de la función se especifica en el código de Python que se utilizará como controlador al crear una función de Lambda. Para obtener instrucciones acerca de cómo crear una función de Lambda utilizando la consola, consulte [Paso 2.1: Creación de una función Hello World de Lambda \(p. 203\)](#). En este ejemplo, el controlador es `hello_python.my_handler` (`nombre-de-archivo.nombre-de-función`). Tenga en cuenta que en la [Introducción \(p. 199\)](#) se utiliza un proyecto que proporciona código de muestra para una función de Lambda. En este caso, ya dispone de un paquete de implementación. Por lo tanto, en la sección `Configure function` debe cargar un zip.

El siguiente comando `create-function` de la AWS CLI crea una función de Lambda. Entre otros, especifica el parámetro `--handler` para establecer el nombre del controlador. Observe que el parámetro `--runtime` especifica `python3.6`. También puede utilizar `python2.7`.

```
aws lambda create-function \  
--region us-west-2 \  
--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
```

El objeto context (Python)

Temas

- [Ejemplo \(p. 46\)](#)
- [Los métodos del objeto context \(Python\) \(p. 47\)](#)
- [Los atributos del objeto context \(Python\) \(p. 47\)](#)

Mientras se está ejecutando una función de Lambda, puede interactuar con el servicio de AWS Lambda para obtener información útil sobre el tiempo de ejecución, como por ejemplo:

- Cantidad de tiempo restante para que AWS Lambda termine la función de Lambda (el tiempo de espera es una de las propiedades de configuración de las funciones de Lambda).
- El grupo de logs y el flujo de logs de CloudWatch asociados a la función de Lambda que se está ejecutando.
- El ID de la solicitud de AWS devuelto al cliente que invocó la función de Lambda. Puede utilizar el ID de la solicitud para cualquier consulta de seguimiento con AWS Support.
- Si se invoca la función de Lambda a través de AWS Mobile SDK, se puede obtener más información sobre la aplicación móvil llamando a la función de Lambda.

AWS Lambda proporciona esta información a través del objeto `context`, que el servicio pasa como segundo parámetro al controlador de la función de Lambda. Para obtener más información, consulte [Controlador de funciones de Lambda \(Python\) \(p. 44\)](#).

Las siguientes secciones proporcionan una función de Lambda de ejemplo que utiliza el objeto `context` y, a continuación, enumera todos los métodos y atributos disponibles.

Ejemplo

Considere el siguiente ejemplo de Python. Tiene una función que también es el controlador. El controlador recibe información del tiempo de ejecución a través del objeto `context` que se pasa como parámetro.

```
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())
```

El código del controlador de este ejemplo simplemente imprime parte de la información del tiempo de ejecución. Cada instrucción `print` crea una entrada de log en CloudWatch. Si invoca la función utilizando la consola de Lambda, esta mostrará los logs. La instrucción `from __future__` permite escribir código compatible con Python versión 2 o 3.

Para probar este código en la consola de AWS Lambda

1. En la consola, cree una función de Lambda mediante el proyecto hello-world. En runtime, elija Python 2.7. En Handler, sustituya `lambda_function.lambda_handler` por `lambda_function.get_my_log_stream`. Para obtener instrucciones al respecto, consulte [Paso 2.1: Creación de una función Hello World de Lambda \(p. 203\)](#).
2. Pruebe la función. También puede actualizar el código para obtener más información de contexto.

En las secciones siguientes se incluye una lista de los métodos y atributos disponibles en el objeto `context` que puede utilizar para obtener información del tiempo de ejecución de una función de Lambda.

Los métodos del objeto context (Python)

El objeto `context` proporciona los siguientes métodos:

`get_remaining_time_in_millis()`

Devuelve el tiempo que falta para que AWS Lambda termine la ejecución de la función, en milisegundos.

Los atributos del objeto context (Python)

El objeto `context` proporciona los siguientes atributos:

`function_name`

Nombre de la función de Lambda que se está ejecutando.

`function_version`

La versión de la función de Lambda que se está ejecutando. Si se utiliza un alias para invocar la función, `function_version` será la versión a la que apunta el alias.

`invoked_function_arn`

El ARN utilizado para invocar esta función. Puede ser el ARN de la función o el ARN del alias. Un ARN incompleto ejecuta la versión `$LATEST` y los alias ejecutan la versión de la función a la que apunta.

`memory_limit_in_mb`

Cantidad máxima de memoria, en MB, configurada para la función de Lambda. El límite de memoria se establece en el momento de crear una función de Lambda, pero puede cambiarse más adelante.

`aws_request_id`

El ID de la solicitud de AWS asociado a la solicitud. Este es el ID que se devuelve al cliente que llamó al método `invoke`.

Note

Si AWS Lambda reintenta la invocación (por ejemplo, en una situación en la que la función de Lambda que está procesando registros de Kinesis genera una excepción), el ID de la solicitud sigue siendo el mismo.

`log_group_name`

El nombre del grupo de logs de CloudWatch donde se encuentran los registros escritos por la función de Lambda.

`log_stream_name`

El nombre del flujo de logs de CloudWatch donde se encuentran los registros escritos por la función de Lambda. El flujo de logs puede cambiar o no para cada invocación de la función de Lambda.

El valor es nulo si la función de Lambda no puede crear un flujo de logs, lo que puede ocurrir si el rol de ejecución que concede los permisos necesarios a la función de Lambda no incluye permisos para las acciones de CloudWatch Logs.

identity

Información acerca del proveedor de identidad de Amazon Cognito cuando se invoca a través del AWS Mobile SDK. Puede ser null.

- identity.cognito_identity_id
- identity.cognito_identity_pool_id

client_context

Información acerca del dispositivo y la aplicación cliente cuando se invocan a través de AWS Mobile SDK. Puede ser null.

- 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

Un elemento dict de valores personalizados establecidos por la aplicación cliente para móviles.

- client_context.env

Un elemento dict de información de entorno proporcionado AWS Mobile SDK.

Registro (Python)

La función de Lambda puede contener instrucciones de registro. AWS Lambda escribe estos logs en CloudWatch. Si utiliza la consola de Lambda para invocar la función de Lambda, la consola muestra los mismos logs.

Las siguientes instrucciones de Python generan entradas de log:

- Instrucciones print
- Funciones Logger en el módulo logging (por ejemplo, logging.Logger.info y logging.Logger.error).

Las funciones print y logging.* escriben logs en CloudWatch Logs, pero las funciones logging.* escriben información adicional para cada entrada de log, como la marca de tiempo y el nivel de log.

Por ejemplo, fíjese en el siguiente código de ejemplo de Python.

```
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 World!'
```

Debido a que el ejemplo de código utiliza el módulo logging para escribir mensajes en los logs, también se obtiene información adicional en el log, como la marca temporal y los niveles de log. El nivel de log identifica el tipo de log, como por ejemplo [INFO], [ERROR] y [DEBUG], tal como se muestra a continuación:

The screenshot shows the AWS Lambda execution results for a successful function call. It includes a summary of the function's configuration and runtime details, and a detailed log output section.

Execution result: succeeded (logs)

Summary

- Code SHA-256: p33JyPdxOLkWvmSOT
- Request ID: bd7abf08-6cc3-11e7-8809-296b5aa7925e
- Duration: 8.83 ms
- Billed duration: 100 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: bd7abf08-6cc3-11e7-8809-296b5aa7925e Version: $LATEST
value1 = Hello World
value2 = value2
value3 = value3
END RequestId: bd7abf08-6cc3-11e7-8809-296b5aa7925e
REPORT RequestId: bd7abf08-6cc3-11e7-8809-296b5aa7925e Duration: 8.83 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 23 MB
```

La captura de pantalla muestra un ejemplo de la sección Log output de la consola de Lambda. También puede encontrar estos logs en CloudWatch. Para obtener más información, consulte [Acceso a los logs de Amazon CloudWatch para AWS Lambda \(p. 124\)](#).

En lugar de utilizar el módulo logging, puede utilizar instrucciones print en el código, como se muestra en el siguiente ejemplo de Python:

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

En este caso, solo se envía a CloudWatch el texto pasado al método print. Las entradas de log no tendrán la información adicional que devuelve logging.*. La instrucción from __future__ permite escribir código compatible con Python versión 2 o 3.

The screenshot shows the AWS Lambda execution results for a successful function call. The summary includes:

- Code SHA-256: Mj6sNoYuSX2UjGoXUy
64+R/UvzWB0xJc7VdX
HRN6hqY=
- Request ID: 6a6a240b-6cc4-11e7-8734-6f6a02ed0559
- Duration: 0.18 ms
- Billed duration: 100 ms

The Log output section displays the following log entries:

```
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
```

La consola utiliza el tipo de invocación `RequestResponse` (invocación síncrona) al invocar la función. Y, por tanto, obtiene el valor de retorno ("Hello world!") de AWS Lambda que se muestra en la consola.

Para probar el código de Python anterior (consola)

1. En la consola, cree una función de Lambda mediante el proyecto `hello-world-python`. En `runtime`, elija Python 2.7. En `Handler`, sustituya `lambda_function.lambda_handler` por `lambda_function.my_other_logging_handler` y en `Role`, elija Basic execution role. También debe sustituir el código proporcionado por el proyecto por el código de esta sección. Para obtener instrucciones paso a paso para crear una función de Lambda utilizando la consola, consulte [Paso 2.1: Creación de una función Hello World de Lambda \(p. 203\)](#).
2. Sustituya el código de la plantilla por el código proporcionado en esta sección.
3. Pruebe la función de Lambda utilizando la plantilla Sample event template denominada Hello World proporcionada en la consola de Lambda.

Dónde encontrar los logs

Puede encontrar los logs que escribe la función de Lambda como se indica a continuación:

- En la consola de AWS Lambda: la sección Log output de la consola de AWS Lambda muestra los logs.
- En el encabezado de respuesta, cuando se invoca una función de Lambda mediante programación: si se invoca una función de Lambda mediante programación, se puede añadir el parámetro `LogType` para recuperar los últimos 4 KB de datos de log escritos en CloudWatch Logs. AWS Lambda devuelve esta información de log en el encabezado `x-amz-log-results` de la respuesta. Para obtener más información, consulte [Invoke \(p. 423\)](#).

Si utiliza la CLI de AWS para invocar la función, puede especificar el parámetro `--log-type parameter` con el valor `Tail` para recuperar la misma información.

- En CloudWatch Logs: para encontrar los logs en CloudWatch, necesita conocer el nombre del grupo de logs y el nombre del flujo de logs. Puede utilizar las propiedades `context.logGroupName` y

context.logStreamName en el código para obtener esta información. Cuando ejecute la función de Lambda, los logs resultantes en la consola o en la CLI le mostrarán el nombre del grupo de logs y el nombre del flujo de logs.

Errores de funciones (Python)

Si la función de Lambda genera una excepción, AWS Lambda reconoce el error y serializa la información de la excepción en JSON y la devuelve. Considere el siguiente ejemplo:

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

Cuando se invoque esta función de Lambda, generará una excepción y AWS Lambda devolverá el siguiente mensaje de error:

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

Observe que el rastro de stack se devuelve como la matriz JSON stackTrace de elementos de rastro de stack.

La forma de obtener la información de error depende del tipo de invocación que el cliente especifique en el momento de invocar la función:

- Si un cliente especifica el tipo de invocación RequestResponse (es decir, ejecución síncrona), se devuelve el resultado al cliente que ha realizado la llamada de invocación.

Por ejemplo, la consola siempre utiliza el tipo de invocación RequestResponse, por lo que mostrará el error en la sección Execution result, tal como se muestra a continuación:

The screenshot shows the AWS Lambda execution result page. At the top, there is a red circle with a question mark icon followed by the text "Execution result: failed (logs)". Below this, there is a "Details" section with a dropdown arrow. A note says "The area below shows the result returned by your function execution." A scrollable text area displays the following JSON error message:

```
"stackTrace": [
    [
        "/var/task/lambda_function.py",
        10,
        "lambda_handler",
        "print(\"value1 = \" + event['key1'])"
    ]
],
"errorType": "KeyError",
"errorMessage": "'key1'"
```

La misma información también se envía a CloudWatch y la sección Log output muestra los mismos logs.

The screenshot shows the AWS Lambda Function Overview page. On the left, there's a summary table with the following data:

| Summary | Log output |
|--|---|
| Code SHA-256 ebqf+Z07NundQwBxb hx6AtiKSFllbS1sgWsV HpEemvI= | 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 2204e873-6d8e-11e7-b475-1f3ceee7af69 69 | START RequestId: 2204e873-6d8e-11e7-b475-1f3ceee7af69 Version: \$LATEST value1 = value1 value2 = value2 value3 = value3 'Something went wrong': KeyError Traceback (most recent call last): File "/var/task/lambda_function.py", line 13, in lambda_handler return event['Something went wrong'] # Echo back the first key value KeyError: 'Something went wrong' |
| Duration 0.52 ms | |
| Billed duration 100 ms | |
| Resources configured 128 MB | |
| Max memory used 17 MB | |

- Si un cliente especifica el tipo de invocación `Event` (es decir, ejecución asíncrona), AWS Lambda no devolverá nada. En cambio, registrará la información de error en CloudWatch Logs. También puede consultar las métricas de errores en CloudWatch Metrics.

En función del origen de eventos, AWS Lambda puede volver a intentar la ejecución de la función de Lambda que ha fallado. Por ejemplo, si Kinesis es el origen de eventos, AWS Lambda reintentará la invocación que produce errores hasta que la función de Lambda se ejecute correctamente o hasta que los registros del flujo caduquen.

Para probar el código de Python anterior (consola)

1. En la consola, cree una función de Lambda mediante el proyecto hello-world. En runtime, elija Python 3.6 o Python 2.7. En Handler, sustituya `lambda_function.lambda_handler` por `lambda_function.always_failed_handler`. Para obtener instrucciones al respecto, consulte [Paso 2.1: Creación de una función Hello World de Lambda \(p. 203\)](#).
2. Sustituya el código de la plantilla por el código proporcionado en esta sección.
3. Pruebe la función de Lambda utilizando la plantilla Sample event template denominada Hello World proporcionada en la consola de Lambda.

Control de errores de las funciones

Puede crear un control de errores personalizado para generar una excepción directamente desde la función de Lambda y gestionarla directamente (Retry o Catch) en una máquina de estado de AWS Step Functions. Para obtener más información, consulte [Administración de las condiciones de error con una máquina de estado](#).

Considere que un `estado CreateAccount` es una `tarea` que escribe los datos de un cliente en una base de datos utilizando una función de Lambda.

- Si la tarea se realiza correctamente, se crea una cuenta y se envía un correo electrónico de bienvenida.

- Si un usuario intenta crear una cuenta para un nombre de usuario que ya existe, la función de Lambda genera un error, lo que hace que la máquina de estado sugiera otro nombre de usuario y vuelva a intentar el proceso de creación del cuenta.

En los siguientes códigos de muestra se ilustra cómo hacerlo. Tenga en cuenta que los errores personalizados en Python deben extender la clase `Exception`.

```
class AccountAlreadyExistsException(Exception): pass

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

Puede configurar Step Functions para que detecte el error utilizando una regla `Catch`. Lambda establece automáticamente el nombre del error en el nombre de clase sencilla de la excepción en tiempo de ejecución:

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

En tiempo de ejecución, AWS Step Functions detecta el error y realiza una [transición](#) al estado `SuggestAccountName`, según se especifica en la transición `Next`.

El control de errores personalizado facilita la creación de aplicaciones [sin servidor](#). Esta característica se integra con todos los lenguajes admitidos por el [Modelo de programación \(p. 8\)](#) de Lambda, lo que le permite diseñar la aplicación en los lenguajes de programación que elija, combinándolos a medida que avanza.

Para obtener más información acerca de cómo crear sus propias aplicaciones sin servidor utilizando AWS Step Functions y AWS Lambda, consulte [AWS Step Functions](#).

Modelo de programación para crear funciones de Lambda en C#

En las siguientes secciones se explica cómo se aplican los [conceptos fundamentales y los patrones de programación comunes](#) al crear código de funciones de Lambda en C#.

Temas

- [Controlador de funciones de Lambda \(C#\) \(p. 54\)](#)
- [El objeto context \(C#\) \(p. 58\)](#)
- [Registro \(C#\) \(p. 59\)](#)
- [Errores de funciones \(C#\) \(p. 60\)](#)

Además, tenga en cuenta que AWS Lambda proporciona lo siguiente:

- Amazon.Lambda.Core: esta biblioteca proporciona un registrador de Lambda estático, interfaces de serialización y un objeto context. El objeto Context ([El objeto context \(C#\) \(p. 58\)](#)) proporciona información del tiempo de ejecución acerca de la función de Lambda.
- Amazon.Lambda.Serialization.Json : es una implementación de la interfaz de serialización de Amazon.Lambda.Core.
- Amazon.Lambda.Logging.AspNetCore : proporciona una biblioteca para el registro desde ASP.NET.
- Objetos de evento (POCOs) para varios servicios de AWS, entre los que se incluyen:
 - Amazon.Lambda.APIGatewayEvents
 - Amazon.Lambda.CognitoEvents
 - Amazon.Lambda.ConfigEvents
 - Amazon.Lambda.DynamoDBEvents
 - Amazon.Lambda.KinesisEvents
 - Amazon.Lambda.S3Events
 - Amazon.Lambda.SNSEvents

Estos paquetes están disponibles en [Nuget Packages](#).

Controlador de funciones de Lambda (C#)

Cuando cree una función de Lambda debe especificar un controlador que AWS Lambda puede invocar cuando el servicio ejecute la función en su nombre.

Un controlador de función de Lambda se define como una instancia o un método estático en una clase. Si desea acceder al objeto context de Lambda, está disponible definiendo un parámetro de método de tipo ILambdaContext, una interfaz que puede utilizar para obtener acceso a información acerca de la ejecución actual, como el nombre de la función actual, la cantidad máxima de memoria, el tiempo que falta para finalizar la ejecución y el registro.

```
returnType handler-name(inputType input, ILambdaContext context) {  
    ...  
}
```

En la sintaxis, tenga en cuenta lo siguiente:

- **inputType**: el primer parámetro del controlador es su entrada, que puede estar formada por datos de eventos (publicados por un origen de eventos) o por una entrada personalizada proporcionada por el usuario, como una cadena o cualquier objeto de datos personalizado.
- **returnType**: si piensa invocar la función de Lambda de forma síncrona (utilizando el tipo de invocación RequestResponse), puede devolver la salida de la función utilizando cualquiera de los tipos de datos admitidos. Por ejemplo, si utiliza una función de Lambda como backend de aplicaciones móviles, la invocación se realiza de forma síncrona. El tipo de datos de salida se serializará en JSON.

Si piensa invocar la función de Lambda de forma asíncrona (utilizando el tipo de invocación Event), returnType debería ser void. Por ejemplo, si utiliza AWS Lambda con orígenes de eventos como Amazon S3 o Amazon SNS, estos orígenes de eventos invocan la función de Lambda mediante el tipo de invocación Event.

Control de flujos

De forma predeterminada, solo se admite el tipo System.IO.Stream como parámetro de entrada.

Por ejemplo, fíjese en el siguiente código de ejemplo de C#.

```
using System.IO;
{
    namespace Example

    public class Hello
    {
        public Stream MyHandler(Stream stream)
        {
            //function logic
        }
    }
}
```

En el código C# del ejemplo, el primer parámetro del controlador es la entrada al controlador (MyHandler), que puede estar formada por datos de eventos (publicados por un origen de eventos como Amazon S3) o por una entrada personalizada proporcionada por el usuario, tal como Stream (como sucede en este ejemplo) o cualquier objeto de datos personalizado. La salida es de tipo Stream.

Control de tipos de datos estándar

Para todos los demás tipos, que se indican a continuación, tendrá que especificar un serializador.

- Tipos primitivos de .NET (como string o int).
- Colecciones y mapas: IList, IEnumerable, IList<T>, Array, IDictionary, IDictionary<TKey, TValue>
- Tipos POCO (objetos CLR estándar simples, o Plain old CLR objects)
- Tipos de eventos predefinidos de AWS
- Para las invocaciones asíncronas, Lambda ignorará el tipo de retorno. El tipo de retorno se puede establecer en void en estos casos.
- Si está utilizando la programación asíncrona de .NET, el tipo de retorno puede ser Task y Task<T>, y utilizar las palabras clave async y await keywords. Para obtener más información, consulte [Uso de async en funciones C# con AWS Lambda \(p. 57\)](#).

A menos que los parámetros de entrada y salida de la función sean del tipo System.IO.Stream, deberá serializarlos. AWS Lambda proporciona un serializador predeterminado que se puede aplicar en el nivel de ensamblado o de método de la aplicación, o puede definir uno implementando la interfaz ILambdaSerializer proporcionada por la biblioteca Amazon.Lambda.Core. Para obtener más información, consulte [Creación de un paquete de implementación \(C#\) \(p. 65\)](#).

Para añadir el atributo serializador predeterminado a un método, primero añada una dependencia de Amazon.Lambda.Serialization.Json en el archivo project.json.

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

El ejemplo siguiente ilustra la flexibilidad que puede obtenerse especificando el serializador predeterminado de Json.NET en un método y otro elegido por el usuario en otro método:

```
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);
    }
}
```

Signaturas para los controladores

Cuando cree funciones de Lambda, debe proporcionar una cadena de controlador que indique a AWS Lambda dónde debe buscar el código que va a invocar. En C#, tiene este formato:

ENSAMBLADO::TIPO::MÉTODO, donde:

- **ENSAMBLADO** es el nombre del archivo de ensamblado de .NET para la aplicación. Cuando utilice la CLI de .NET Core para compilar la aplicación, si no ha establecido el nombre del ensamblado con la opción buildOptions.outputName en project.json, el nombre de **ENSAMBLADO** será el nombre de la carpeta que contiene el archivo project.json. Para obtener más información, consulte [CLI de .NET Core \(p. 65\)](#). En este caso, supongamos que el nombre de la carpeta es HelloWorldApp.
- **TIPO** es el nombre completo del tipo de controlador, que consta del **EspacioDeNombres** y del **NombreDeLaClase**. En este caso Example.Hello.
- **MÉTODO** es el nombre del controlador de la función, en este caso, MyHandler.

En definitiva, la firma tendrá este formato:

Ensamblado::EspacioDeNombres.NombreDeLaClase::NombreDelMétodo.

Ahora, considere el siguiente ejemplo:

```
using System.IO;
{
    namespace Example

    public class Hello
    {
        public Stream MyHandler(Stream stream)
        {
            //function logic
        }
    }
}
```

La cadena del controlador sería: HelloWorldApp::Example.Hello::MyHandler.

Para obtener instrucciones acerca de cómo crear una función de Lambda utilizando este código C#, consulte [Paso 2.4: Creación de una función de Lambda escrita en C# \(opcional\) \(p. 210\)](#).

Important

Si el método especificado en la cadena de controlador está sobrecargado, debe proporcionar la firma exacta del método que Lambda debe invocar. AWS Lambda rechazará una firma que de otra forma sería válida si la resolución requeriría seleccionar entre varias firmas (sobrecargadas).

Restricciones del controlador de funciones de Lambda

Tenga en cuenta que existen algunas restricciones que afectan a la signatura del controlador:

- No puede ser `unsafe` ni utilizar tipos de puntero en la signatura del controlador, aunque el contexto `unsafe` puede utilizarse dentro del método del controlador y sus dependencias. Para obtener más información, consulte [unsafe \(Referencia de C#\)](#).
- No puede pasar un número variable de parámetros utilizando la palabra clave `params`, ni utilizar `ArgIterator` como parámetro de entrada o de retorno que se utiliza para admitir un número variable de parámetros.
- El controlador no puede ser un método genérico (por ejemplo, `IList<T> Sort<T>(IList<T> input)`).
- No se admiten los controladores asíncronos con la signatura `async void`.

Uso de `async` en funciones C# con AWS Lambda

Si sabe que la función de Lambda requerirá un proceso de larga duración, como cargar archivos grandes en Amazon S3 o leer una gran cantidad de registros desde DynamoDB, puede utilizar el patrón `async/await`. Si crea un controlador con esta signatura, Lambda ejecutará la función de forma síncrona y esperará un máximo de 5 minutos para que se complete la ejecución antes de volver o de agotar el tiempo de espera. Por ejemplo:

```
public async Task<Response> ProcessS3ImageResizeAsync(SimpleS3Event input)
{
    var response = await client.DoAsyncWork(input);
    return response;
}
```

Si utiliza este patrón, debe tener en cuenta algunas consideraciones:

- AWS Lambda no admite métodos `async void`.
- Si crea una función de Lambda asíncrona sin implementar el operador `await`, .NET emitirá una advertencia de compilación y se producirá un comportamiento inesperado. Por ejemplo, algunas acciones asíncronas se ejecutarán, mientras que otras no. O algunas acciones asíncronas no terminarán antes de que finalice la ejecución de la función.

```
public async Task ProcessS3ImageResizeAsync(SimpleS3Event event) // Compiler warning
{
    client.DoAsyncWork(input);
}
```

- La función de Lambda puede incluir varias llamadas asíncronas, que pueden invocarse en paralelo. Puede utilizar los métodos `Task.WaitAll` y `Task.WaitAny` para trabajar con varias tareas. Para utilizar el método `Task.WaitAll`, pase una lista de las operaciones en una matriz al método. En el ejemplo siguiente, tenga en cuenta que si olvida incluir cualquier operación en la matriz, la llamada puede volver antes de que finalice su operación.

```
public async Task SaveAsync(Profile profile)
{
    var s3Save = s3.SaveImage(profile.image);
    var ddbSave = ddb.SaveAttributes(profile.Attributes);
    var ddbSave2 = ddb.SaveConnections(profile.connections); // Lambda will return before
    // this call completes
                                         // No compiler warnings
    return await Task.WaitAll(new Task[]{ s3Save, ddbSave }); // Did not "await" for
    ddbSave2
}
```

Para utilizar el método `Task.WaitAny`, también se pasa una lista de las operaciones en una matriz al método. La llamada vuelve tan pronto como finaliza la primera operación, incluso si las demás siguen ejecutándose.

```
public async Task<SearchResult> SearchAsync(Query q)
{
    var siteSearch1 = site1.SearchAsync(q);
    var siteSearch2 = site2.SearchAsync(q);
    var siteSearch3 = site3.SearchAsync(q);
    var tasks[] = new Task[]{siteSearch1, siteSearch2, siteSearch3};

    var index = await Task.WaitAny(tasks); // Returns as soon as any of the tasks complete,
    other task may run in background
    return tasks[index].Result;
}
```

No recomendamos utilizar `Task.WaitAny` para los motivos anteriores.

El objeto context (C#)

Puede obtener información útil sobre cómo interactúa la función de Lambda con el tiempo de ejecución de AWS Lambda añadiendo el parámetro `ILambdaContext` al método. Al hacerlo, AWS Lambda proporciona detalles del tiempo de ejecución, como el flujo de logs de CloudWatch asociado a la función o el identificador del cliente que llamó a las funciones, al que se accede a través de las propiedades proporcionadas por el objeto `context`.

Para ello, cree un método con la siguiente firma:

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

Las propiedades del objeto `context` son:

- `MemoryLimitInMB`: cantidad máxima de memoria, en MB, configurada para la función de Lambda.
- `FunctionName`: nombre de la función de Lambda que se está ejecutando.
- `FunctionVersion`: la versión de la función de Lambda que se está ejecutando. Si se utiliza un alias para invocar la función, `FunctionVersion` será la versión a la que apunta el alias.
- `InvokedFunctionArn`: el ARN utilizado para invocar esta función. Puede ser el ARN de la función o el ARN del alias. Un ARN incompleto ejecuta la versión `$LATEST` y los alias ejecutan la versión de la función a la que apunta.
- `AwsRequestId`: el ID de la solicitud de AWS asociado a la solicitud. Es el ID devuelto al cliente que invocó esta función de Lambda. Puede utilizar el ID de la solicitud para cualquier consulta de seguimiento con AWS Support. Tenga en cuenta que si AWS Lambda reintenta la función (por ejemplo, en una situación en la que la función de Lambda que está procesando registros de Kinesis genera una excepción), el ID de la solicitud sigue siendo el mismo.
- `LogStreamName`: el nombre del flujo de logs de CloudWatch para la ejecución de la función de Lambda en concreto. Puede ser null si el usuario de IAM proporcionado no tiene permiso para realizar acciones de CloudWatch.
- `LogGroupName`: el nombre del grupo de logs de CloudWatch asociado a la función de Lambda invocada. Puede ser null si el usuario de IAM proporcionado no tiene permiso para realizar acciones de CloudWatch.
- `ClientContext`: información acerca del dispositivo y la aplicación cliente cuando se invoca a través de AWS Mobile SDK. Puede ser null. `ClientContext` proporciona información del cliente, como el ID del cliente, el título de la aplicación, el nombre de la versión, el código de la versión y el nombre del paquete de la aplicación.

- **Identity:** información acerca del proveedor de identidad de Amazon Cognito cuando se invoca a través de AWS Mobile SDK. Puede ser null.
- **RemainingTime:** tiempo que falta para terminar la ejecución de la función, en milisegundos. En el momento en que se crea la función de Lambda, se establece el límite de tiempo máximo, momento en el que AWS Lambda terminará la ejecución de la función. La información acerca del tiempo restante para la ejecución de la función se puede utilizar para especificar el comportamiento de la función al acercarse al tiempo de espera. Este es un campo de tipo `TimeSpan`.
- **Logger:** el registrador de Lambda asociado al objeto `ILambdaContext`. Para obtener más información, consulte [Registro \(C#\) \(p. 59\)](#).

El siguiente fragmento de código C# contiene una función de controlador sencilla que muestra el valor del parámetro de entrada `y`, a continuación, imprime parte de la información de contexto.

```
public async Task Handler(ILambdaContext context)
{
    Console.WriteLine("Function name: " + context.FunctionName);
    Console.WriteLine("RemainingTime: " + context.RemainingTime);
    await Task.Delay(TimeSpan.FromSeconds(0.42));
    Console.WriteLine("RemainingTime after sleep: " + context.RemainingTime);
}
```

Registro (C#)

La función de Lambda puede contener instrucciones de registro y, a su vez, AWS Lambda escribe estos logs en CloudWatch Logs.

En el modelo de programación de C#, existen tres formas de registrar los datos de una función:

- Utilizar los métodos estáticos `Write` o `WriteLine` proporcionados por la clase `Console` de C#. Cualquier cosa que se escriba en `stdout` o `stderr`, utilizando `Console.Write` o un método similar, será registrada en CloudWatch Logs.

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

- Utilizar el método `Log` de la clase `Amazon.Lambda.Core.LambdaLogger`. Es una clase estática que se puede utilizar en cualquier lugar de la aplicación. Para utilizarla, debe incluir la biblioteca `Amazon.Lambda.Core` .

```
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);
    }
}
```

Cada llamada a `LambdaLogger.Log` da como resultado un evento de CloudWatch Logs, siempre que el tamaño del evento esté dentro de los límites permitidos. Para obtener información acerca de los

Límites de CloudWatch Logs, consulte [Límites de CloudWatch Logs](#) en la Guía del usuario de Amazon CloudWatch.

- Utilizar el registrador de `ILambdaContext`. El objeto `ILambdaContext` (si se especifica) del método contiene una propiedad `Logger` que representa un `LambdaLogger`. A continuación se muestra un ejemplo del uso de este método:

```
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);
    }
}
```

Dónde encontrar los logs

Puede encontrar los logs que escribe la función de Lambda como se indica a continuación:

- Busque los logs en CloudWatch Logs. El objeto `ILambdaContext` proporciona las propiedades `LogStreamName` y `LogGroupName`. Utilice estas propiedades para encontrar el flujo de logs específico donde se escriben los logs.
- Si invoca una función de Lambda a través de la consola, el tipo de invocación es siempre `RequestResponse` (es decir, ejecución síncrona) y la consola muestra los logs que escribe la función de Lambda mediante el objeto `LambdaLogger`. AWS Lambda también devuelve logs de los métodos `Console.WriteLine` y `Console.WriteLine`.
- Si se invoca una función de Lambda mediante programación, se puede añadir el parámetro `LogType` para recuperar los últimos 4 KB de datos de log escritos en CloudWatch Logs. Para obtener más información, consulte [Invoke \(p. 423\)](#). AWS Lambda devuelve esta información de log en el encabezado `x-amz-log-results` de la respuesta. Si utiliza la AWS Command Line Interface para invocar la función, puede especificar el parámetro `--log-type` con el valor `Tail`.

Errores de funciones (C#)

Cuando se produzca una excepción en una función de Lambda, Lambda le proporcionará información sobre la excepción. Las excepciones pueden producirse en dos lugares distintos:

- Inicialización (cuando Lambda carga el código, valida la cadena de controlador y crea una instancia de la clase, si no es estática).
- La invocación de la función de Lambda.

La información serializada sobre la excepción se devuelve como la carga de un objeto JSON modelado, y se registra en los logs de CloudWatch.

En la fase de inicialización, pueden producirse excepciones para cadenas de controlador no válidas, métodos o tipos que no cumplen las reglas (consulte [Restricciones del controlador de funciones de Lambda \(p. 57\)](#)) o cualquier otro método de validación (como olvidar el atributo serializador y utilizar POCO como tipo de entrada o salida). Estas excepciones son de tipo `LambdaException`. Por ejemplo:

```
{
    "errorType": "LambdaException",
    "errorMessage": "Invalid lambda function handler: 'http://this.is.not.a.valid.handler/'.
                    The valid format is 'ASSEMBLY::TYPE::METHOD'."}
```

Si el constructor genera una excepción, el error también es del tipo `LambdaException`, pero la excepción generada durante la construcción se proporciona en la propiedad `cause`, que es un objeto de excepción modelado:

```
{
  "errorType": "LambdaException",
  "errorMessage": "An exception was thrown when the constructor for type 'LambdaExceptionTestFunction.ThrowExceptionInConstructor' was invoked. Check inner exception for more details.",
  "cause": {
    "errorType": "TargetInvocationException",
    "errorMessage": "Exception has been thrown by the target of an invocation.",
    "stackTrace": [
      "at System.RuntimeTypeHandle.CreateInstance(RuntimeType type, Boolean publicOnly, Boolean noCheck, Boolean&canBeCached, RuntimeMethodHandleInternal&ctor, Boolean& bNeedSecurityCheck)",
      "at System.RuntimeType.CreateInstanceSlow(Boolean publicOnly, Boolean skipCheckThis, Boolean fillCache, StackCrawlMark& stackMark)",
      "at System.Activator.CreateInstance(Type type, Boolean nonPublic)",
      "at System.Activator.CreateInstance(Type type)"
    ],
    "cause": {
      "errorType": "ArithmaticException",
      "errorMessage": "Sorry, 2 + 2 = 5",
      "stackTrace": [
        "at LambdaExceptionTestFunction.ThrowExceptionInConstructor..ctor()"
      ]
    }
  }
}
```

Como muestra el ejemplo, las excepciones internas siempre se conservan (como la propiedad `cause`) y pueden anidarse profundamente.

Las excepciones también pueden producirse durante la invocación. En este caso, se conserva el tipo de la excepción y esta se devuelve directamente como carga y en los logs de CloudWatch. Por ejemplo:

```
{
  "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.GetResponse()",
        "at LambdaDemo107.LambdaEntryPoint.<CheckWebsiteStatus>d__0.MoveNext()"
      ]
    }
  }
}
```

```
        "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)"
    ]
}
}
}
```

El método con el que se comunica la información del error depende del tipo de invocación:

- Tipo de invocación `RequestResponse` (es decir, ejecución síncrona): en este caso, se obtiene el mensaje de error.

Por ejemplo, si invoca una función de Lambda utilizando la consola de Lambda, `RequestResponse` siempre es el tipo de invocación, y la consola muestra la información de error devuelta por AWS Lambda en la sección `Execution result` de la consola.

- Tipo de invocación `Event` (es decir, ejecución asíncrona): en este caso, AWS Lambda no devuelve nada. En su lugar, registra la información de error en las métricas de CloudWatch y CloudWatch Logs.

En función del origen de eventos, AWS Lambda puede volver a intentar la ejecución de la función de Lambda que ha fallado. Para obtener más información, consulte [Reintentos si se producen errores \(p. 192\)](#).

Control de errores de las funciones

Puede crear un control de errores personalizado para generar una excepción directamente desde la función de Lambda y gestionarla directamente (Retry o Catch) en una máquina de estado de AWS Step Functions. Para obtener más información, consulte [Administración de las condiciones de error con una máquina de estado](#).

Considere que un `estado CreateAccount` es una `tarea` que escribe los datos de un cliente en una base de datos utilizando una función de Lambda.

- Si la tarea se realiza correctamente, se crea una cuenta y se envía un correo electrónico de bienvenida.
- Si un usuario intenta crear una cuenta para un nombre de usuario que ya existe, la función de Lambda genera un error, lo que hace que la máquina de estado sugiera otro nombre de usuario y vuelva a intentar el proceso de creación del cuenta.

En los siguientes códigos de muestra se ilustra cómo hacerlo. Tenga en cuenta que los errores personalizados en C# deben extender la clase `Exception`.

```
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!");
        }
    }
}
```

Puede configurar Step Functions para que detecte el error utilizando una regla `Catch`. Lambda establece automáticamente el nombre del error en el nombre de clase sencilla de la excepción en tiempo de ejecución:

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

En tiempo de ejecución, AWS Step Functions detecta el error y realiza una [transición](#) al estado `SuggestAccountName`, según se especifica en la transición `Next`.

El control de errores personalizado facilita la creación de aplicaciones [sin servidor](#). Esta característica se integra con todos los lenguajes admitidos por el [Modelo de programación \(p. 8\)](#) de Lambda, lo que le permite diseñar la aplicación en los lenguajes de programación que elija, combinándolos a medida que avanza.

Para obtener más información acerca de cómo crear sus propias aplicaciones sin servidor utilizando AWS Step Functions y AWS Lambda, consulte [AWS Step Functions](#).

Creación de un paquete de implementación

Para crear una función de Lambda, primero debe crear un paquete de implementación de la función de Lambda, que es un archivo .zip o .jar que contiene el código y las dependencias. Al crear el archivo zip, incluya únicamente el código y sus dependencias, no la carpeta donde se encuentran.

- [Creación de un paquete de implementación \(Node.js\) \(p. 64\)](#)
- [Creación de un paquete de implementación \(Java\) \(p. 70\)](#)
- [Creación de un paquete de implementación \(C#\) \(p. 65\)](#)
- [Creación de un paquete de implementación \(Python\) \(p. 77\)](#)

Creación de un paquete de implementación (Node.js)

Para crear una función de Lambda, primero debe crear un paquete de implementación de la función de Lambda, que es un archivo .zip que contiene el código y las dependencias.

Puede crear un paquete de implementación o escribir el código directamente en la consola de Lambda, en cuyo caso la consola crea el paquete de implementación automáticamente y lo carga, con lo que se crea la función de Lambda. Tenga en cuenta lo siguiente para determinar si puede utilizar la consola para crear una función de Lambda:

- Escenario sencillo: si el código personalizado solo necesita la biblioteca del SDK de AWS, puede utilizar el editor de código integrado de la consola de AWS Lambda. Con la consola, puede editar y cargar el código en AWS Lambda. La consola comprimirá el código con la información de configuración relevante en un paquete de implementación que el servicio de Lambda puede ejecutar.

También puede probar el código en la consola invocándolo manualmente y utilizando los datos del evento de muestra.

Note

El servicio de Lambda tiene preinstalado AWS SDK para Node.js.

- Escenario avanzado: si escribe código que utiliza otros recursos, como, por ejemplo, una biblioteca de gráficos para el procesamiento de imágenes, o desea utilizar la CLI de AWS en lugar de la consola, primero debe crear el paquete de implementación de la función de Lambda y, a continuación, utilizar la consola o la CLI para cargar el paquete.

Note

Después de crear un paquete de implementación, puede cargarlo directamente o cargar primero el archivo .zip en un bucket de Amazon S3 de la misma región de AWS en la que desea crear la función de Lambda y, a continuación, especificar el nombre del bucket y el nombre de la clave de objeto cuando crear la función de Lambda mediante la consola o la CLI de AWS.

A continuación se muestra un ejemplo de procedimiento para crear un paquete de implementación (fuera de la consola). Supongamos que desea crear un paquete de implementación que incluye un archivo de código filename.js, y que el código utiliza la biblioteca `async`.

1. Abra un editor de texto y escriba el código. Guarde el archivo (por ejemplo, `filename.js`). Utilizará el nombre de archivo para especificar el controlador en el momento de crear la función de Lambda.
2. En el mismo directorio, utilice npm para instalar las bibliotecas de las que depende el código. Por ejemplo, si el código utiliza la biblioteca `async`, utilice el siguiente comando npm.

```
npm install async
```

3. El directorio tendrá la siguiente estructura:

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

4. Comprima el contenido de la carpeta que constituye el paquete de implementación (por ejemplo, `sample.zip`).

A continuación, especifique el nombre del archivo .zip como paquete de implementación en el momento de crear la función de Lambda.

Si desea incluir sus propios archivos binarios, incluidos los nativos, añádalos al archivo Zip que va a cargar y, a continuación, haga referencia a ellos (incluida la ruta relativa dentro del archivo Zip que ha creado) cuando los llame desde Node.js o desde otros procesos que haya iniciado previamente. Asegúrese de incluir lo siguiente al principio del código de la función: `process.env['PATH'] = process.env['PATH'] + ':' + process.env['LAMBDA_TASK_ROOT']`

Para obtener más información acerca de cómo incluir archivos binarios nativos en un paquete de función de Lambda, consulte [Running Executables in AWS Lambda](#).

Creación de un paquete de implementación (C#)

Puede crear aplicaciones de AWS Lambda basadas en .NET Core y empaquetarlas para su implementación de las siguientes formas:

- Utilice la CLI de .NET Core, que puede descargar [aquí](#) para crear la aplicación de Lambda.
- Utilice el complemento de Lambda para AWS ToolKit for Microsoft Visual Studio, que puede descargar [aquí](#).

Temas

- [CLI de .NET Core \(p. 65\)](#)
- [AWS Toolkit for Visual Studio \(p. 68\)](#)

CLI de .NET Core

La CLI de .NET Core ofrece una manera multiplataforma de crear aplicaciones de Lambda basadas en .NET.

Antes de empezar

En esta sección se presupone que ha hecho lo siguiente:

- Instalar la CLI de .NET Core. Si todavía no lo ha hecho, hágalo [aquí](#).

Creación de un proyecto de .NET

Para crear una aplicación mediante la CLI de .NET Core, abra un símbolo del sistema, vaya a la carpeta en la que ha instalado el ejecutable de .NET Core y siga estos pasos:

1. Cree el directorio donde se guardará el proyecto utilizando el siguiente comando: `mkdir ejemplo`
2. Vaya a ese directorio con el siguiente comando: `cd ejemplo`
3. Escriba el siguiente comando: `dotnet new console`

Esto creará dos archivos en el directorio `ejemplo`:

- `Program.cs`, que es donde deberá escribir la función de Lambda.
- `project.json`, que es el archivo en el que se declaran las dependencias de NuGet (o las dependencias de proyectos locales). NuGet es el administrador de paquetes de la plataforma .NET. Para obtener más información, consulte [Nuget.org](#).

Note

Los métodos de Lambda no utilizan el punto de entrada `Main()` proporcionado de forma predeterminada en .NET; por lo tanto, abra el archivo `project.json` y elimine la propiedad "buildOptions". Después de hacer esto, su archivo `project.json` tendrá un aspecto similar a este (la versión exacta dependerá de cuándo se instaló la CLI de .Net Core):

```
{  
  "version": "1.0.0-*",  
  "dependencies": {},  
  "frameworks": {  
    "netcoreapp1.0": {  
      "dependencies": {  
        "Microsoft.NETCore.App": {  
          "type": "platform",  
          "version": "1.1.0"  
        }  
      },  
      "imports": "dnxcore50"  
    }  
  }  
}
```

4. Abra el archivo `Program.cs` con el editor que desee, como Microsoft Visual Studio.
- Sustituya el código predeterminado que se proporciona por el código del controlador de la función de Lambda:

En este punto, la estructura del archivo `.cs` debería ser similar a la siguiente:

```
using System;  
using System.IO;  
  
namespace CSharpLambdaFunction  
{  
  public class LambdaHandler  
  {  
    public Stream myHandler(Stream inputStream)  
    {  
      //function logic  
    }  
  }  
}
```

```
    }  
}
```

La firma del controlador de la función de Lambda debería tener el formato **Ensamblando : EspacioDeNombres.NombreDeClase : NombreDeMétodo**. Para obtener más información, consulte [Signaturas para los controladores \(p. 56\)](#).

Uso de un serializador

Para las funciones de Lambda que utilizan tipos de entrada o salida distintos de un objeto `Stream`, tendrá que añadir una biblioteca de serialización a la aplicación. Puede hacerlo de las siguientes maneras:

- Utilizando Json.NET. Lambda proporcionará una implementación para el serializador de JSON utilizando JSON.NET como un paquete de NuGet.
- Creando su propia biblioteca de serialización mediante la implementación de la interfaz `ILambdaSerializer`, que está disponible formando parte de la biblioteca `Amazon.Lambda.Core`. La interfaz define dos métodos:
 - `T Deserialize<T>(Stream requestStream);`

Puede implementar este método para deserializar la carga de solicitud desde la API `Invoke` en el objeto que se pasa al controlador de la función de Lambda.

- `T Serialize<T>(T response, Stream responseStream);`

Puede implementar este método para serializar el resultado que devuelve el controlador de la función de Lambda en la carga de respuesta que devuelve la API `Invoke`.

Puede utilizar cualquier serializador que deseé añadiéndolo como una dependencia al archivo `project.json`.

```
{  
  "version": "1.0.0-*",  
  "buildOptions": {  
  },  
  
  "dependencies": {  
    "Microsoft.NETCore.App": {  
      "type": "platform",  
      "version": "1.0.1"  
    },  
  
    "Newtonsoft.Json": "9.0.1",  
  
    "Amazon.Lambda.Core": "1.0.0*",  
    "Amazon.Lambda.Serialization.Json": "1.0.0",  
  
    "Amazon.Lambda.Tools" : {  
      "type" :"build",  
      "version": "0.9.0-preview1"  
    },  
  },  
  
  "tools": {  
    "Amazon.Lambda.Tools" : "0.9.0-preview1"  
  },  
  
  "frameworks": {  
    "netcoreapp1.0": {  
      "imports": "dnxcore50"  
    }  
  }  
}
```

```
    }  
}
```

Seguidamente, deberá añadirlo a su archivo AssemblyInfo.cs. Por ejemplo, si utiliza el serializador predeterminado de Json.NET, esto es lo que debería añadir:

```
[assembly:LambdaSerializer(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]
```

Note

Puede definir un atributo de serialización personalizado en el nivel de método, que sustituirá al serializador predeterminado especificado en el nivel de ensamblado. Para obtener más información, consulte [Control de tipos de datos estándar \(p. 55\)](#).

Creación del paquete de implementación

Para crear el paquete de implementación, abra un símbolo del sistema, vaya a la carpeta que contiene el archivo `project.json` y ejecute los siguientes comandos:

- `dotnet restore` para restablecer las referencias a las dependencias del proyecto que hayan cambiado durante el proceso de desarrollo.
- `dotnet publish` para compilar la aplicación y empaquetar el código fuente y las dependencias en una carpeta. La salida de la ventana de comandos le indicará dónde se ha creado la carpeta. Por ejemplo:

```
publish: Published to C:\Users\yourname\project-folder\bin\debug\netcoreapp1.1\publish
```

El contenido de esta carpeta representa la aplicación y, como mínimo, debería contener lo siguiente:

nombre-de-la-aplicación.deps.json
nombre-de-la-aplicación.dll
nombre-de-la-aplicación.pdb
nombre-de-la-aplicación.runtimeconfig.json

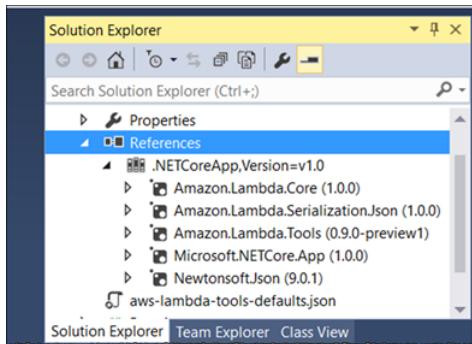
Comprima el contenido de la carpeta (no la propia carpeta). Este es el paquete de implementación.

AWS Toolkit for Visual Studio

Puede crear aplicaciones de Lambda basadas en .NET mediante el complemento de Lambda para AWS Toolkit for Visual Studio. El plugin está disponible formando parte de un paquete NuGet.

Paso 1: Creación y compilación de un proyecto

1. Lance Microsoft Visual Studio y elija New project.
 - a. En el menú File, elija New, seguido de Project.
 - b. En la ventana New Project, elija AWS Lambda Project (.NET Core), seguido de OK.
 - c. En la ventana Select Blueprint, dispondrá de una serie de aplicaciones de ejemplo con código de muestra para comenzar a crear una aplicación de Lambda basada en .NET.
 - d. Para crear una aplicación de Lambda desde cero, elija Blank Function, seguido de Finish.
 - e. Tenga en cuenta que las bibliotecas necesarias para crear una aplicación de Lambda basada en .NET se proporcionan en el nodo References del proyecto.



2. Abra el archivo Function.cs. Se le proporcionará una plantilla para implementar el código del controlador de la función de Lambda.

```
using System;
using Amazon.Lambda.Core;
using Amazon.Lambda.Serialization;

// Assembly attribute to enable the Lambda function's JSON input to be converted into a .NET class.
[assembly: LambdaSerializerAttribute(typeof(Amazon.Lambda.Serialization.Json.JsonSerializer))]

namespace AWSLambda
{
    public class LambdaFunction
    {

        /// <summary>
        /// A simple function that takes a string and does a ToUpper
        /// </summary>
        /// <param name="input"></param>
        /// <param name="context"></param>
        /// <returns></returns>
        public string FunctionHandler(string input, ILambdaContext context)
        {
            return input?.ToUpper();
        }
    }
}
```

3. Una vez que haya escrito el código que representa la función de Lambda, puede cargarlo haciendo clic con el botón derecho del ratón en el nodo Project de la aplicación y, a continuación, eligiendo Publish to AWS Lambda.
4. En la ventana Upload Lambda Function, haga lo siguiente:

- Especifique la región en el campo Region
- Especifique el nombre de la función en el campo Function Name
- Especifique el nombre del ensamblado en el campo Assembly Name
- Especifique el nombre del tipo en el campo Type Name
- Especifique el nombre del método en el campo Method Name

A continuación, elija Next

5. En la ventana Advanced Function Details, haga lo siguiente:

- En el campo Role Name, especifique el nombre del rol de IAM necesario para la ejecución de la función de Lambda. Si todavía no ha creado un rol de ejecución, haga lo siguiente:
 1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.
 2. Siga los pasos de [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM para crear un rol de IAM (rol de ejecución). Cuando siga los pasos para crear un rol, tenga en cuenta lo siguiente:
 - En Role Name, utilice un nombre que sea exclusivo dentro de su cuenta de AWS.

- En Select Role Type, elija AWS Service Roles y, a continuación, seleccione un rol de servicio que conceda a dicho servicio permisos para asumir el rol.
- En Attach Policy, elija una política de permisos adecuada para ejecutar la función de Lambda.
- (Opcional) En Environment, especifique las variables de entorno que desee utilizar. Para obtener más información, consulte [Variables de entorno \(p. 97\)](#).
- (Opcional) Especifique valores para las opciones Memory (MB) o Timeout (Secs).
- (Opcional) Especifique el valor de VPC si la función de Lambda necesita acceder a recursos que se ejecutan dentro de una VPC privada. Para obtener más información, consulte [Configuración de una función de Lambda para acceder recursos en una Amazon VPC \(p. 108\)](#).
- Elija Next seguido de Upload para implementar la aplicación.

Creación de un paquete de implementación (Java)

El paquete de implementación puede ser un archivo .zip o .jar independiente. Puede elegir el que deseé. Puede utilizar cualquier herramienta de compilación y compresión con la que esté familiarizado para crear un paquete de implementación.

Dispone de ejemplos donde se utiliza Maven para crear archivos jar independientes y ejemplos donde se utiliza Gradle para crear un archivo .zip. Para obtener más información, consulte los siguientes temas:

Temas

- [Creación de un paquete de implementación .jar utilizando Maven sin ningún IDE \(Java\) \(p. 70\)](#)
- [Creación de un paquete de implementación .jar utilizando Maven y el IDE de Eclipse \(Java\) \(p. 72\)](#)
- [Creación de un paquete de implementación .zip \(Java\) \(p. 75\)](#)
- [Creación de funciones de Lambda mediante el IDE de Eclipse y el complemento del SDK de AWS \(Java\) \(p. 77\)](#)

Creación de un paquete de implementación .jar utilizando Maven sin ningún IDE (Java)

Esta sección muestra cómo empaquetar el código Java en un paquete de implementación utilizando Maven en la línea de comandos.

Temas

- [Antes de empezar \(p. 70\)](#)
- [Información general sobre la estructura del proyecto \(p. 71\)](#)
- [Paso 1: Creación de un proyecto \(p. 71\)](#)
- [Paso 2: Compilación del proyecto \(creación de un paquete de implementación\) \(p. 72\)](#)

Antes de empezar

Deberá instalar la herramienta de compilación de línea de comandos Maven. Para obtener más información, vaya a [Maven](#). Si está utilizando Linux, verifique el administrador de paquetes.

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

Si está utilizando Homebrew

```
brew install maven
```

Información general sobre la estructura del proyecto

Después de configurar el proyecto, debe tener la siguiente estructura de carpetas:

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

El código estará en la carpeta /java. Por ejemplo, si el nombre del paquete es example y contiene la clase Hello.java, la estructura será:

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

Después de compilar el proyecto, el archivo .jar obtenido (es decir, el paquete de implementación) estará en el subdirectorio *project-dir*/target.

Paso 1: Creación de un proyecto

Siga los pasos de esta sección para crear un proyecto de Java.

1. Cree el directorio del proyecto (*project-dir*).
2. En el directorio *project-dir*, cree lo siguiente:
 - Archivo Project Object Model, pom.xml. Añada la información del proyecto y los detalles de configuración siguientes para que Maven compile el proyecto.

```
<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>
```

```
</plugin>
</plugins>
</build>
</project>
```

Note

- En la sección `dependencies`, el `groupId` (es decir, `com.amazonaws`) es el ID del grupo de Amazon AWS para los artefactos de Maven en el repositorio central de Maven. El `artifactId` (es decir, `aws-lambda-java-core`) es la biblioteca principal de AWS Lambda que proporciona definiciones de las interfaces `RequestHandler`, `RequestStreamHandler` y `Context` de AWS Lambda para su uso en la aplicación de Java. En el momento de la compilación, Maven resuelve estas dependencias.
- En la sección de complementos, Apache `maven-shade-plugin` es un complemento que Maven descargará y utilizará durante el proceso de compilación. Este complemento se utiliza para empaquetar varios jar para crear un archivo .jar independiente (un archivo .zip) que será el paquete de implementación.
- Si está siguiendo otros temas del tutorial de esta guía, es posible que los tutoriales específicos le soliciten añadir más dependencias. Asegúrese de añadir las dependencias que sean necesarias.

3. En `project-dir`, cree la siguiente estructura:

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

4. En el subdirectorio `/java`, añada los archivos de Java y la estructura de carpetas, si los hay. Por ejemplo, si el nombre del paquete de Java es `example` y el código fuente es `Hello.java`, la estructura de directorios tiene este aspecto:

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

Paso 2: Compilación del proyecto (creación de un paquete de implementación)

Ahora puede compilar el proyecto utilizando Maven en la línea de comandos.

1. En un símbolo del sistema, cambie al directorio del proyecto (`project-dir`).
2. Ejecute el siguiente comando `mvn` para compilar el proyecto:

```
$ mvn package
```

El .jar resultante se guarda como `project-dir/target/lambda-java-example-1.0-SNAPSHOT.jar`. El nombre del .jar se crea concatenando los valores de `artifactId` y `version` del archivo `pom.xml`.

La compilación crea el .jar resultante, utilizando información contenida en `pom.xml` para hacer las transformaciones necesarias. Este es un archivo .jar independiente (archivo .zip) que incluye todas las dependencias. Este es el paquete de implementación que puede cargar en AWS Lambda para crear una función de Lambda.

Creación de un paquete de implementación .jar utilizando Maven y el IDE de Eclipse (Java)

Esta sección muestra cómo empaquetar el código Java en un paquete de implementación utilizando el IDE de Eclipse y el complemento Maven para Eclipse.

Temas

- [Antes de empezar \(p. 73\)](#)
- [Paso 1: Creación y compilación de un proyecto \(p. 73\)](#)

Antes de empezar

Instale el complemento Maven para Eclipse.

1. Inicie Eclipse. En el menú Help de Eclipse, elija Install New Software.
2. En la ventana Install, escriba `http://download.eclipse.org/technology/m2e/releases` en el cuadro Work with: y elija Add.
3. Siga los pasos para completar la configuración.

Paso 1: Creación y compilación de un proyecto

En este paso, iniciará Eclipse y creará un proyecto de Maven. Añadirá las dependencias necesarias y creará el proyecto. La compilación producirá un archivo .jar, que es el paquete de implementación.

1. Cree un nuevo proyecto de Maven en Eclipse.
 - a. En el menú File, elija New, seguido de Project.
 - b. En la ventana New Project, elija Maven Project.
 - c. En la ventana New Maven Project, elija Create a simple project y deje el resto de selecciones con sus valores predeterminados.
 - d. En las ventanas New Maven Project y Configure project, escriba la siguiente información para Artifact:
 - Group Id: doc-examples
 - Artifact Id: lambda-java-example
 - Version: 0.0.1-SNAPSHOT
 - Packaging: jar
 - Name: lambda-java-example
2. Añada la dependencia `aws-lambda-java-core` al archivo `pom.xml`.

Proporciona definiciones de las interfaces `RequestHandler`, `RequestStreamHandler` y `Context`. Esto le permite compilar código que puede utilizar con AWS Lambda.

- a. Abra el menú contextual (haga clic con el botón derecho) del archivo `pom.xml`, elija Maven y, a continuación, Add Dependency.
- b. En las ventanas Add Dependency, escriba los siguientes valores:

Group Id: com.amazonaws

Artifact Id: aws-lambda-java-core

Version: 1.1.0

Note

Si está siguiendo otros temas del tutorial de esta guía, es posible que los tutoriales específicos le soliciten añadir más dependencias. Asegúrese de añadir las dependencias que sean necesarias.

3. Añada la clase de Java al proyecto.

- a. Abra el menú contextual (haga clic con el botón derecho) del subdirectorio `src/main/java` del proyecto, elija New y, a continuación, Class.
- b. En la ventana New Java Class, escriba los siguientes valores:

- Package: **example**
- Name: **Hello**

Note

Si está siguiendo otros temas del tutorial de esta guía, es posible que los tutoriales específicos le recomiendan un nombre de paquete o de clase diferente.

- c. Añada el código Java. Si está siguiendo otros temas del tutorial de esta guía, añada el código proporcionado.
4. Compile el proyecto.

Abra el menú contextual (haga clic con el botón derecho) del proyecto en Package Explorer y elija Run As, seguido de Maven Build.... En la ventana Edit Configuration, escriba **package** en el cuadro Goals.

Note

El archivo .jar resultante, `lambda-java-example-0.0.1-SNAPSHOT.jar`, no es el archivo .jar independiente final que puede utilizar en el paquete de implementación. En el siguiente paso, añadirá el `maven-shade-plugin` de Apache para crear el .jar independiente. Para obtener más información, vaya a [Apache Maven Shade Plugin](#).

5. Añada el complemento `maven-shade-plugin` y vuelva a compilar.

El complemento `maven-shade-plugin` tomará los artefactos (archivos .jar) producidos por el objetivo `package` (produce archivos .jar de código de cliente), y creará un archivo .jar independiente que contiene el código de cliente compilado, así como las dependencias resueltas de `pom.xml`.

- a. Abra el menú contextual (haga clic con el botón derecho) del archivo `pom.xml`, elija Maven y, a continuación, Add Plugin.
- b. En la ventana Add Plugin, escriba los siguientes valores:
 - Group Id: `org.apache.maven.plugins`
 - Artifact Id: `maven-shade-plugin`
 - Version: 2.3
- c. Vuelva a compilar de nuevo.

Esta vez crearemos el archivo .jar como antes y, a continuación, utilizaremos el complemento `maven-shade-plugin` para extraer las dependencias y crear el .jar independiente

- i. Abra el menú contextual (haga clic con el botón derecho) del proyecto, elija Run As y, a continuación Maven build....
- ii. En la ventana Edit Configuration, escriba **package shade:shade** en el cuadro Goals.
- iii. Elija Run.

Puede encontrar el archivo .jar independiente resultante (es decir, el paquete de implementación), en el subdirectorio `/target` .

Abra el menú contextual (haga clic con el botón derecho) del subdirectorio `/target` y elija Show In, seguido de System Explorer; encontrará el archivo `lambda-java-example-0.0.1-SNAPSHOT.jar`.

Creación de un paquete de implementación .zip (Java)

En esta sección se proporcionan ejemplos de cómo crear un archivo .zip como paquete de implementación. Puede utilizar la herramienta de compresión que desee para crear este archivo zip. Independientemente de la herramienta que utilice, el archivo .zip debe tener la siguiente estructura:

- Todos los archivos de clase compilados y los archivos de recursos en el nivel raíz.
- Todos los archivos jar necesarios para ejecutar el código en el directorio /lib.

Note

También puede crear un archivo .jar independiente (así como un archivo comprimido) como paquete de implementación. Para ver ejemplos de cómo crear un archivo .jar independiente utilizando Maven, consulte [Creación de un paquete de implementación \(Java\) \(p. 70\)](#).

En los siguientes ejemplos se utiliza la herramienta de compilación e implementación Gradle para crear el archivo .zip.

Important

Se necesita la versión 2.0 o posterior de Gradle.

Antes de empezar

Tendrá que descargar Gradle. Para obtener instrucciones, vaya al sitio web de Gradle <https://gradle.org/>.

Ejemplo 1: Creación del archivo .zip utilizando Gradle y el repositorio central de Maven

Al final de este procedimiento, dispondrá un directorio de proyecto (*project-dir*) cuyo contenido tendrá la siguiente estructura:

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

La carpeta /java contendrá el código. Por ejemplo, si el nombre del paquete es example y contiene la clase Hello.java, la estructura será:

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

Después de compilar el proyecto, el archivo .zip obtenido (es decir, el paquete de implementación) estará en el subdirectorio *project-dir*/build/distributions.

1. Cree el directorio del proyecto (*project-dir*).
2. En *project-dir*, cree el archivo build.gradle y añada el contenido siguiente:

```
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

- La sección de repositorios hace referencia al repositorio central de Maven. En el momento de la compilación, obtiene las dependencias (es decir, las dos bibliotecas de AWS Lambda) del repositorio central de Maven.
- La tarea `buildZip` describe cómo crear el archivo .zip del paquete de implementación.

Por ejemplo, si descomprime el archivo .zip resultante, debería encontrar cualquiera de los archivos de clase compilados y los archivos de recursos en el nivel raíz. También debería encontrar un directorio /lib con los archivos jar necesarios para ejecutar el código.

- Si está siguiendo otros temas del tutorial de esta guía, es posible que los tutoriales específicos le soliciten añadir más dependencias. Asegúrese de añadir las dependencias que sean necesarias.

3. En `project-dir`, cree la siguiente estructura:

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

4. En el subdirectorio /java, añada los archivos de Java y la estructura de carpetas, si los hay. Por ejemplo, si el nombre del paquete de Java es `example` y el código fuente es `Hello.java`, la estructura de directorios tiene este aspecto:

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

5. Ejecute el siguiente comando gradle para compilar y comprimir el proyecto en un archivo .zip.

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

6. Verifique el archivo `project-dir.zip` resultante en el subdirectorio `project-dir/build/distributions`.
7. Ahora puede cargar el archivo .zip, el paquete de implementación, en AWS Lambda para crear una función de Lambda y probarla invocándola manualmente con los datos del evento de muestra. Para obtener instrucciones, consulte [Paso 2.3: Creación de una función de Lambda escrita en Java \(opcional\) \(p. 209\)](#).

Ejemplo 2: Creación del archivo .zip con Gradle utilizando archivos jar locales

Puede optar por no utilizar el repositorio central de Maven y tener todas las dependencias en la carpeta del proyecto. En este caso, la carpeta del proyecto (`project-dir`) tendrá la siguiente estructura:

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

Por tanto, si el código Java tiene el paquete `example` y la clase `Hello.java`, el código estará en el siguiente subdirectorio:

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

El archivo `build.gradle` debe contener lo siguiente:

```
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
```

Tenga en cuenta que las dependencias especifican `fileTree`, que identifica `project-dir/jars` como el subdirectorio que incluirá todos los archivos jar necesarios.

Ahora puede compilar el paquete. Ejecute el siguiente comando gradle para compilar y comprimir el proyecto en un archivo .zip.

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

Creación de funciones de Lambda mediante el IDE de Eclipse y el complemento del SDK de AWS (Java)

El conjunto de herramientas de Eclipse para el SDK de AWS ofrece un complemento de Eclipse que permite crear un paquete de implementación y cargarlo para crear una función de Lambda. Si utiliza el IDE de Eclipse como entorno de desarrollo, este complemento le permite crear código Java, crear y cargar un paquete de implementación y crear una función de Lambda. Para obtener más información, consulte [Guía de introducción a AWS Toolkit for Eclipse](#). Para ver un ejemplo de cómo utilizar el kit de herramientas para crear funciones de Lambda, consulte [Uso de Lambda con AWS Toolkit for Eclipse](#).

Creación de un paquete de implementación (Python)

Para crear una función de Lambda, primero debe crear un paquete de implementación de la función de Lambda, que es un archivo .zip que contiene el código y las dependencias.

Puede crear un paquete de implementación o escribir el código directamente en la consola de Lambda, en cuyo caso la consola crea el paquete de implementación automáticamente y lo carga, con lo que se crea la función de Lambda. Tenga en cuenta lo siguiente para determinar si puede utilizar la consola para crear una función de Lambda:

- Escenario sencillo: si el código personalizado solo necesita la biblioteca del SDK de AWS, puede utilizar el editor de código integrado de la consola de AWS Lambda. Con la consola, puede editar y cargar el código en AWS Lambda. La consola comprimirá el código con la información de configuración relevante en un paquete de implementación que el servicio de Lambda puede ejecutar.

También puede probar el código en la consola invocándolo manualmente y utilizando los datos del evento de muestra.

Note

El servicio de Lambda tiene preinstalado AWS SDK para Python.

- Escenario avanzado: si escribe código que utiliza otros recursos, como, por ejemplo, una biblioteca de gráficos para el procesamiento de imágenes, o desea utilizar la CLI de AWS en lugar de la consola,

primero debe crear el paquete de implementación de la función de Lambda y, a continuación, utilizar la consola o la CLI para cargar el paquete.

Note

Después de crear un paquete de implementación, puede cargarlo directamente o cargar primero el archivo .zip en un bucket de Amazon S3 de la misma región de AWS en la que desea crear la función de Lambda y, a continuación, especificar el nombre del bucket y el nombre de la clave de objeto cuando crear la función de Lambda mediante la consola o la CLI de AWS.

A continuación se muestra un ejemplo de procedimiento para crear un paquete de implementación (fuera de la consola).

Note

Esto debería funcionar para la mayoría de las instalaciones estándar de Python y pip al utilizar módulos puros de Python en una función de Lambda. Si incluye módulos que tienen dependencias nativas o ha instalado Python con Homebrew en OS X, consulte la siguiente sección que proporciona instrucciones para crear un paquete de implementación cuando se utiliza Virtualenv. Para obtener más información, consulte [Creación de un paquete de implementación utilizando un entorno de Python creado con Virtualenv \(p. 79\)](#) y el sitio web de [Virtualenv](#).

Utilizará pip para instalar las dependencias o las bibliotecas. Para obtener información acerca de la instalación de pip, vaya a [Installation](#).

1. Cree un directorio, por ejemplo, `project-dir`.
2. Guarde todos los archivos de código fuente de Python (los archivos .py) en el nivel raíz de este directorio.
3. Instale las bibliotecas con pip. También debe instalarlas en el nivel raíz del directorio.

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

Por ejemplo, el siguiente comando instala la biblioteca HTTP `requests` en el directorio `project-dir`.

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

Si utiliza Mac OS X y ha instalado Python con Homebrew (consulte [Homebrew](#)), el comando anterior no funcionará. Una solución sencilla consiste en añadir un archivo `setup.cfg` en la carpeta `/path/to/project-dir` con el contenido siguiente.

```
[install]
prefix=
```

4. Comprima el contenido del directorio `project-dir`, que es el paquete de implementación.

Important

Comprima el contenido del directorio, no el directorio. El contenido del archivo zip está disponible como el directorio de trabajo actual de la función de Lambda. Por ejemplo: `/project-dir/codefile.py/lib/yourlibraries`

Note

AWS Lambda incluye el AWS SDK para Python (Boto 3), por lo que no es necesario que lo incluya en el paquete de implementación. Sin embargo, si desea utilizar una versión de

Boto3 distinta de la que se incluye de forma predeterminada, puede incluirla en el paquete de implementación.

Creación de un paquete de implementación utilizando un entorno de Python creado con Virtualenv

En esta sección se explica cómo crear un paquete de implementación cuando se utiliza un entorno de Python que se ha creado con la herramienta Virtualenv. Considere el siguiente ejemplo:

- Se ha creado el siguiente entorno aislado de Python utilizando la herramienta Virtualenv y se ha activado el entorno:

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

Puede activar el entorno en Windows, OS X y Linux como se indica a continuación:

- En Windows, la activación se realiza mediante `activate.bat`:

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

- En OS X y Linux, mediante el script `activate`:

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

- Además, se supone que ha instalado el paquete `requests` en el entorno activado (dando por hecho que se va a utilizar en el código). Puede instalar estos paquetes como se indica a continuación:

```
pip install requests
```

Para crear un paquete de implementación, haga lo siguiente:

1. En primer lugar, cree un archivo `.zip` con el código de Python que desea cargar en AWS Lambda.
2. Añada las bibliotecas del entorno virtual que ha activado anteriormente al archivo `.zip`. Es decir, añada el contenido del directorio siguiente al archivo `.zip` (observe de nuevo que se añade el contenido del directorio y no el directorio en sí).

Para Windows, el directorio es:

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

Para OS X y Linux, el directorio es:

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

Note

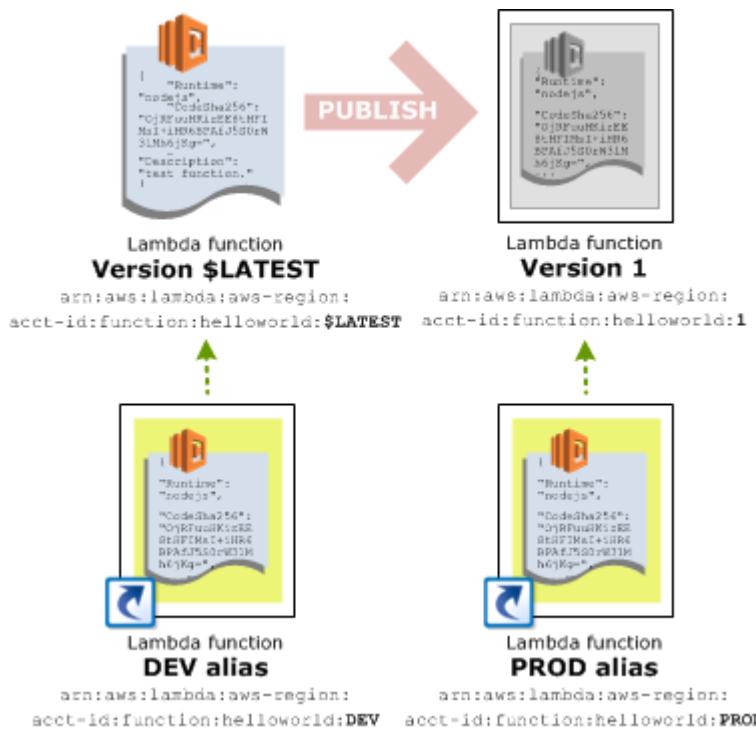
Si no encuentra los paquetes en el directorio `site-packages` del entorno virtual, puede encontrarlos en el directorio `dist-packages`.

Para obtener un ejemplo de creación de un paquete de implementación de Python, consulte [Python \(p. 225\)](#).

Control de versiones y alias de las funciones de AWS Lambda

Gracias al control de versiones, puede administrar mejor el código de las funciones de Lambda en producción, ya que le permite publicar una o varias versiones de una función de Lambda. Por ello, puede trabajar con diferentes variaciones de la función de Lambda en el flujo de trabajo de desarrollo; por ejemplo, versiones de desarrollo, beta y de producción. Cada versión de una función de Lambda tiene su propio nombre de recurso de Amazon (ARN). Después de publicar una versión, es immutable (es decir, que no se puede cambiar).

AWS Lambda permite crear alias para cada una de las versiones de una función de Lambda. Conceptualmente, un alias de AWS Lambda es un puntero a una versión específica de una función de Lambda, pero también es un recurso similar a una función de Lambda, y cada alias tiene su propio ARN. Cada alias mantiene un ARN para una versión de la función a la que apunta (tenga en cuenta que un alias solo puede apuntar a una versión de una función, no a otro alias). A diferencia de las versiones, que son inmutables, los alias son mutables (es decir, se pueden modificar) y se pueden actualizar para que apunten a diferentes versiones.



Los alias le permiten abstraer el proceso de promoción a producción de versiones nuevas de las funciones de Lambda a partir del mapeo de la versión de la función de Lambda y su origen de eventos. Para obtener más información, consulte [Funcionamiento \(p. 188\)](#).

Por ejemplo, supongamos que Amazon S3 es el origen de eventos que invoca una función de Lambda cuando se crean objetos en un bucket. Cuando Amazon S3 es el origen de eventos, la información de mapeo de origen de eventos se almacena en la configuración de notificaciones del bucket. En la configuración, puede identificar el ARN de la función de Lambda que Amazon S3 puede invocar, pero, en este caso, cada vez que publique una versión nueva de la función de Lambda, necesitará actualizar la configuración de notificaciones para que Amazon S3 invoque la versión correcta. En lugar de especificar el ARN de la función, puede especificar el ARN de un alias en la configuración de notificaciones (por ejemplo, el ARN del alias PROD). A medida que promocione a producción versiones nuevas de la función

de Lambda, solo necesitará actualizar el alias PROD para que apunte a la última versión estable, y no necesitará actualizar la configuración de notificaciones en Amazon S3.

Lo mismo se aplica cuando es necesario volver a una versión anterior de una función de Lambda. En este caso, basta con actualizar el alias PROD para que apunte a una versión distinta de la función, y no es necesario actualizar los mapeos de los orígenes de eventos.

Recomendamos que utilice el control de versiones y los alias para implementar funciones de Lambda cuando cree aplicaciones con múltiples dependencias y en las que trabajen varios desarrolladores.

Para obtener información detallada, consulte los siguientes temas:

Temas

- [Introducción al control de versiones de AWS Lambda \(p. 81\)](#)
- [Introducción a los alias de AWS Lambda \(p. 85\)](#)
- [Control de versiones, alias y políticas de recursos \(p. 94\)](#)
- [Administración del control de versiones mediante la Consola de administración de AWS, la AWS CLI o las API de Lambda \(p. 95\)](#)

Introducción al control de versiones de AWS Lambda

En esta sección se explica cómo crear una función de Lambda y publicar una versión de ella. También se explica cómo actualizar el código y la información de configuración de la función cuando se han publicado una o varias versiones. Además, esta sección incluye información sobre cómo eliminar versiones de la función, tanto versiones específicas como la totalidad de la función de Lambda (con todas sus versiones y alias asociados).

Creación de una función de Lambda (la versión \$LATEST)

Cuando se crea una función de Lambda, solo hay una versión. Se trata de la versión \$LATEST.



Puede hacer referencia a esta función mediante su nombre de recurso de Amazon (ARN). Existen dos ARN asociados a esta versión inicial:

- ARN completo: el ARN de la función con el sufijo de la versión.

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

- ARN incompleto: el ARN de la función sin el sufijo de la versión.

Puede utilizar este ARN en todas las operaciones pertinentes, pero no puede utilizarlo para crear un alias. Para obtener más información, consulte [Introducción a los alias de AWS Lambda \(p. 85\)](#).

El ARN incompleto tiene sus propias políticas de recursos.

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

Note

A menos que decida publicar versiones, la versión \$LATEST es la única versión de la función de Lambda de la que dispone. Puede utilizar el ARN completo o incompleto en el mapeo de origen de eventos para invocar esta \$LATEST versión.

A continuación se muestra un ejemplo de respuesta de una llamada a la API `CreateFunction`:

```
{  
    "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"  
}
```

Para obtener más información, consulte [CreateFunction \(p. 385\)](#).

En esta respuesta, AWS Lambda devuelve el ARN incompleto de la función que se acaba de crear, así como su versión, \$LATEST. La respuesta también muestra que Version es \$LATEST. El valor de CodeSha256 es la suma de comprobación del paquete de implementación que se ha cargado.

Publicación de una versión de una función de Lambda

Cuando se publica una versión, AWS Lambda hace una copia de snapshot del código de la función de Lambda (y de su configuración) en la versión \$LATEST. Una versión publicada es immutable. Es decir, no se puede cambiar el código ni la información de configuración. La versión nueva tiene un ARN único que incluye un sufijo con el número de versión, tal como se muestra a continuación:



Puede publicar una versión utilizando cualquiera de los siguientes métodos:

- Publicar una versión de forma explícita: utilice la API `PublishVersion` para publicar una versión de forma explícita. Para obtener más información, consulte [PublishVersion \(p. 445\)](#). Esta acción crea una versión nueva utilizando el código y la configuración de la versión `$LATEST`.
- Publicar una versión en el momento en que se crea o se actualiza una función de Lambda: utilice las solicitudes `CreateFunction` o `UpdateFunctionCode` para publicar también una versión añadiendo el parámetro opcional `publish` en la solicitud:
 - Especifique el parámetro `publish` en la solicitud `CreateFunction` para crear una función de Lambda (la versión `$LATEST`) y, a continuación, publicarla inmediatamente mediante la creación de una instantánea a la que se asigna la versión 1. Para obtener más información acerca de `CreateFunction`, consulte [CreateFunction \(p. 385\)](#).
 - Especifique el parámetro `publish` en la solicitud `UpdateFunctionCode` para actualizar el código en la versión `$LATEST` y, a continuación, publicar una versión a partir de la versión `$LATEST`. Para obtener más información acerca de `UpdateFunctionCode`, consulte [UpdateFunctionCode \(p. 466\)](#).

Si especifica el parámetro `publish` en el momento de crear una función de Lambda, la información de configuración de la función que AWS Lambda devuelve como respuesta muestra el número de la versión que se acaba de publicar, tal como se muestra a continuación (en el ejemplo, es la versión 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

Lambda solo publica una versión nueva si el código todavía no se ha publicado o si el código ha cambiado en comparación con la versión `$LATEST`. Si no existe ningún cambio, se devolverá la última versión `$LATEST` publicada.

Le recomendamos que publique una versión al mismo tiempo que crea una función de Lambda o que actualiza el código de la función de Lambda, especialmente cuando participan varios desarrolladores en el desarrollo de la misma función de Lambda. Para ello, puede utilizar el parámetro `publish` en la solicitud. Cuando hay varios desarrolladores trabajando en un proyecto, es posible que el desarrollador A cree una función de Lambda (la versión `$LATEST`) y que, antes de que el desarrollador A publique una versión, el desarrollador B actualice el código (paquete de implementación) asociado a la versión `$LATEST`. En este caso, se pierde el código original cargado por el desarrollador A. Cuando los dos desarrolladores añaden el parámetro `publish`, se impide la condición de carrera descrita.

Note

Las versiones publicadas son inmutables. Es decir, no se puede cambiar el código ni la información de configuración asociados a una versión.

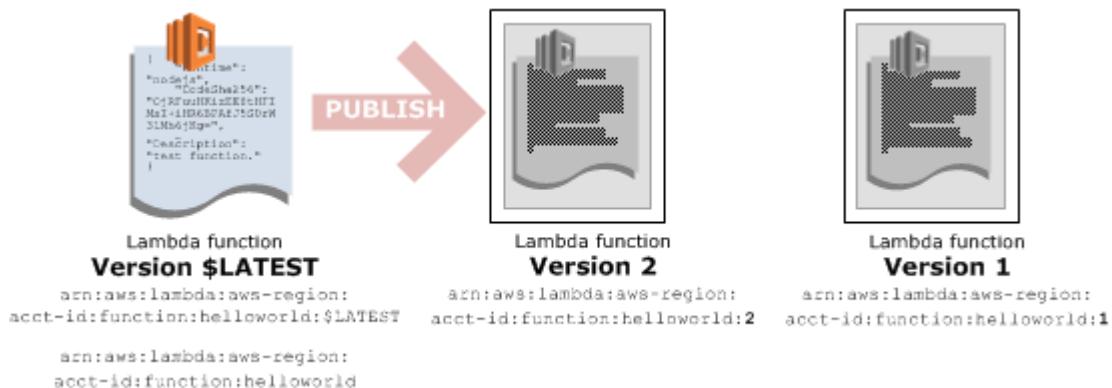
Cada versión de una función de Lambda es un recurso único con su propio nombre de recurso de Amazon (ARN). En el siguiente ejemplo se muestra el ARN de la versión número de 1 de la función de Lambda `helloworld`:

```
arn:aws:lambda:aws-region:acct-id:function:helloworld:1
```

Note

Se trata de un ARN completo, donde el número de la versión es un sufijo. Las versiones publicadas solo pueden tener un ARN completo.

Puede publicar varias versiones. Cada vez que se publica una versión, AWS Lambda copia la versión \$LATEST (el código y la información de configuración) para crear una versión nueva. Cuando se publican versiones adicionales, AWS Lambda asigna al control de versiones un número de secuencia que crece de forma monótona, incluso si la función se ha eliminado y se ha vuelto a crear. Los números de versión nunca se reutilizan, incluso para una función que se haya eliminado y vuelto a crear, de modo que el consumidor de dicha versión pueda confiar en que el ejecutable de esa versión nunca cambiará (excepto si se elimina). Si desea volver a utilizar un calificador, utilice alias con las versiones. Los alias sí se pueden eliminar y volver a crear con el mismo nombre.



Actualización del código y de la configuración de una función de Lambda

AWS Lambda mantiene el código más reciente de la función en la versión \$LATEST. Cuando se actualiza el código de la función, AWS Lambda sustituye el código de la versión \$LATEST de la función de Lambda. Para obtener más información, consulte [UpdateFunctionCode \(p. 466\)](#).

Las versiones publicadas son inmutables. No se puede actualizar el código ni la información de configuración asociada a una versión publicada.

Dispone de las siguientes opciones para publicar de una nueva versión cuando actualice el código de una función de Lambda:

- Publicar una versión en la misma solicitud de actualización de código: utilice la API [UpdateFunctionCode](#) (recomendado).
- Primero actualizar el código y, a continuación, publicar una versión de forma explícita: utilice la API [PublishVersion](#).

Puede actualizar el código y la información de configuración (por ejemplo, la descripción, el tamaño de la memoria y el tiempo de espera de ejecución) de la versión \$LATEST de una función de Lambda. Sin embargo, las versiones publicadas son inmutables. Es decir, no se puede cambiar el código ni la información de configuración.

Eliminación de una función de Lambda y de una versión específica

Con el control de versiones, dispone de las siguientes opciones:

- Eliminar una versión específica: puede eliminar una versión de una función de Lambda especificando la versión que desea eliminar en la solicitud [DeleteFunction](#). Si hay alias que dependen de esta versión, la solicitud fallará. AWS Lambda elimina la versión únicamente si no hay alias que dependan de esta

versión. Para obtener más información acerca de los alias, consulte [Introducción a los alias de AWS Lambda \(p. 85\)](#).

- Eliminar la función de Lambda por completo (todas sus versiones y alias): para eliminar la función de Lambda y todas sus versiones, no especifique ninguna versión en la solicitud `DeleteFunction`. Esto elimina la función por completo, incluyendo todas sus versiones y alias.

Important

Puede eliminar una versión específica de una función, pero no puede eliminar la versión `$LATEST`.

Temas relacionados

[Introducción a los alias de AWS Lambda \(p. 85\)](#)

[Administración del control de versiones mediante la Consola de administración de AWS, la AWS CLI o las API de Lambda \(p. 95\)](#)

Introducción a los alias de AWS Lambda

Puede crear alias para una función de Lambda. Un alias de AWS Lambda es como un puntero a una versión específica de una función de Lambda. Para obtener más información sobre el control de versiones, consulte [Introducción al control de versiones de AWS Lambda \(p. 81\)](#). Un alias permite tener acceso a la función de Lambda a la que apunta (por ejemplo, para invocar la función) sin que el intermediario tenga que conocer la versión específica a la que apunta el alias.

Los alias de AWS Lambda permiten abordar los siguientes casos de uso:

- Mayor facilidad para la promoción de versiones nuevas de las funciones de Lambda y su reversión cuando sea necesario: después de crear inicialmente una función de Lambda (la versión `$LATEST`) puede publicar primero la versión 1 de la función. Si crea un alias denominado PROD que apunta a la versión 1, puede utilizarlo para invocar la versión 1 de la función de Lambda.

A continuación, puede actualizar el código (la versión `$LATEST`) con todas sus mejoras y, después, publicar otra versión estable y mejorada (versión 2). Puede promocionar a producción la versión 2 reasignando el alias PROD para que apunte a la versión 2. Si tiene algún problema, puede revertir fácilmente la versión de producción a la versión 1 reasignando el alias PROD de forma que apunte a la versión 1.

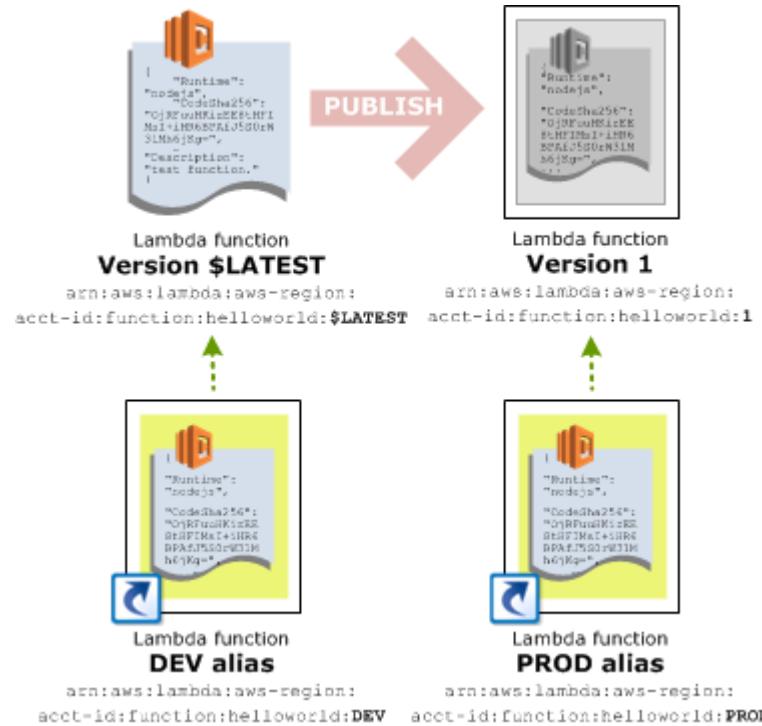
Note

En este contexto, los términos promoción y reversión hacen referencia a la reasignación de los alias a versiones distintas de la función.

- Simplificación de la administración de mapeos de orígenes de eventos: en lugar de utilizar los ARN de las funciones de Lambda en los mapeos de orígenes de eventos, mediante la utilización del ARN de un alias puede asegurarse de que no va a tener que actualizar los mapeos de orígenes de eventos al promocionar a una versión nueva o revertir a una versión anterior.

Un alias de AWS Lambda es un recurso similar a una función de Lambda. Sin embargo, no puede crear un alias por sí solo. Un alias se crea para una función de Lambda existente. Si una función de Lambda es un recurso, puede pensar en un alias de AWS Lambda como un subrecurso asociado a una función de Lambda.

Tanto la función de Lambda como el alias son recursos de AWS Lambda y, al igual que todos los demás recursos de AWS, ambos tienen nombres de recurso de Amazon (ARN) únicos. En el siguiente ejemplo se muestra una función de Lambda (la versión `$LATEST`), con una versión publicada. Cada versión tiene un alias que apunta a ella.



Puede acceder a la función utilizando el ARN de la función o el ARN del alias.

- Dado que la versión de la función es \$LATEST, puede acceder a ella utilizando el ARN completo o incompleto.
- ARN completo de una función (con el sufijo de la versión \$LATEST):

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

- Cuando se utiliza cualquiera de los ARN de los alias, se utiliza un ARN completo. Cada ARN de alias tiene un sufijo de nombre de alias.

```
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 ofrece las siguientes API para que pueda crear y gestiona los alias:

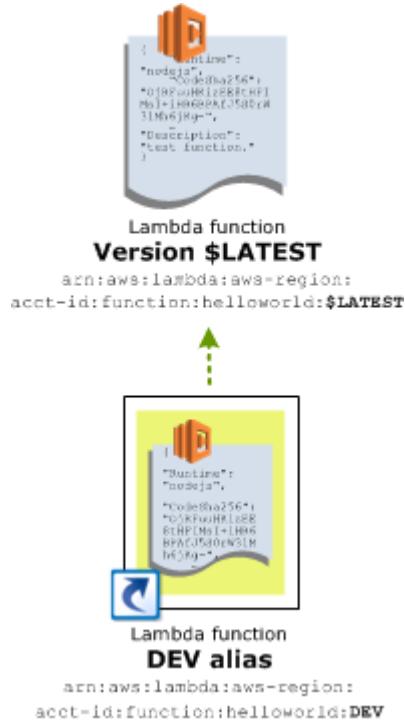
- [CreateAlias \(p. 376\)](#)
- [UpdateAlias \(p. 458\)](#)
- [GetAlias \(p. 405\)](#)
- [ListAliases \(p. 431\)](#)
- [DeleteAlias \(p. 393\)](#)

Ejemplo: uso de alias para administrar versiones de funciones de Lambda

A continuación se muestra un escenario de ejemplo de cómo utilizar el control de versiones y los alias para promocionar a producción versiones nuevas de funciones de Lambda.

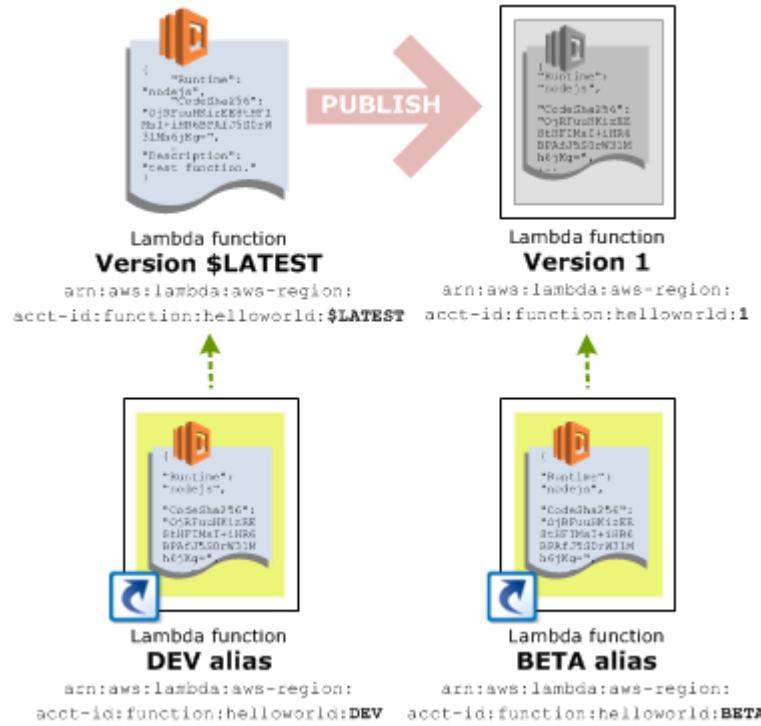
En primer lugar, cree una función de Lambda

Se trata de la versión **\$LATEST**. También debe crear un alias (DEV, para desarrollo) que apunte a la función nueva. Los desarrolladores pueden utilizar este alias para probar la función con orígenes de eventos en un entorno de desarrollo.



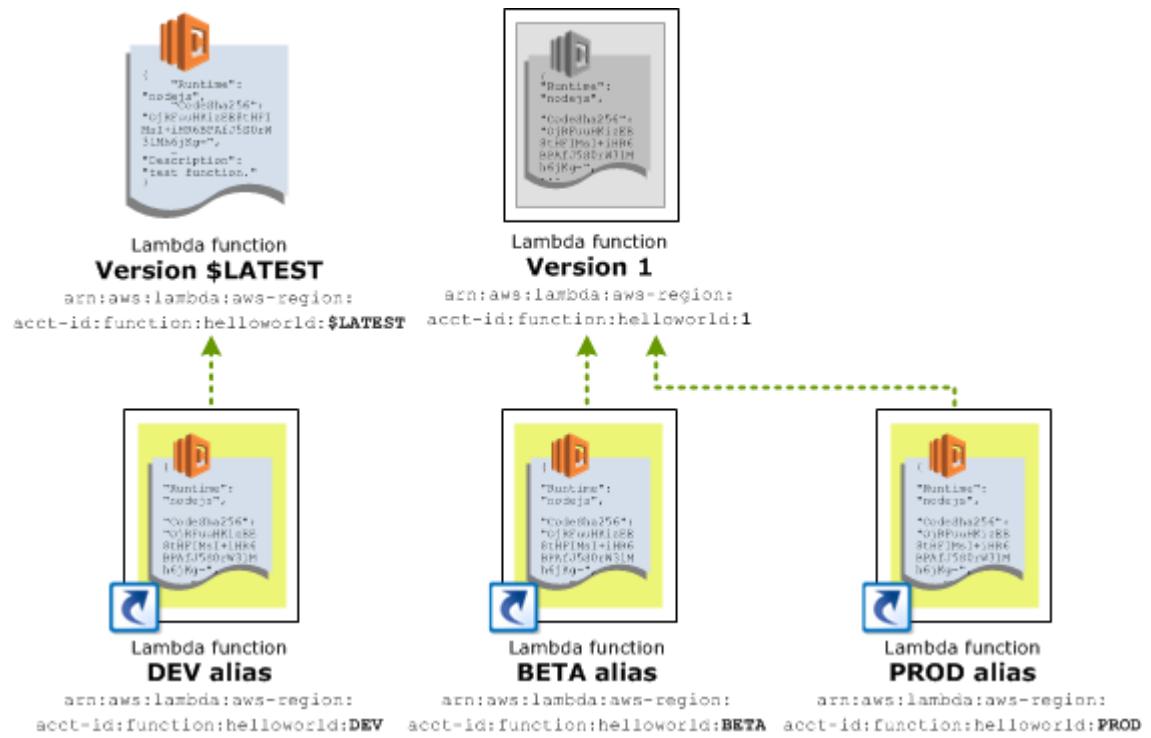
Pruebe la versión de la función con orígenes de eventos en un entorno beta, en una forma estable, mientras continúa desarrollando versiones nuevas.

Publique una versión a partir de la versión **\$LATEST** y utilice otro alias (BETA) que apunte a ella. Esto le permite asociar orígenes de eventos beta a este alias específico. En los mapeos de orígenes de eventos, utilice el alias BETA para asociar la función de Lambda al origen de eventos.



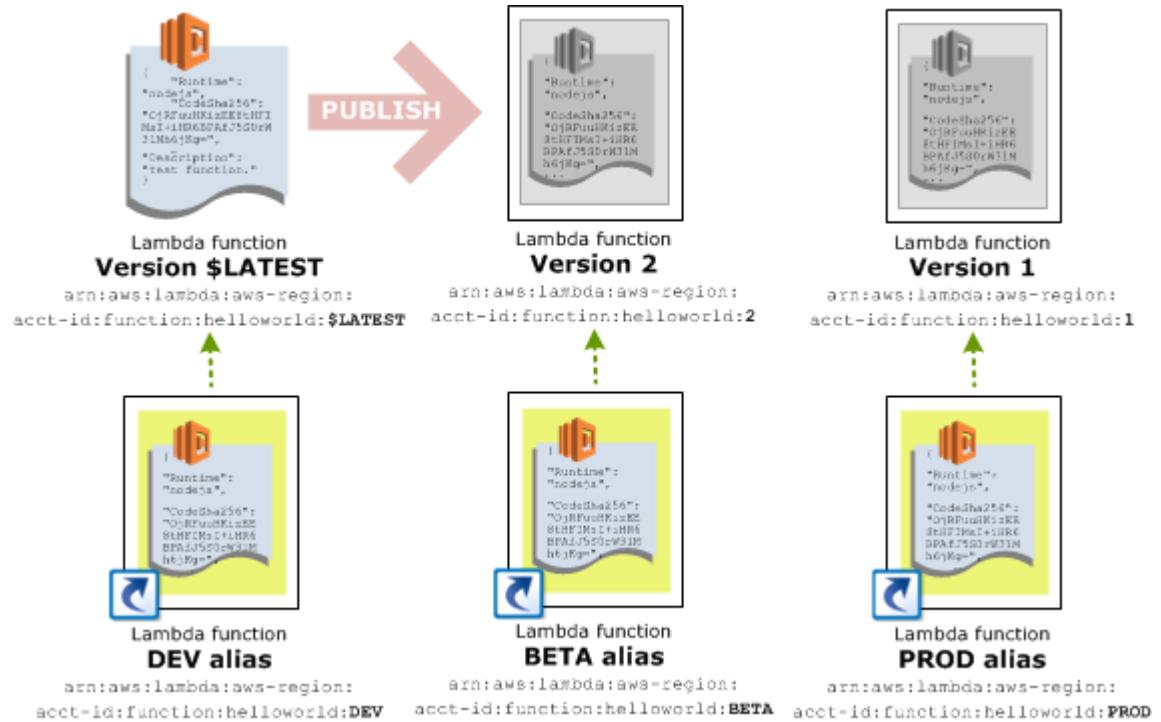
Promocione a producción la versión de la función de Lambda para que funcione con orígenes de eventos del entorno de producción

Tras probar la versión BETA, puede definir la versión de producción mediante la creación de un alias que mapee a la versión 1. Esto significa que desea que los orígenes de eventos de producción apunten a esta versión específica. Para ello, cree el alias PROD y utilice el ARN del alias PROD en todos los mapeos de orígenes de eventos de producción.



Continúe el desarrollo, publique más versiones y pruébelas

A medida que desarrolle el código, puede actualizar la versión `$LATEST` cargando el código actualizado y, a continuación, publicándolo para las pruebas beta con solo cambiar el alias `BETA` para que apunte a él. Esta sencilla reasignación del alias `BETA` permite poner en beta la versión 2 de la función de Lambda sin cambiar ninguno de los orígenes de eventos. Esta es la forma en que los alias permiten controlar qué versiones de una función se utilizan con orígenes de eventos específicos en un entorno de desarrollo.



Si desea probar a crear esta configuración utilizando la CLI de AWS, consulte [Tutorial: Uso de los alias de AWS Lambda \(p. 90\)](#).

Temas relacionados

[Introducción al control de versiones de AWS Lambda \(p. 81\)](#)

[Tutorial: Uso de los alias de AWS Lambda \(p. 90\)](#)

[Administración del control de versiones mediante la Consola de administración de AWS, la AWS CLI o las API de Lambda \(p. 95\)](#)

Tutorial: Uso de los alias de AWS Lambda

Este tutorial basado en la AWS CLI crea versiones de funciones de Lambda y alias que apuntan a ellas como se describe en el [Ejemplo: uso de alias para administrar versiones de funciones de Lambda \(p. 86\)](#).

En este ejemplo se utiliza la región us-west-2 (EE.UU. Oeste, Oregón) para crear la función de Lambda y el alias.

1. En primer lugar, debe crear un paquete de implementación que puede cargar para crear la función de Lambda.
 - a. Abrir un editor de texto y, a continuación, copie el siguiente código.

```
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. Guarde el archivo como `helloworld.js`.
- c. Comprima el archivo `helloworld.js` como `helloworld.zip`.
2. Cree un rol de IAM (rol de ejecución) para especificarlo al crear la función de Lambda.
 - a. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.
 - b. Siga los pasos de [Roles de IAM](#) en la Guía del usuario de IAM para crear un rol de IAM (rol de ejecución). Cuando siga los pasos para crear un rol, tenga en cuenta lo siguiente:
 - En Select Role Type, elija AWS Service Roles, seguido de AWS Lambda.
 - En Attach Policy, elija la política denominada `AWSLambdaBasicExecutionRole`.
 - c. Anote el nombre de recurso de Amazon (ARN) del rol de IAM. Necesitará este valor en el siguiente paso al crear la función de Lambda.
3. Cree una función de Lambda (`helloworld`).

```
aws lambda create-function \
--region us-west-2 \
--function-name helloworld \
--zip-file file:///file-path/helloworld.zip \
--role arn:aws:iam::account-id:role/lambda_basic_execution \
--handler helloworld.handler \
--runtime nodejs6.10 \
--profile adminuser
```

La respuesta devuelve la información de configuración que incluye `$LATEST`, como la versión de la función, tal como se muestra en la siguiente respuesta de ejemplo:

```
{
  "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": ""
}
```

4. Cree un alias (DEV) que haga referencia a la versión `$LATEST` de la función de Lambda `helloworld`:

```
aws lambda create-alias \
--region us-west-2 \
--function-name helloworld \
--description "sample alias" \
--function-version "\$LATEST" \
--name DEV \
--profile adminuser
```

La respuesta devuelve la información del alias, incluida la versión de la función a la que apunta y el ARN del alias. El ARN es el mismo que el ARN de la función, con el nombre del alias como sufijo. A continuación se muestra un ejemplo de respuesta:

```
{  
    "AliasArn": "arn:aws:lambda:us-west-2:account-id:function:helloworld:DEV",  
    "FunctionVersion": "$LATEST",  
    "Name": "DEV",  
    "Description": "sample alias"  
}
```

5. Publique una versión de la función de Lambda helloworld.

```
aws lambda publish-version \  
--region us-west-2 \  
--function-name helloworld \  
--profile adminuser
```

La respuesta devuelve la información de configuración de la versión de la función, e incluye el número de la versión y el ARN de la función con el sufijo de la versión. A continuación se muestra un ejemplo de respuesta:

```
{  
    "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": ""  
}
```

6. Cree un alias (**BETA**) para la versión 1 de la función de Lambda helloworld.

```
aws lambda create-alias \  
--region us-west-2 \  
--function-name helloworld \  
--description "sample alias" \  
--function-version 1 \  
--name BETA \  
--profile adminuser
```

Ahora tiene dos alias para la función helloworld. El alias **DEV** apunta a la versión **\$LATEST** de la función y el alias **BETA** apunta a la versión 1 de la función de Lambda.

7. Ahora supongamos que desea poner en producción la versión 1 de la función helloworld. Cree otro alias (**PROD**) que apunte a la versión 1.

```
aws lambda create-alias \  
--region us-west-2 \  
--function-name helloworld \  
--description "sample alias" \  
--function-version 1 \  
--name PROD \  
--profile adminuser
```

En este momento, tiene los alias **BETA** y **PROD** apuntando a la versión 1 de la función de Lambda.

8. Ahora puede publicar una versión nueva (por ejemplo, la versión 2), pero primero necesita actualizar el código y cargar un paquete de implementación modificado. Si no se modifica la versión \$LATEST, no se puede publicar más de una versión de ella. Suponiendo que ha actualizado el paquete de implementación, que lo ha cargado y que ha publicado la versión 2, ahora puede modificar el alias BETA para que apunte a la versión 2 de la función de Lambda.

```
aws lambda update-alias \  
--region us-west-2 \  
--function-name helloworld \  
--function-version 2 \  
--name BETA \  
--profile adminuser
```

Ahora tiene tres alias que apuntan a versiones distintas de la función de Lambda: el alias DEV apunta a la versión \$LATEST, el alias BETA apunta a la versión 2 y el alias PROD apunta a la versión 1 de la función de Lambda.

Para obtener más información acerca de cómo utilizar la consola de AWS Lambda para administrar el control de versiones, consulte [Administración del control de versiones mediante la Consola de administración de AWS, la AWS CLI o las API de Lambda \(p. 95\)](#).

Concesión de permisos en un modelo de inserción

En un modelo de inserción (consulte [Mapeo de orígenes de eventos \(p. 134\)](#)), los orígenes de eventos, como Amazon S3 invocan la función de Lambda. Estos orígenes de eventos mantienen un mapeo que identifica la versión o el alias de una función que invocarán cuando se produzcan eventos. Tenga en cuenta lo siguiente:

- Recomendamos que especifique el alias de una función de Lambda en la configuración de mapeo (consulte [Introducción a los alias de AWS Lambda \(p. 85\)](#)). Por ejemplo, si el origen de eventos es Amazon S3, especifique el ARN del alias en la configuración de notificaciones del bucket para que Amazon S3 pueda invocar el alias cuando detecte eventos específicos.
- En el modelo de inserción, debe conceder permisos a los orígenes de eventos mediante una política de recursos asociada a la función de Lambda. En el control de versiones, los permisos que añada son específicos para el calificador que especifique en la solicitud AddPermission (consulte [Control de versiones, alias y políticas de recursos \(p. 94\)](#)).

Por ejemplo, el siguiente comando de la AWS CLI concede permisos a Amazon S3 para invocar el alias PROD de la función de Lambda helloworld (tenga en cuenta que el parámetro --qualifier especifica el nombre del alias).

```
aws lambda add-permission \  
--region us-west-2 \  
--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
```

En este caso, Amazon S3 ahora puede invocar el alias PROD, y AWS Lambda puede ejecutar la versión de la función de Lambda helloworld a la que apunta el alias PROD. Para que esto funcione, debe utilizar el ARN del alias PROD en la configuración de notificaciones del bucket de S3.

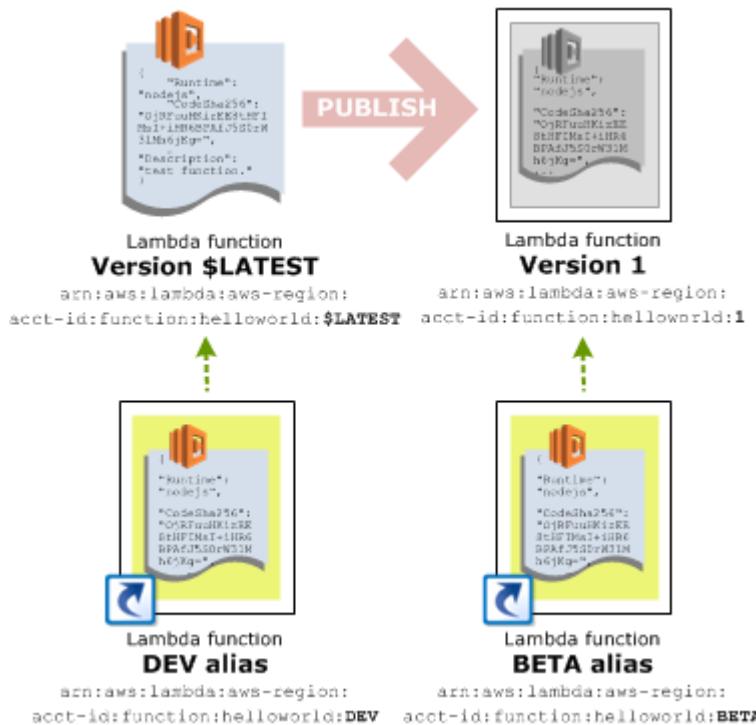
Para obtener información acerca de cómo controlar los eventos de Amazon S3, consulte [Tutorial: Uso de AWS Lambda con Amazon S3 \(p. 215\)](#).

Note

Si utiliza la consola de AWS Lambda para añadir un origen de eventos a una función de Lambda, la consola añade automáticamente los permisos necesarios.

Control de versiones, alias y políticas de recursos

El control de versiones y los alias permiten acceder a una función de Lambda utilizando distintos ARN. Por ejemplo, fíjese en el siguiente escenario:



Por ejemplo, puede invocar la versión 1 de la función `helloworld` utilizando cualquiera de los dos ARN siguientes:

- Mediante el ARN completo de la función:

```
arn:aws:lambda:aws-region:acct-id:function:helloworld:1
```

Note

Un ARN incompleto de una función (el ARN de una función sin un sufijo de versión o de alias) se mapea a la versión `$LATEST`.

- Mediante el ARN del alias `BETA`:

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

En un modelo de inserción, los orígenes de eventos (como Amazon S3 y las aplicaciones personalizadas) pueden invocar cualquiera de las versiones de la función de Lambda, siempre que se concedan los permisos necesarios a estos orígenes de eventos mediante el uso de una política de acceso asociada a

la función de Lambda. Para obtener más información acerca del modelo de inserción, consulte [Mapeo de orígenes de eventos \(p. 134\)](#).

Suponiendo que se conceda el permiso, surge la pregunta siguiente: "¿puede un origen de eventos invocar una versión de una función utilizando cualquiera de los ARN asociados?" La respuesta depende de cómo se haya identificado la función en la solicitud de adición de permisos (consulte [AddPermission \(p. 372\)](#)). La clave para comprender esto radica en que el permiso que se concede solamente se aplica al ARN utilizado en la solicitud adición de permisos:

- Si utiliza el nombre completo de la función (como `helloworld:1`), el permiso es válido para invocar la versión 1 de la función `helloworld` únicamente si se utiliza su ARN completo (si se utiliza cualquier otro ARN, se producirá un error de permisos).
- Si utiliza un nombre de alias (como `helloworld:BETA`), el permiso es válido únicamente para invocar la función `helloworld` utilizando el ARN del alias BETA (si se utiliza cualquier otro ARN, se producirá un error de permisos, incluido el ARN de la versión de la función a la que apunta el alias).
- Si utiliza el nombre incompleto de la función (como `helloworld`), el permiso es válido únicamente para invocar la función `helloworld` utilizando su ARN de función incompleto (si se utiliza cualquier otro ARN, se producirá un error de permisos).

Note

Tenga en cuenta que, aunque la política de acceso solo está en el ARN incompleto, el código y la configuración de la función de Lambda invocada siguen siendo los de la versión `$LATEST` de la función. El ARN incompleto de la función se mapea a la versión `$LATEST`, pero los permisos que se añaden son específicos del ARN.

- Si utiliza el nombre completo de la función que incluye la versión `$LATEST` (`helloworld:$LATEST`), el permiso es válido para invocar la versión `$LATEST` de la función `helloworld` únicamente si se utiliza su ARN completo (si se utiliza el ARN incompleto, se producirá un error de permisos).

Administración del control de versiones mediante la Consola de administración de AWS, la AWS CLI o las API de Lambda

Puede administrar el control de versiones de las funciones de Lambda mediante programación con los SDK de AWS (o hacer las llamadas a las API de AWS Lambda directamente, si es necesario), utilizando la AWS Command Line Interface (AWS CLI) o la consola de AWS Lambda.

AWS Lambda ofrece las siguientes API para la administración del control de versiones y los alias:

[PublishVersion \(p. 445\)](#)

[ListVersionsByFunction \(p. 442\)](#)

[CreateAlias \(p. 376\)](#)

[UpdateAlias \(p. 458\)](#)

[DeleteAlias \(p. 393\)](#)

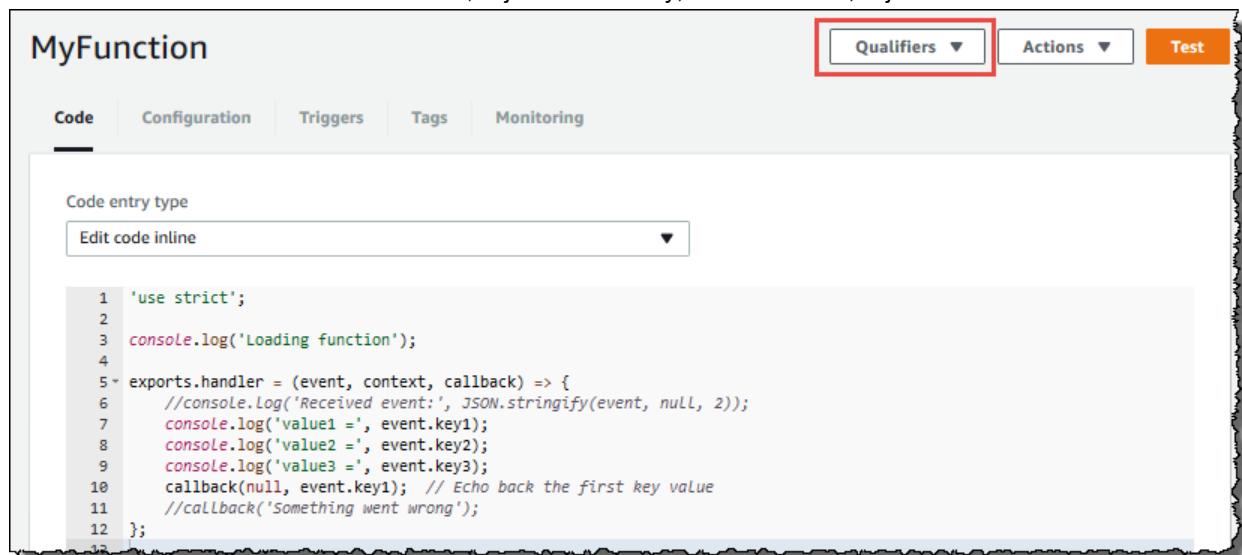
[GetAlias \(p. 405\)](#)

[ListAliases \(p. 431\)](#)

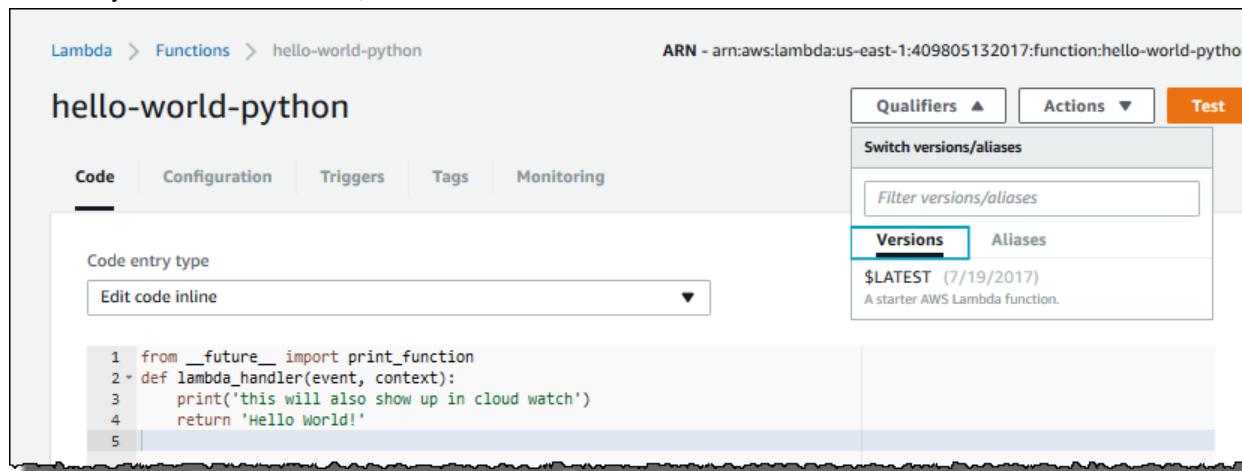
Además de estas API, las API apropiadas existentes también admiten operaciones relacionadas con el control de versiones.

Para ver un ejemplo de cómo puede utilizar la AWS CLI, consulte [Tutorial: Uso de los alias de AWS Lambda \(p. 90\)](#).

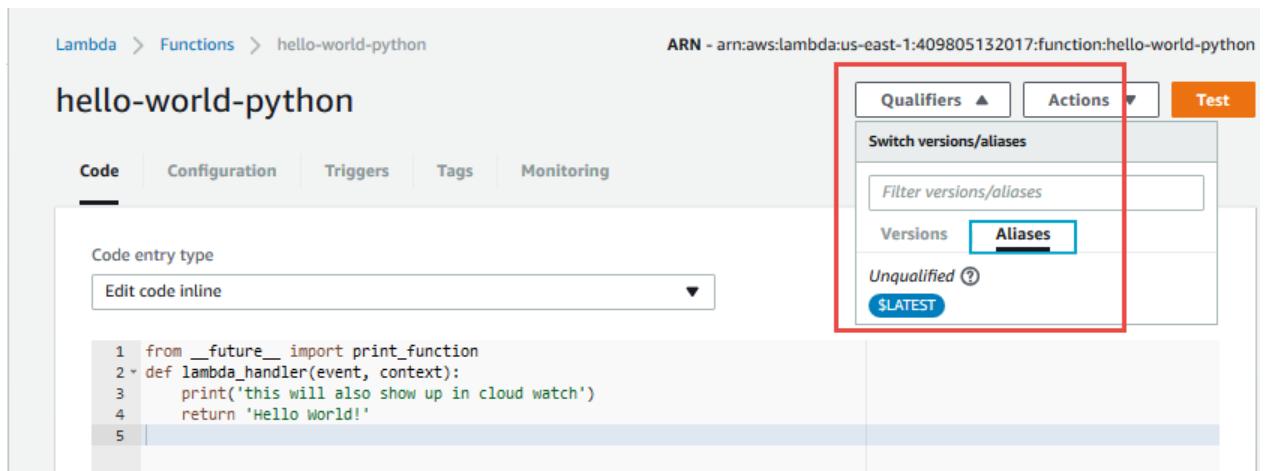
En esta sección se explica cómo puede utilizar la consola de AWS Lambda para administrar el control de versiones. En la consola de AWS Lambda, elija una función y, a continuación, elija Qualifiers.



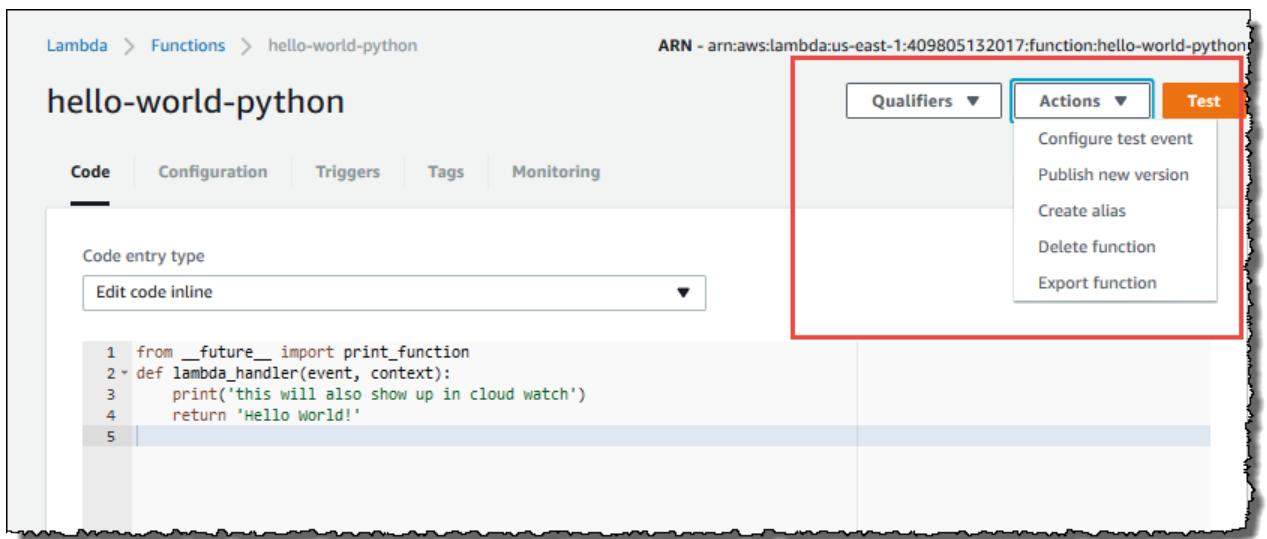
Cuando se expande el menú Qualifiers, muestra las pestañas Versions y Aliases, tal como se muestra en la siguiente captura de pantalla. En el panel Versions, puede ver una lista de las versiones de la función seleccionada. Si todavía no ha publicado ninguna versión de la función seleccionada, el panel Versions solo incluye la versión \$LATEST, tal como se muestra a continuación:



Elija pestaña Aliases para ver una lista de los alias de la función. Inicialmente, no tendrá ningún alias, tal como se muestra a continuación:



Ahora puede utilizar el menú Actions para publicar una versión o crear los alias que deseé para la función de Lambda seleccionada.



Para obtener más información sobre el control de versiones y los alias, consulte [Control de versiones y alias de las funciones de AWS Lambda \(p. 80\)](#).

Variables de entorno

Las variables de entorno para las funciones de Lambda le permiten pasar ajustes de forma dinámica al código de la función y a las bibliotecas, sin necesidad de realizar cambios en el código. Las variables de entorno son pares clave-valor que se crean y modifican como parte de la configuración de la función utilizando la consola de AWS Lambda, la CLI de AWS Lambda o el SDK de AWS Lambda. A continuación, AWS Lambda pone estos pares clave-valor a disposición de la función de Lambda mediante API estándar compatibles con el lenguaje, como `process.env` para las funciones escritas en Node.js.

Puede utilizar las variables de entorno para indicar a las bibliotecas el directorio en el que deben instalar los archivos, dónde almacenar las salidas, la configuración de conexión y de registro, etc. Al separar estos ajustes de la lógica de la aplicación, no es necesario actualizar el código de la función cuando se requiere cambiar su comportamiento en función de distintas configuraciones.

Configuración

Supongamos que desea que una función de Lambda se comporte de forma distinta a medida que se desplaza a través de las fases del ciclo de vida, desde el desarrollo a la implementación. Por ejemplo, las fases de desarrollo, pruebas y producción pueden contener bases de datos con las que la función debe conectarse y que requieren información de conexión diferente y el uso de nombres de tabla distintos. Puede crear variables de entorno para hacer referencia a los nombres de las bases de datos, la información de conexión o los nombres de las tablas, y establecer el valor para la función basándose en la fase en la que esta se está ejecutando (por ejemplo, desarrollo, pruebas o producción) sin necesidad de cambiar el código.

Las siguientes capturas de pantalla muestran cómo modificar la configuración de la función mediante la consola de AWS. La primera captura de pantalla configura los ajustes de la función correspondientes a una fase de prueba. La segunda configura los ajustes para una fase de producción.

The screenshot shows the 'Environment variables' section of the AWS Lambda function configuration. It displays two environment variables: 'Database' (value 'Test_DB') and 'DB_Connection' (value 'TEST'). Both values are highlighted with a red rectangle. Below this, there are 'Key' and 'Value' fields with 'Remove' buttons. The 'Encryption configuration' section is also visible, showing options for enabling helpers for encryption in transit and selecting a KMS key for encrypting environment variables at rest. A dropdown menu currently set to '(default) aws/lambda' has a red rectangle around it, and an 'Enter value' button is next to it.

▼ 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

Observe la sección Encryption configuration. Podrá obtener más información acerca de su uso en el tutorial [Creación de una función de Lambda utilizando variables de entorno para almacenar información confidencial](#) (p. 103).

También puede utilizar la CLI de AWS para crear funciones de Lambda que contengan variables de entorno. Para obtener más información, consulte las API [CreateFunction](#) (p. 385) y [UpdateFunctionConfiguration](#) (p. 472). También es posible utilizar variables de entorno al crear y actualizar funciones mediante AWS CloudFormation. Asimismo, las variables de entorno pueden emplearse para configurar ajustes específicos del tiempo de ejecución del lenguaje o de una biblioteca incluida en la función. Por ejemplo, puede modificar PATH para especificar el directorio en el que se almacenan ejecutables. También es posible establecer variables de entorno específicas del tiempo de ejecución, como, por ejemplo, PYTHONPATH para Python o NODE_PATH para Node.js.

En el siguiente ejemplo se crea una función de Lambda que establece la variable de entorno LD_LIBRARY_PATH, que se utiliza para especificar un directorio en el que se cargan dinámicamente las bibliotecas compartidas en tiempo de ejecución. En este ejemplo, el código de la función de Lambda utiliza la biblioteca compartida del directorio /usr/bin/test/lib64. Observe que el parámetro Runtime utiliza nodejs6.10, pero también puede especificar nodejs4.3.

```
aws lambda create-function \  
--region us-east-1  
--function-name myTestFunction  
--zip-file file://path/package.zip  
--role role-arn  
--environment Variables="{LD_LIBRARY_PATH=/usr/bin/test/lib64}"  
--handler index.handler  
--runtime nodejs6.10  
--profile default
```

Reglas para la nomenclatura de variables de entorno

No existe ningún límite respecto al número de variables de entorno que se pueden crear, siempre que el tamaño total del conjunto no supere los 4 KB.

Otros requisitos son:

- Deben comenzar por letras [a-zA-Z].
- Solo pueden contener caracteres alfanuméricos y guiones bajos [a-zA-Z0-9_-].

Además, existe un conjunto específico de claves reservadas para AWS Lambda. Si intenta establecer los valores de cualquiera de estas claves reservadas, recibirá un mensaje de error en el que se indica que la acción no está permitida. Para obtener más información acerca de estas claves, consulte [Variables de entorno disponibles para las funciones de Lambda \(p. 196\)](#).

Variables de entorno y control de versiones de las funciones

El control de versiones de las funciones ofrece una forma de administrar el código de la función de Lambda al permitirle publicar una o varias versiones de dicha función a medida que esta avanza por las fases de desarrollo, pruebas y producción. Para cada versión de una función de Lambda que se publica, las variables de entorno (además de otras configuraciones específicas de la función, como los límites `MemorySize` y `Timeout`) se guardan en forma de instantánea de la versión y esos ajustes son inmutables (no pueden modificarse).

A medida que los requisitos de la aplicación y de la configuración evolucionan, puede crear nuevas versiones de la función de Lambda y actualizar las variables de entorno para que cumplan esos requisitos antes de publicar la versión más reciente. La versión actual de la función es `$LATEST`.

Además, puede crear alias, que son punteros a una determinada versión de la función. La ventaja de los alias es que, en el caso de que sea necesario volver a una versión anterior de la función, le permiten dirigir el alias a dicha versión, que contiene las variables de entorno necesarias. Para obtener más información, consulte [Control de versiones y alias de las funciones de AWS Lambda \(p. 80\)](#).

Cifrado de variables de entorno

Al crear o actualizar funciones de Lambda que utilizan variables de entorno, AWS Lambda las cifra utilizando [AWS Key Management Service](#). Al invocar la función de Lambda, dichos valores se descifran y se ponen a disposición del código de Lambda.

La primera vez que se crean o actualizan funciones de Lambda que utilizan variables de entorno en una región, se crea automáticamente una clave de servicio predeterminada en AWS KMS. Esta clave se utiliza para cifrar variables de entorno. No obstante, si desea utilizar auxiliares de cifrado junto con KMS para cifrar las variables de entorno una vez creada la función de Lambda debe crear su propia clave de AWS KMS y elegirla en lugar de la clave predeterminada. No elija la clave predeterminada, ya que dará errores. Crear su propia clave le ofrece una mayor flexibilidad, incluida la opción de crear, rotar, desactivar y definir los controles de acceso y auditar las claves de cifrado utilizadas para proteger los datos. Para obtener más información, consulte la [Guía para desarrolladores de AWS Key Management Service](#).

Si utiliza su propia clave, se le facturará de acuerdo con las directrices de [Precios de AWS Key Management Service](#). No se le cobrará si utiliza la clave de servicio predeterminada proporcionada por AWS Lambda.

Si utiliza la clave de servicio de KMS predeterminada para Lambda no son necesarios permisos de IAM adicionales en el rol de ejecución de la función, y este funcionará automáticamente sin necesidad de realizar cambios. Si proporciona su propia clave de KMS (personalizada), deberá añadir `kms:Decrypt` al rol de ejecución. Además, el usuario que va a crear y actualizar la función de Lambda debe tener permisos para utilizar la clave de KMS. Para obtener más información acerca de las claves de KMS, consulte [Uso de políticas de claves en AWS KMS](#).

Almacenamiento de información confidencial

Tal como se ha mencionado en la sección anterior, al implementar la función de Lambda, todas las variables de entorno que se han especificado se cifran de forma predeterminada después del proceso de implementación, pero no durante este. A continuación, AWS Lambda las descifra automáticamente al invocar la función. Si necesita almacenar información confidencial en una variable de entorno, recomendamos encarecidamente que cifre dicha información antes de implementar la función de Lambda.

Afortunadamente, la consola de Lambda le facilita esta operación al proporcionarle auxiliares de cifrado que utilizan [AWS Key Management Service](#) para almacenar dicha información confidencial como `Ciphertext`. Asimismo, la consola de Lambda ofrece código auxiliar de descifrado para descifrar esa información que se puede utilizar en el código de la función de Lambda. Para obtener más información, consulte [Creación de una función de Lambda utilizando variables de entorno para almacenar información confidencial](#) (p. 103).

Escenarios de error

Si la configuración de la función supera los 4 KB, o utiliza claves de variable de entorno reservadas por AWS Lambda, la operación de creación o actualización presentará un error de configuración. Durante el tiempo de ejecución, es posible que el cifrado o el descifrado de las variables de entorno presenten un error. Si AWS Lambda no puede descifrar las variables de entorno debido a una excepción de servicio de AWS KMS, AWS KMS devolverá un mensaje de excepción explicando las condiciones del error y las medidas que se pueden aplicar para solucionar el problema. Estas se registrarán en el flujo de logs de la función en Amazon CloudWatch Logs. Por ejemplo, si la clave de KMS que utiliza para obtener acceso a las variables de entorno está deshabilitada, verá el siguiente mensaje de error:

```
Lambda was unable to configure access to your environment variables because the KMS key used is disabled.  
Please check your KMS key settings.
```

Paso siguiente

[Creación de una función de Lambda utilizando variables de entorno](#) (p. 101)

Creación de una función de Lambda utilizando variables de entorno

En esta sección se explica cómo modificar el comportamiento de una función de Lambda mediante cambios en la configuración que no requieren la modificación del código de la función.

En este tutorial, hará lo siguiente:

- Crear un paquete de implementación con código de muestra que devuelve el valor de una variable de entorno que especifica el nombre de un bucket de Amazon S3.
- Invocar una función de Lambda y verificar que el nombre del bucket de Amazon S3 que se devuelve coincide con el valor establecido por la variable de entorno.
- Actualizar la función de Lambda cambiando el nombre del bucket de Amazon S3 especificado por la variable de entorno.
- Invocar de nuevo la función de Lambda y verificar que el nombre del bucket de Amazon S3 que se devuelve coincide con el valor actualizado.

Paso 1: Preparación

Asegúrese de haber completado los pasos siguientes:

- Inscribirse en una cuenta de AWS y crear un usuario administrador en ella.
- Instalar y configurar la AWS CLI.

Para obtener instrucciones, consulte [Paso 1: Configuración de una cuenta de AWS y de la AWS CLI \(p. 199\)](#).

Paso 2: Configuración del entorno de Lambda

En esta sección, hará lo siguiente:

- Crear el paquete de implementación de la función de Lambda utilizando el código de muestra proporcionado.
- Crear un rol de ejecución de Lambda.
- Crear la función de Lambda cargando el paquete de implementación y, a continuación, probarla invocándola de forma manual.

Paso 2.1: Creación del paquete de implementación

El siguiente código de muestra lee la variable de entorno de una función de Lambda que devuelve el nombre de un bucket de Amazon S3.

1. Abra un editor de texto y copie el siguiente código:

```
var AWS = require('aws-sdk');

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

    var bucketName = process.env.S3_BUCKET;
    callback(null, bucketName);
}
```

2. Guarde el archivo como index.js.
3. Comprima el archivo index.js como Test_Environment_Variables.zip.

Paso 2.2: Creación de un rol de ejecución

Cree un rol de IAM (rol de ejecución) para especificarlo al crear la función de Lambda.

1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.
2. Siga los pasos de [Roles de IAM](#) en la Guía del usuario de IAM para crear un rol de IAM (rol de ejecución). Cuando siga los pasos para crear un rol, tenga en cuenta lo siguiente:
 - En Select Role Type, elija AWS Service Roles, seguido de AWS Lambda.
 - En Attach Policy, elija la política denominada AWSLambdaBasicExecutionRole.
3. Anote el nombre de recurso de Amazon (ARN) del rol de IAM. Necesitará este valor en el siguiente paso al crear la función de Lambda.

Paso 2.3: Creación y prueba de la función de Lambda

En esta sección, creará una función de Lambda que contiene una variable de entorno que especifica un bucket de Amazon S3 denominado `Test`. Al invocar la función, esta simplemente devuelve el nombre del bucket de Amazon S3. A continuación, debe actualizar la configuración cambiando el nombre del bucket de Amazon S3 por `Prod` y, al invocar la función de nuevo, esta devuelve el nombre actualizado del bucket de Amazon S3.

Para crear la función de Lambda, abra un símbolo del sistema y ejecute el siguiente comando `create-function` de la CLI de AWS de Lambda. Debe proporcionar la ruta del archivo .zip y el ARN del rol de ejecución. Observe que el parámetro `Runtime` utiliza `nodejs6.10`, pero también puede especificar `nodejs4.3`.

```
aws lambda create-function \
--region us-east-1 \
--function-name ReturnBucketName \
--zip-file fileb://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

Si lo desea, puede cargar el archivo .zip en un bucket de Amazon S3 en la misma región de AWS y, a continuación, especificar el nombre del objeto y el bucket en el comando anterior. Debe sustituir el parámetro `--zip-file` por el parámetro `--code`. Por ejemplo:

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

A continuación, ejecute el siguiente comando `invoke` de la CLI de Lambda para invocar la función. Tenga en cuenta que el comando solicita la ejecución asíncrona. También puede invocarla de forma síncrona especificando `RequestResponse` como valor del parámetro `invocation-type`.

```
aws lambda invoke \
--invocation-type Event \
--function-name ReturnBucketName \
--region us-east-1 \
--profile default \
outputfile.txt
```

La función de Lambda devolverá el nombre del bucket de Amazon S3 como "Test".

A continuación, ejecute el siguiente comando `update-function-configuration` de la CLI de Lambda para actualizar la variable de entorno de Amazon S3 apuntándola al bucket Prod.

```
aws lambda update-function-configuration
--function-name ReturnBucketName \
--region us-east-1 \
--environment Variables={S3_BUCKET=Prod} \
```

Ejecute el comando `aws lambda invoke` de nuevo con los mismos parámetros. Esta vez, la función de Lambda devolverá el nombre del bucket de Amazon S3 como Prod.

Creación de una función de Lambda utilizando variables de entorno para almacenar información confidencial

Además de para especificar las opciones de configuración para la función de Lambda, también puede utilizar las variables de entorno para almacenar información confidencial, como, por ejemplo, la contraseña de una base de datos, mediante [AWS Key Management Service](#) y los auxiliares de cifrado de la consola de Lambda. Para obtener más información, consulte [Cifrado de variables de entorno \(p. 100\)](#). El siguiente ejemplo muestra cómo hacerlo y también cómo utilizar KMS para descifrar esa información.

Este tutorial le mostrará cómo utilizar la consola de Lambda para cifrar una variable de entorno que contiene información confidencial, y proporciona código de muestra para descifrar dicha información que puede utilizar en sus funciones de Lambda.

Paso 1: Creación de la función de Lambda

1. Inicie sesión en la Consola de administración de AWS y abra la consola de AWS Lambda en <https://console.aws.amazon.com/lambda/>.
2. Seleccione Create a Lambda function.
3. En Select blueprint, elija el botón Author from scratch.
4. En Basic information, haga lo siguiente:
 - En Name*, especifique el nombre de la función de Lambda.
 - En Role*, elija Choose an existing role.
 - En Existing role*, elija lambda_basic_execution.

Note

Si la política del rol de ejecución no tiene el permiso `decrypt`, deberá añadirlo.

- Elija Create function.

Paso 2: Configuración de la función de Lambda

1. En Configuration, en Runtime, especifique nodejs6.10 o nodejs4.3.
2. En la sección Lambda function code puede utilizar la opción Edit code inline para sustituir el código del controlador de la función de Lambda por código personalizado.
3. Elija la pestaña Triggers. En la página Triggers, tiene la opción de elegir un servicio que activa automáticamente la función de Lambda; para ello, seleccione el botón Add trigger y, a continuación, elija el cuadro gris con puntos suspensivos (...) para mostrar una lista de los servicios disponibles. En este ejemplo, no configure un disparador y elija Configuration.
4. Elija la pestaña Monitoring. Esta página proporcionará métricas de CloudWatch inmediatas para las invocaciones de la función de Lambda, así como enlaces a otras guías útiles, incluida [Solución de problemas de aplicaciones basadas en Lambda \(p. 176\)](#).
5. Expanda la sección Environment variables.
6. Escriba el par clave-valor. Expanda la sección Encryption configuration. Observe que Lambda proporciona una clave de servicio predeterminada en KMS key to encrypt at rest que cifra la información una vez cargada. Si la información suministrada es confidencial, puede seleccionar adicionalmente la casilla Enable helpers for encryption in transit y proporcionar una clave personalizada. Esto enmascara el valor introducido y da como resultado una llamada a AWS KMS para cifrar dicho valor y devolverlo como `Ciphertext`. Si no ha creado una clave de KMS para su cuenta, se le proporcionará un enlace a la consola de AWS IAM para crear una. La cuenta debe tener permisos `encrypt` y `decrypt` para dicha clave. Observe que el botón Encrypt cambia a Decrypt después de elegirlo. Esto le da la opción de actualizar la información. Cuando haya terminado, elija el botón Encrypt.

El botón Code proporciona código de descifrado de muestra específico del tiempo de ejecución de la función de Lambda que puede utilizar en su aplicación.

Note

No puede utilizar la clave de servicio de Lambda predeterminada para cifrar información confidencial en el lado del cliente. Para obtener más información, consulte [Cifrado de variables de entorno \(p. 100\)](#).

Etiquetado de funciones de Lambda

Las funciones de Lambda pueden abarcar varias aplicaciones en diferentes regiones. Es posible simplificar el proceso de seguimiento de la frecuencia y el costo de cada invocación a la función mediante el uso de etiquetas. Las etiquetas son pares clave-valor que se asocian a los recursos de AWS para organizarlos mejor. Resultan de especial utilidad cuando se dispone de muchos recursos del mismo tipo, en el caso de AWS Lambda, funciones. El uso de etiquetas permite a los clientes con cientos de funciones de Lambda filtrar las que contienen la misma etiqueta para obtener acceso a un conjunto específico de ellas y analizarlas. Dos de las ventajas clave del etiquetado de funciones de Lambda son:

- Agrupación y filtrado: la aplicación de etiquetas permite utilizar la consola de Lambda o la CLI para aislar una lista de funciones de Lambda incluidas en una aplicación o un departamento de facturación específicos. Para obtener más información, consulte [Filtrado de funciones de Lambda etiquetadas \(p. 107\)](#).
- Asignación de costos: debido a que la compatibilidad de Lambda con el etiquetado está integrada en la facturación de AWS, es posible desglosar las facturas por categorías dinámicas y asignar funciones a centros de costos específicos. Por ejemplo, si etiqueta todas las funciones de Lambda con la clave "Departamento", todos los costos de AWS Lambda se pueden desglosar por departamento. Posteriormente, puede proporcionar un valor de departamento específico, como "Departamento 1" o "Departamento 2", para dirigir el costo de la invocación de las funciones al centro de costos apropiado. De esta forma, la asignación de costos será accesible a través de informes de facturación detallados, facilitándole la clasificación en categorías y el seguimiento de sus costos de AWS.

Temas

- [Etiquetado de funciones de Lambda para la facturación \(p. 105\)](#)
- [Aplicación de etiquetas a funciones de Lambda \(p. 105\)](#)
- [Filtrado de funciones de Lambda etiquetadas \(p. 107\)](#)
- [Restricciones de las etiquetas \(p. 108\)](#)

Etiquetado de funciones de Lambda para la facturación

Puede usar etiquetas para organizar la factura de AWS de modo que refleje su propia estructura de costos. Para ello, puede agregar claves de etiquetas cuyos valores se incluirán en el informe de asignación de costos. Para obtener más información acerca de la configuración de un informe de asignación de costos que incluya las claves de etiquetas que ha seleccionado como partidas en el informe, consulte el [Informe de asignación de costos mensual](#) en Acerca de la facturación de cuentas de AWS.

Para ver el costo de los recursos combinados, puede organizar la información de facturación basándose en las funciones que tienen los mismos valores de clave de etiqueta. Por ejemplo, puede etiquetar varias funciones de Lambda con un nombre de aplicación específico y luego organizar la información de facturación para ver el costo total de esa aplicación en distintos servicios. Para obtener más información, consulte [Uso de etiquetas de asignación de costos](#) en la Guía del usuario de Administración de costos y facturación de AWS.

Important

En AWS Lambda el único recurso que se puede etiquetar es una función. No se puede etiquetar un alias ni una versión específica de una función. Las invocaciones al alias o a la versión de una función se facturarán como invocaciones a la función original.

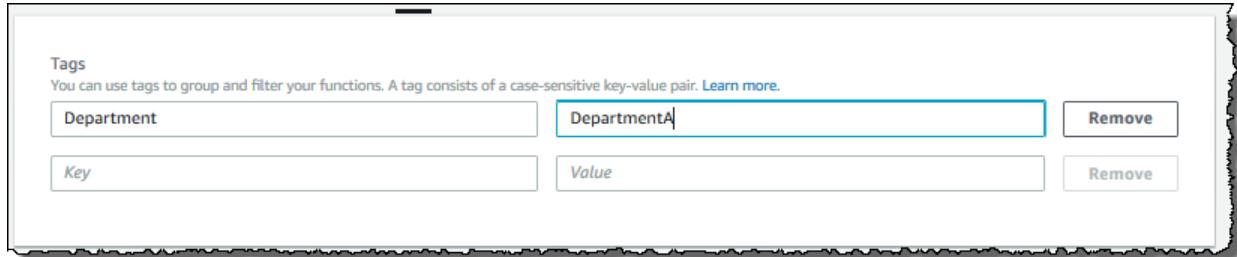
Aplicación de etiquetas a funciones de Lambda

La forma de etiquetar las funciones de Lambda depende de cómo se creen estas. Puede aplicarlas utilizando la consola de Lambda o la CLI, tal como se explica en las secciones siguientes:

- Aplicación de etiquetas a funciones de Lambda mediante la consola (p. 106)
- Aplicación de etiquetas a funciones de Lambda mediante la CLI (p. 106)

Aplicación de etiquetas a funciones de Lambda mediante la consola

Puede añadir etiquetas a una función en la sección Tags de la pestaña Configuration.



Para eliminar etiquetas de una función existente, abra la función, elija la sección Tags y, a continuación, elija el botón Remove junto al par clave-valor.



Aplicación de etiquetas a funciones de Lambda mediante la CLI

Cuando se crea una función de Lambda mediante el comando [CreateFunction \(p. 385\)](#), es posible añadir etiquetas rellenando el parámetro Tags. Si desea especificar varios valores de etiqueta, enciérralos entre comillas, tal como se muestra a continuación:

```
$ 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
```

Para aplicar o añadir más etiquetas a una función existente, puede utilizar la API [TagResource \(p. 454\)](#) y proporcionarla el ARN (nombre de recurso de Amazon) de la función de Lambda, junto con los pares clave-valor que componen las etiquetas.

```
$ aws lambda tag-resource \
--resource function arn \
--tags DEPARTMENT="Department C, Department D"
```

Por el contrario, si desea eliminar alguna o todas las etiquetas de una función de Lambda, utilice la API [UntagResource \(p. 456\)](#) y proporcione de nuevo el ARN (nombre de recurso de Amazon) de la función, junto con la lista de claves de etiquetas que se deben eliminar de esta.

```
$ aws lambda untag-resource \
--resource function arn \
```

```
--tagkeys list of tag keys to be removed
```

Filtrado de funciones de Lambda etiquetadas

Una vez que haya agrupado las funciones de Lambda mediante etiquetas, puede utilizar las capacidades de filtrado de la consola de Lambda o de la AWS CLI para verlas en función de sus requisitos específicos.

Filtrado de funciones de Lambda mediante la consola

La consola de Lambda contiene un campo de búsqueda que le permite filtrar la lista de funciones de acuerdo con un conjunto de atributos de función específicos, incluido Tags. Suponga que tiene dos funciones denominadas MyFunction y MyFunction2 que tienen una clave Tags denominada Department. Para ver estas funciones, elija el campo de búsqueda y observe el filtrado automático que incluye una lista de las claves Tags:

The screenshot shows the AWS Lambda 'Functions' list page with 25 results. On the left, there is a sidebar with checkboxes for 'Function attributes': Description, Function name, Runtime, Tags, and Department. The 'Tags' checkbox is selected. In the main area, there is a search bar with the placeholder 'Filter by tags and attributes or search by keyword'. Below it, a table lists three functions: 'mbda function.' (Node.js 6.10, 333 bytes), 'mbda function.' (Node.js 6.10, 333 bytes), and 'mbda function.' (Python 2.7, 360 bytes). The table has columns for 'Runtime' and 'Code size'.

Elija la clave Department. Lambda devolverá cualquier función que contenga dicha clave.

Ahora suponga que el valor de clave de la etiqueta MyFunction es "Department A" y el valor de clave de MyFunction2 es "Department B". Puede filtrar la búsqueda seleccionando el valor de la clave Department, en este caso Department A, tal como se muestra a continuación.

The screenshot shows the AWS Lambda 'Functions' list page with 25 results. The search bar now contains 'tag:Department:'. Below the search bar, a dropdown menu shows options: '(all values)', '(empty)', 'DepartmentA', and 'DepartmentB'. The 'DepartmentA' option is selected. The table lists two functions: 'mbda function.' (Node.js 6.10, 333 bytes) and 'mbda function.' (Node.js 6.10, 333 bytes). The table has columns for 'Runtime' and 'Code size'.

Esto solo devolverá MyFunction.

Puede seguir restringiendo la búsqueda incluyendo valores aceptados en los campos de la sección Function attributes, incluidos Description, Function name o Runtime.

Note

Se puede crear un máximo de 50 etiquetas por cada función de Lambda. Si elimina la función de Lambda, las etiquetas asociadas también se eliminarán.

Filtrado de funciones de Lambda mediante la CLI

Si desea ver las etiquetas que se aplican a una función de Lambda específica, puede utilizar cualquiera de los siguientes comandos de la API de Lambda:

- [ListTags \(p. 440\)](#): proporcione el ARN (nombre de recurso de Amazon) de la función de Lambda para ver una lista de las etiquetas asociadas a esta función:

```
$ aws lambda list-tags \
--resource function arn \
--region region \
--profile adminuser
```

- [GetFunction \(p. 411\)](#): proporcione el nombre de la función de Lambda para ver una lista de las etiquetas asociadas a esta función:

```
$ aws lambda get-function \
--function-name function name \
--region region \
--profile adminuser
```

También puede utilizar la API [GetResources](#) de AWS Tagging Service para filtrar los recursos por etiquetas. La API GetResources admite hasta 10 filtros, y cada uno contiene una clave de etiqueta y hasta 10 valores de etiqueta. Proporcione a GetResources un valor “ResourceType” si desea filtrar por tipos de recurso específicos. Para obtener más información acerca de AWS Tagging Service, consulte [Trabajar con grupos de recursos](#).

Restricciones de las etiquetas

Se aplican las siguientes restricciones a las etiquetas:

- Número máximo de etiquetas por recurso: 50
- Longitud máxima de la clave: 127 caracteres Unicode en UTF-8
- Longitud máxima del valor: 255 caracteres Unicode en UTF-8
- Las claves y los valores de las etiquetas distinguen entre mayúsculas y minúsculas.
- No utilice el prefijo `aws :` en los nombres o valores de las etiquetas, porque está reservado para uso de AWS. Los nombres y valores de etiquetas que tienen este prefijo no se pueden editar. Las etiquetas que tengan este prefijo no cuentan para el límite de etiquetas por recurso.
- Si se va utilizar su esquema de etiquetado en múltiples servicios y recursos, recuerde que otros servicios pueden tener otras restricciones sobre caracteres permitidos. Los caracteres generalmente permitidos son: letras, espacios y números representables en UTF-8, además de los siguientes caracteres especiales: `+ - = . _ : / @`.

Configuración de una función de Lambda para acceder recursos en una Amazon VPC

Normalmente, los recursos se crean dentro de Amazon Virtual Private Cloud (Amazon VPC) de forma que no se pueda tener acceso a ellos a través de la red pública de Internet. Estos recursos pueden ser recursos de servicios de AWS, como data warehouses de Amazon Redshift clústeres de Amazon ElastiCache o instancias de Amazon RDS. También pueden ser sus propios servicios que se ejecutan en sus propias instancias EC2. De forma predeterminada, no se puede obtener acceso a los recursos de una VPC desde una función de Lambda.

De forma predeterminada, AWS Lambda ejecuta el código de la función de forma segura dentro de una VPC. Sin embargo, si desea que la función de Lambda pueda tener acceso a los recursos dentro de la VPC privada, debe proporcionar información de configuración específica de VPC adicional que incluya los ID de subred y los ID de los grupos de seguridad de la VPC. AWS Lambda utiliza esta información para

configurar interfaces de red elásticas ([ENI](#)) que permiten que la función se conecte de forma segura a otros recursos privados dentro de la VPC privada.

Important

AWS Lambda no admite la conexión con recursos dentro de las VPC de tenencia dedicada. Para obtener más información, consulte [Instancias dedicadas](#).

Configuración de una función de Lambda para el acceso a Amazon VPC

Para añadir información de la VPC a la configuración de la función de Lambda, utilice el parámetro `VpcConfig`, bien en el momento de crear una función de Lambda (consulte [CreateFunction \(p. 385\)](#)), o bien añadiéndolo a la configuración de la función de Lambda existente (consulte [UpdateFunctionConfiguration \(p. 472\)](#)). A continuación se muestran algunos ejemplos realizados con la AWS CLI:

- El comando `create-function` de la CLI especifica el parámetro `--vpc-config` para proporcionar información de la VPC en el momento de la creación de una función de Lambda. Observe que el parámetro `--runtime` especifica `python3.6`. También puede utilizar `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

El rol de ejecución de la función de Lambda debe tener permisos para crear, describir y eliminar ENI. AWS Lambda proporciona una política de permisos, `AWSLambdaVPCAccessExecutionRole`, con permisos para las acciones de EC2 necesarias (`ec2:CreateNetworkInterface`, `ec2:DescribeNetworkInterfaces` y `ec2:DeleteNetworkInterface`) que puede utilizar al crear un rol. Puede examinar la política en la consola de IAM. No elimine este rol inmediatamente después de la ejecución de la función de Lambda. Existe un retraso entre el momento en que se ejecuta la función de Lambda y la eliminación de las ENI. Si elimina el rol inmediatamente después de la ejecución de la función, deberá encargarse de eliminar las ENI.

- El comando `update-function-configuration` de la CLI especifica el parámetro `--vpc-config` para añadir información de la VPC a la configuración de una función de Lambda existente.

```
$ aws lambda update-function-configuration \
--function-name ExampleFunction \
--vpc-config SubnetIds=comma-separated-vpc-subnet-ids,SecurityGroupIds=security-group-ids
```

Para eliminar la información relacionada con la VPC de la configuración de la función de Lambda, utilice la API `UpdateFunctionConfiguration` proporcionando una lista vacía de ID de subredes e ID de grupos de seguridad, tal como se muestra en el siguiente ejemplo de comando de la CLI.

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

Tenga en cuenta las siguientes consideraciones adicionales:

- Recomendamos que evite la resolución de DNS de nombres de host públicos para la VPC. Esto puede tardar algunos segundos en resolverse, lo que añade varios segundos de tiempo facturable a la solicitud. Por ejemplo, si la función de Lambda obtiene acceso a una instancia de Amazon RDS en la VPC, lance la instancia con la opción `no-publicly-accessible`.
- Cuando se añade a una configuración de VPC a una función de Lambda, esta solo puede tener acceso a los recursos de esa VPC. Si una función de Lambda necesita obtener acceso tanto a los recursos de la VPC como a la red pública de Internet, debe existir de una instancia de traducción de direcciones de red (NAT) dentro de la VPC.
- Cuando se configura una función de Lambda para que se ejecute en una VPC, se incurre en una penalización de arranque de ENI adicional. Esto significa que la resolución de direcciones puede retrasarse al intentar conectarse a los recursos de la red.

Acceso a Internet para funciones de Lambda

AWS Lambda utiliza la información de la VPC proporcionada para configurar [ENI](#) que permiten a la función de Lambda tener acceso a los recursos de la VPC. A cada ENI se le asigna una dirección IP privada dentro del rango de direcciones IP de las subredes especificadas, pero no se le asigna ninguna dirección IP pública. Por lo tanto, si la función de Lambda requiere acceso a Internet (por ejemplo, para acceder a los servicios de AWS que no tienen puntos de enlace de la VPC, como Kinesis), puede configurar una instancia de NAT en la VPC o utilizar la gateway NAT de Amazon VPC. Para obtener más información, consulte [Puerta de enlace NAT](#) en la Guía del usuario de Amazon VPC. No puede utilizar una gateway de Internet asociada a la VPC, ya que esto requiere que la ENI disponga de direcciones IP públicas.

Important

Si la función de Lambda necesita acceso a Internet, no la asocie a una subred pública ni a una subred privada sin acceso a Internet. En su lugar, asóciela únicamente a subredes privadas con acceso a Internet a través de una instancia NAT o una gateway NAT de Amazon VPC.

Directrices para la configuración de funciones de Lambda habilitadas para VPC

La función de Lambda se escala automáticamente en función del número de eventos que procesa. A continuación se indican directrices generales para configurar funciones de Lambda habilitadas para VPC que admitan el comportamiento de escalado.

- Si la función de Lambda tiene acceso a una VPC, debe asegurarse de que la VPC tiene suficiente capacidad de ENI para admitir los requisitos de escalado de la función de Lambda. Puede utilizar la siguiente fórmula para determinar aproximadamente la capacidad de ENI.

Projected peak concurrent executions * (Memory in GB / 1.5GB)

Donde:

- Projected peak concurrent execution (Número máximo previsto de ejecuciones simultáneas): utilice la información de [Ejecuciones simultáneas de la función de Lambda \(p. 189\)](#) para determinar este valor.
- Memory (Memoria): la cantidad de memoria que ha configurado para la función de Lambda.

- Las subredes que especifique deben tener un número de direcciones IP disponibles suficiente para el número de ENI.

También recomendamos que especifique al menos una subred en cada zona de disponibilidad de la configuración de la función de Lambda. De esta forma, la función de Lambda se puede ejecutar en otra zona de disponibilidad si una de ellas deja de funcionar o se queda sin direcciones IP.

Note

Si la VPC no dispone de suficientes ENI o direcciones IP de subredes, la función de Lambda no se escalará a medida que crezca el número de solicitudes y aumentarán los errores de la función. Actualmente, AWS Lambda no registra en CloudWatch Logs los errores originados por un número insuficiente de ENI o direcciones IP. Si experimenta un aumento del número de errores sin los correspondientes logs de CloudWatch Logs, puede invocar la función de Lambda de forma síncrona para obtener las respuestas de error (por ejemplo, puede probar la función de Lambda en la consola de AWS Lambda debido a que esta invoca la función de forma síncrona y muestra los errores).

Tutoriales: Configuración de una función de Lambda para acceder los recursos de una Amazon VPC

En esta sección se proporcionan tutoriales de ejemplo completos en los que creará y configurará una función de Lambda para obtener acceso a los recursos de una Amazon VPC, como un clúster de Amazon ElastiCache o una instancia de base de datos de Amazon RDS.

Temas

- [Tutorial: Configuración de una función de Lambda para obtener acceso a Amazon ElastiCache en una Amazon VPC \(p. 111\)](#)
- [Tutorial: Configuración de una función de Lambda para obtener acceso a Amazon RDS en una Amazon VPC \(p. 115\)](#)

Tutorial: Configuración de una función de Lambda para obtener acceso a Amazon ElastiCache en una Amazon VPC

En este tutorial, aprenderá a hacer lo siguiente:

- Crear un clúster de Amazon ElastiCache en la Amazon Virtual Private Cloud (Amazon VPC) predeterminada en la región us-east-1. Para obtener más información acerca de Amazon ElastiCache, consulte [Amazon ElastiCache](#).
- Crear una función de Lambda para obtener acceso al clúster de ElastiCache. Al crear la función de Lambda debe proporcionar los ID de subred de Amazon VPC, así como un grupo de seguridad de VPC para que la función de Lambda pueda obtener acceso a los recursos de la VPC. En este tutorial, con fines ilustrativos, la función de Lambda genera un UUID, lo escribe en la caché y lo recupera de esta.
- Invocar la función de Lambda manualmente y verificar que ha tenido acceso al clúster de ElastiCache en la VPC.

Important

Este tutorial utiliza la Amazon VPC predeterminada de la región us-east-1 de su cuenta. Para obtener más información acerca de Amazon VPC, consulte [Primeros pasos con Amazon VPC](#) en la Guía del usuario de Amazon VPC.

Paso siguiente

[Paso 1: Crear un clúster de ElastiCache \(p. 112\)](#)

Paso 1: Crear un clúster de ElastiCache

En este paso, creará un clúster de ElastiCache en la Amazon VPC predeterminada de la región us-east-1 de su cuenta.

1. Ejecute el siguiente comando de la AWS CLI para crear un clúster de Memcached en la VPC predeterminada de la región us-east-1 de su cuenta.

```
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
```

Puede buscar el grupo de seguridad de VPC predeterminado en la consola de VPC en Security Groups. La función de Lambda de ejemplo añadirá y recuperará un elemento de este clúster.

También puede lanzar un clúster de caché mediante la consola de Amazon ElastiCache. Para obtener instrucciones, consulte [Introducción a Amazon ElastiCache](#) en la Guía del usuario de Amazon ElastiCache.

2. Anote el punto de enlace de configuración para el clúster de caché que ha lanzado. Puede obtenerlo de la consola de Amazon ElastiCache. Especificará este valor en el código de la función de Lambda en la siguiente sección.

Paso siguiente

[Paso 2: Creación de una función de Lambda \(p. 112\)](#)

Paso 2: Creación de una función de Lambda

En este paso, hará lo siguiente:

- Crear un paquete de implementación de la función de Lambda utilizando el código de muestra proporcionado.
- Crear un rol de IAM (rol de ejecución). En el momento de cargar el paquete de implementación, debe especificar este rol para que Lambda pueda asumirlo y ejecutar la función en su nombre.

La política de permisos concede permisos de AWS Lambda para configurar interfaces de red elásticas (ENI) para permitir que la función de Lambda tenga acceso a los recursos de la VPC. En este ejemplo, la función de Lambda obtiene acceso a un clúster de ElastiCache en la VPC.

- Crear la función de Lambda cargando el paquete de implementación.

Temas

- [Paso 2.1: Creación de un paquete de implementación \(p. 113\)](#)
- [Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 114\)](#)
- [Paso 2.3: Creación de la función de Lambda \(carga del paquete de implementación\) \(p. 114\)](#)

Paso 2.1: Creación de un paquete de implementación

Note

En este momento, el código de ejemplo de la función de Lambda solo se proporciona en Python.

Python

El siguiente ejemplo de código de Python lee y escribe un elemento en el clúster de ElastiCache.

1. Abrir un editor de texto y, a continuación, copie el siguiente código.

Note

La instrucción `from __future__ import print_function` permite escribir código compatible con Python versión 2 o 3. Si utiliza la versión 3.6 del tiempo de ejecución, no es necesario incluirla.

```
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 == str(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: " + str(uuid_obtained)
```

2. Guarde el archivo como `app.py`.
3. Instale las siguientes dependencias de bibliotecas utilizando pip:
 - `pymemcache`: el código de la función de Lambda utiliza esta biblioteca para crear un objeto `HashClient` que establezca y obtenga elementos de memcache (consulte [pymemcache](#)).
 - `elasticache-auto-discovery`: la función de Lambda utiliza esta biblioteca para obtener los nodos del clúster de Amazon ElastiCache (consulte [elasticache-auto-discovery](#)).
4. Comprima todos estos archivos en un archivo denominado `app.zip` para crear el paquete de implementación. Para obtener instrucciones paso a paso, consulte [Creación de un paquete de implementación \(Python\) \(p. 77\)](#).

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 114\)](#)

Paso 2.2: Creación del rol de ejecución (rol de IAM)

En este paso, creará un rol de AWS Identity and Access Management (IAM) con el tipo de rol y la política de permisos de acceso predefinidos que se indican a continuación:

- AWS Lambda (rol de servicio de AWS): este rol concede permisos a AWS Lambda para asumir el rol.
- AWSLambdaVPCAccessExecutionRole (política de permisos de acceso): es la política que se asocia al rol. La política otorga permisos para las acciones de EC2 que AWS Lambda necesita para administrar las interfaces de red elásticas (ENI). Puede ver esta política administrada de AWS en la consola de IAM.

Para obtener más información sobre los roles de IAM, consulte [Roles de IAM](#) en la Guía del usuario de IAM. Utilice el siguiente procedimiento para crear el rol de IAM.

Para crear un rol de IAM (rol de ejecución)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.
2. Siga los pasos de [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM para crear un rol de IAM (rol de ejecución). Cuando siga los pasos para crear un rol, tenga en cuenta lo siguiente:
 - En Role Name, utilice un nombre que sea único dentro de la cuenta de AWS (por ejemplo, lambda-vpc-execution-role).
 - En Select Role Type, elija AWS Service Roles, seguido de AWS Lambda. Esto garantiza que los permisos del servicio de AWS Lambda asuman el rol.
 - En Attach Policy, elija AWSLambdaVPCAccessExecutionRole. Los permisos de esta política son suficientes para la función de Lambda de este tutorial.
3. Anote el ARN del rol. Lo necesitará en el siguiente paso al crear la función de Lambda.

Paso siguiente

[Paso 2.3: Creación de la función de Lambda \(carga del paquete de implementación\) \(p. 114\)](#)

Paso 2.3: Creación de la función de Lambda (carga del paquete de implementación)

En este paso, creará la función de Lambda (`AccessMemCache`) utilizando el comando `create-function` de la AWS CLI.

En el símbolo del sistema, ejecute el siguiente comando `create-function` de la CLI de Lambda utilizando el perfil adminuser.

Debe actualizar el siguiente comando `create-function` proporcionando la ruta del archivo .zip y el ARN del rol de ejecución. El valor del parámetro `--runtime` puede ser `python3.6`, `python2.7`, `nodejs` o `java8`, dependiendo del lenguaje que utilice para crear el código.

Note

En este momento, el código de ejemplo de la función de Lambda solo se proporciona en Python.

```
$ 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
```

Puede encontrar los ID de subred y el ID del grupo de seguridad predeterminado de la VPC en la consola de VPC.

Si lo desea, puede cargar el archivo .zip en un bucket de Amazon S3 en la misma región de AWS y, a continuación, especificar el nombre del objeto y el bucket en el comando anterior. Debe reemplazar el parámetro --zip-file por el parámetro --code, como se muestra a continuación:

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

Note

También puede crear la función de Lambda utilizando la consola de AWS Lambda. Al crear la función de Lambda, elija una VPC para la función y, a continuación, seleccione las subredes y los grupos de seguridad en los campos proporcionados.

Paso siguiente

[Paso 3: Prueba de la función de Lambda \(invocación manual\) \(p. 115\)](#)

Paso 3: Prueba de la función de Lambda (invocación manual)

En este paso, invocará la función de Lambda manualmente utilizando el comando `invoke`. Cuando la función de Lambda se ejecuta, genera un UUID y lo escribe en el clúster de ElastiCache especificado en el código de la función de Lambda. A continuación, la función de Lambda recupera el elemento de la caché.

1. Invoque la función de Lambda (`AccessMemCache`) utilizando el comando `invoke` de AWS Lambda.

```
$ aws lambda invoke \
--function-name AccessMemCache \
--region us-east-1 \
--profile adminuser \
output.txt
```

2. Compruebe que la función de Lambda se ha ejecutado correctamente del modo siguiente:

- Revise el archivo `output.txt`.
- Revise los resultados en la consola de AWS Lambda.
- Verifique los resultados en CloudWatch Logs.

¿Qué se hace ahora?

Ahora que ha creado una función de Lambda que tiene acceso a un clúster de ElastiCache en la VPC, puede hacer que la función se invoque en respuesta a eventos. Para obtener más información acerca de la configuración de orígenes de eventos y ver ejemplos, consulte [Casos de uso \(p. 214\)](#).

Tutorial: Configuración de una función de Lambda para obtener acceso a Amazon RDS en una Amazon VPC

En este tutorial, aprenderá a hacer lo siguiente:

- Lanzar una instancia del motor de base de datos MySQL de Amazon RDS en la Amazon VPC predeterminada. En la instancia de MySQL, debe crear una base de datos (ExampleDB) con una tabla de muestra (Employee). Para obtener más información acerca de Amazon RDS, consulte [Amazon RDS](#).
- Crear una función de Lambda para obtener acceso a la base de datos ExampleDB, crear una tabla (Employee), añadir algunos registros y recuperar los registros de la tabla.
- Invocar la función de Lambda manualmente y verificar los resultados de la consulta. De este modo, se comprueba que la función de Lambda ha podido tener acceso a la instancia de MySQL de RDS en la VPC.

Important

Este tutorial utiliza la Amazon VPC predeterminada de la región us-east-1 de su cuenta. Para obtener más información acerca de Amazon VPC, consulte [Primeros pasos con Amazon VPC](#) en la Guía del usuario de Amazon VPC.

Paso siguiente

[Paso 1: Creación de una instancia de MySQL y de la base de datos ExampleDB en Amazon RDS \(p. 116\)](#)

Paso 1: Creación de una instancia de MySQL y de la base de datos ExampleDB en Amazon RDS

En este tutorial, la función de Lambda del ejemplo crea una tabla (Employee), introduce algunos registros y, a continuación, recupera los registros. La tabla creada por la función de Lambda tiene el siguiente esquema:

```
Employee(EmpID, Name)
```

Donde EmpID es la clave principal. Ahora, debe añadir algunos registros a esta tabla.

En primer lugar, lance una instancia de MySQL de RDS en la VPC predeterminada con la base de datos ExampleDB. Si ya tiene una instancia de MySQL de RDS ejecutándose en la VPC predeterminada, omita este paso.

Important

En este tutorial se utiliza el motor de base de datos MySQL de RDS lanzado en la VPC predeterminada de la región us-east-1.

Puede lanzar una instancia de MySQL de RDS utilizando uno de los siguientes métodos:

- Siga las instrucciones de [Crear una instancia de base de datos MySQL y conectarse a una base de datos en una instancia de base de datos MySQL](#) en la Guía del usuario de Amazon Relational Database Service.
- Utilice el siguiente comando de AWS CLI:

```
$ 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
```

Anote el nombre de la base de datos, el nombre de usuario y la contraseña. También necesita la dirección del host (punto de enlace) de la instancia de base de datos, que puede obtener de la consola de RDS (es posible que tenga que esperar hasta que la instancia esté disponible y el valor del punto de enlace aparezca en la consola).

Paso siguiente

[Paso 2: Creación de una función de Lambda \(p. 117\)](#)

Paso 2: Creación de una función de Lambda

En este paso, hará lo siguiente:

- Crear un paquete de implementación de la función de Lambda utilizando el código de muestra proporcionado.
- Crear un rol de IAM (rol de ejecución) que especificará al crear la función de Lambda. Este es el rol que AWS Lambda asume al ejecutar la función de Lambda.

La política de permisos asociada a este rol concede permisos de AWS Lambda para configurar interfaces de red elásticas (ENI) para permitir que la función de Lambda tenga acceso a los recursos de la VPC.

- Crear la función de Lambda cargando el paquete de implementación.

Temas

- [Paso 2.1: Creación de un paquete de implementación \(p. 117\)](#)
- [Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 118\)](#)
- [Paso 2.3: Creación de la función de Lambda \(carga del paquete de implementación\) \(p. 119\)](#)

Paso 2.1: Creación de un paquete de implementación

Note

En este momento, el código de ejemplo de la función de Lambda solo se proporciona en Python.

Python

En el siguiente ejemplo de código de Python se ejecuta una consulta SELECT en la tabla Employee de la instancia de RDS de MySQL que ha creado en la VPC. El código crea una tabla en la base de datos ExampleDB, añade registros de muestra y recupera los registros.

1. Abrir un editor de texto y, a continuación, copie el siguiente código.

```
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
    """

    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

Recomendamos que `pymysql.connect()` se ejecute fuera del controlador, tal como se muestra, para un mejor desempeño.

2. Guarde el archivo como `app.py`.
3. Instale las siguientes dependencias de bibliotecas utilizando pip:
 - `pymysql`: el código de la función de Lambda utiliza esta biblioteca para obtener acceso a la instancia de MySQL (consulte [PyMySQL](#)).
4. Cree un archivo de configuración que contenga la siguiente información y guárdelo como `rds_config.py`:

```
#config file containing credentials for rds mysql instance
db_username = "username"
db_password = "password"
db_name = "databasename"
```

5. Comprima todos estos archivos en un archivo denominado `app.zip` para crear el paquete de implementación. Para obtener instrucciones paso a paso, consulte [Creación de un paquete de implementación \(Python\) \(p. 77\)](#).

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 118\)](#)

Paso 2.2: Creación del rol de ejecución (rol de IAM)

En este paso, se crea un rol de ejecución (rol de IAM) para la función de Lambda utilizando el tipo de rol predefinido y la política de permisos de acceso que se indican a continuación.:

- AWS Lambda: (rol de servicio de AWS) este rol concede permisos a AWS Lambda para asumir el rol.
- AWSLambdaVPCAccessExecutionRole (política de permisos de acceso): este rol concede permisos a AWS Lambda para que las acciones de EC2 puedan crear las ENI y para que la función de Lambda pueda acceder a los recursos de la VPC y a las acciones de CloudWatch Logs para escribir los logs.

Para obtener más información sobre los roles de IAM, consulte [Roles de IAM](#) en la Guía del usuario de IAM. Utilice el siguiente procedimiento para crear el rol de IAM.

Para crear un rol de IAM (rol de ejecución)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.
2. Siga los pasos de [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM para crear un rol de IAM (rol de ejecución). Cuando siga los pasos para crear un rol, tenga en cuenta lo siguiente:
 - En Role Name, utilice un nombre que sea único dentro de la cuenta de AWS (por ejemplo, lambda-vpc-execution-role).
 - En Select Role Type, elija AWS Service Roles, seguido de AWS Lambda. Esto garantiza que los permisos del servicio de AWS Lambda asuman el rol.
 - En Attach Policy, elija AWSLambdaVPCAccessExecutionRole. Los permisos de esta política son suficientes para la función de Lambda de este tutorial.
3. Anote el ARN del rol. Lo necesitará en el siguiente paso al crear la función de Lambda.

Paso siguiente

[Paso 2.3: Creación de la función de Lambda \(carga del paquete de implementación\) \(p. 119\)](#)

[Paso 2.3: Creación de la función de Lambda \(carga del paquete de implementación\)](#)

En este paso, creará la función de Lambda (`ReadMySqlTable`) utilizando el comando `create-function` de la AWS CLI.

En el símbolo del sistema, ejecute el siguiente comando `create-function` de la CLI de Lambda utilizando el perfil `adminuser`.

Debe actualizar el siguiente comando `create-function` proporcionando la ruta del archivo .zip y el ARN del rol de ejecución. El valor del parámetro `--runtime` puede ser `python2.7`, `nodejs` o `java8`, dependiendo del lenguaje que utilice para crear el código.

Note

En este momento, el código de ejemplo de la función de Lambda solo se proporciona en Python. Puede especificar `python3.6` o `python2.7` para el parámetro `--runtime`.

```
$ 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
```

Si lo desea, puede cargar el archivo .zip en un bucket de Amazon S3 en la misma región de AWS y, a continuación, especificar el nombre del objeto y el bucket en el comando anterior. Debe reemplazar el parámetro `--zip-file` por el parámetro `--code`, como se muestra a continuación:

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

Note

También puede crear la función de Lambda utilizando la consola AWS Lambda (utilice los valores de los parámetros mostrados en el comando CLI anterior).

Paso siguiente

[Paso 3: Prueba de la función de Lambda \(invocación manual\) \(p. 120\)](#)

Paso 3: Prueba de la función de Lambda (invocación manual)

En este paso, invocará la función de Lambda manualmente utilizando el comando `invoke`. Cuando se ejecuta la función de Lambda, realiza la consulta SELECT en la tabla Employee de la instancia de MySQL de RDS e imprime los resultados (estos resultados también van a CloudWatch Logs).

1. Invoque la función de Lambda (`ReadMySqlTable`) utilizando el comando `invoke` de AWS Lambda.

```
$ aws lambda invoke \
--function-name CreateTableAddRecordsAndRead \
--region us-east-1 \
--profile adminuser \
output.txt
```

2. Compruebe que la función de Lambda se ha ejecutado correctamente del modo siguiente:

- Revise el archivo `output.txt`.
- Revise los resultados en la consola de AWS Lambda.
- Verifique los resultados en CloudWatch Logs.

Solución de problemas y monitorización de funciones de AWS Lambda con Amazon CloudWatch

AWS Lambda monitoriza automáticamente las funciones de Lambda en su nombre, e informa sobre las métricas a través de Amazon CloudWatch. Para ayudarle a monitorizar el código mientras se ejecuta, Lambda controla automáticamente el número de solicitudes, la latencia por solicitud y el número de solicitudes que generan un error, y publica las métricas correspondientes en CloudWatch. Puede utilizar estas métricas para configurar alarmas personalizadas de CloudWatch. Para obtener más información sobre CloudWatch, consulte la [Guía del usuario de Amazon CloudWatch](#).

Puede ver las tasas de solicitudes y las tasas de errores para cada una de sus funciones de Lambda utilizando la consola de AWS Lambda, la consola de CloudWatch y otros recursos de Amazon Web Services (AWS). En los siguientes temas se describen las métricas de CloudWatch para las funciones de Lambda y cómo acceder a ellas.

- [Acceso a las métricas de Amazon CloudWatch para AWS Lambda \(p. 122\)](#)
- [Métricas de AWS Lambda \(p. 125\)](#)

Puede introducir instrucciones de registro en el código comprobar que el código está funcionando según lo previsto. Lambda se integra automáticamente con Amazon CloudWatch Logs y envía todos los logs generados por el código a un grupo de CloudWatch Logs asociado a una función de Lambda (`/aws/lambda/<nombre de la función>`). Para obtener más información acerca de los grupos de logs y cómo acceder a ellos a través de la consola de CloudWatch, consulte [Monitoring System, Application, and Custom Log Files](#) en la Guía del usuario de Amazon CloudWatch. Para obtener más información acerca de cómo acceder a las entradas de log de CloudWatch, consulte [Acceso a los logs de Amazon CloudWatch para AWS Lambda \(p. 124\)](#).

Note

Si el código de la función de Lambda se está ejecutando, pero no se genera ningún dato de log después de varios minutos, podría significar que el rol de ejecución de la función de Lambda no ha concedido permisos para escribir datos de log a CloudWatch Logs. Para obtener información acerca de cómo asegurarse de que se ha configurado correctamente el rol de ejecución para conceder estos permisos, consulte [Administración de permisos mediante un rol de IAM \(rol de ejecución\) \(p. 193\)](#).

Escenarios de solución de problemas de AWS Lambda

En estas secciones se describen ejemplos de cómo monitorizar y solucionar problemas relacionados con las funciones de Lambda utilizando las funciones de registro y monitorización de CloudWatch.

Escenario de solución de problemas n.º 1: La función de Lambda no funciona según lo previsto

En este escenario, acaba de terminar el [Tutorial: Uso de AWS Lambda con Amazon S3 \(p. 215\)](#). Sin embargo, la función de Lambda que ha creado para cargar una imagen en miniatura en Amazon S3 cuando se crea un objeto de S3 no funciona según lo previsto. Al cargar objetos en Amazon S3, observa que las imágenes en miniatura no se están cargando. Puede solucionar este problema de las siguientes maneras.

Para determinar por qué la función de Lambda no funciona según lo previsto

1. Compruebe el código y verifique que funciona correctamente. Si aumenta la tasa de errores, significa que no es así.

Puede probar el código localmente como haría con cualquier otra función de Node.js, o probarlo en la consola de Lambda utilizando la funcionalidad de invocación de prueba de la consola, o bien usar el comando `Invoke` de la AWS CLI. Cada vez que se ejecuta el código como respuesta a un evento, se escribe una entrada de log en el grupo de logs asociado a la función de Lambda, que es `/aws/lambda/<nombre de la función>`.

A continuación se presentan algunos ejemplos de errores que podrían aparecer en los logs:

- Si ve un rastro de stack en el log, probablemente hay un error en el código. Revise el código y depure el error al que hace referencia el rastro de stack.
- Si ve un error `permissions denied` en el log, significa que el rol de IAM que ha proporcionado como rol de ejecución no tiene los permisos necesarios. Compruebe el rol de IAM y verifique que tiene todos los permisos necesarios para obtener acceso a cualquier recurso de AWS al que haga referencia el código. Para asegurarse de que ha configurado correctamente el rol de ejecución, consulte [Administración de permisos mediante un rol de IAM \(rol de ejecución\) \(p. 193\)](#).
- Si ve un error `timeout exceeded` en el log, significa que la configuración de tiempo de espera no es suficiente para el tiempo que tarda en ejecutarse el código de la función. Esto puede deberse a que el tiempo de espera es demasiado bajo, o a que el código tarda demasiado en ejecutarse.

- Si ve un error `memory exceeded` en el log, significa que la configuración de memoria no es suficiente. Establezca a un valor más grande. Para obtener información acerca de la cantidad máxima de memoria, consulte [CreateFunction \(p. 385\)](#). Si cambia la configuración de memoria, también puede cambiar la forma en que se le cobra por la duración. Para obtener información sobre precios, consulte el [sitio web del producto AWS Lambda](#).
2. Compruebe la función de Lambda y verifique que está recibiendo solicitudes.

Incluso si el código de la función funciona según lo previsto y responde correctamente a las invocaciones de prueba, es posible que la función no esté recibiendo solicitudes desde Amazon S3. Si Amazon S3 puede invocar la función, debería ver un aumento en las métricas de las solicitudes de CloudWatch. Si no observa un aumento de las solicitudes de CloudWatch, compruebe la política de permisos de acceso asociada a la función.

Escenario de solución de problemas n.º 2: Aumento de la latencia en la ejecución de la función de Lambda

En este escenario, acaba de terminar el [Tutorial: Uso de AWS Lambda con Amazon S3 \(p. 215\)](#). Sin embargo, la función de Lambda que ha creado para cargar una imagen en miniatura en Amazon S3 cuando se crea un objeto de S3 no funciona según lo previsto. Al cargar objetos en Amazon S3, observa que las imágenes en miniatura se están cargando, pero el código está tardando mucho más tiempo del esperado en ejecutarse. Puede buscar la solución a este problema de dos maneras distintas. Por ejemplo, puede monitorizar la métrica de latencia de CloudWatch para la función de Lambda con objeto de ver si la latencia está aumentando. También puede que observe un aumento de la métrica de errores de CloudWatch para la función de Lambda, que puede ser debido a errores de tiempo de espera.

Para determinar por qué existe una mayor latencia en la ejecución de una función de Lambda

1. Pruebe el código con distintos ajustes de memoria.

Si el código tarda demasiado en ejecutarse, podría ser debido a que no tiene suficientes recursos informáticos para ejecutar su lógica. Aumente la memoria asignada a la función y pruebe el código de nuevo, utilizando la funcionalidad de invocación de prueba de la consola de Lambda. Puede consultar la memoria utilizada, el tiempo que tarda en ejecutarse del código, y la memoria asignada en las entradas de log de la función. Si cambia la configuración de memoria, puede cambiar la forma en que se le cobra por la duración. Para obtener información acerca de los precios, consulte [AWS Lambda](#).

2. Investigue el origen del cuello de botella de ejecución utilizando los logs.

Puede probar el código localmente como haría con cualquier otra función de Node.js, o probarlo desde Lambda utilizando la funcionalidad de invocación de pruebas de la consola de Lambda o mediante el comando `asyncInvoke` de la AWS CLI. Cada vez que se ejecuta el código como respuesta a un evento, se escribe una entrada de log en el grupo de logs asociado a la función de Lambda, que se denomina `/aws/lambda/<nombre de la función>`. Añada instrucciones de registro en varias partes del código, como, por ejemplo, en las llamadas a otros servicios, para ver la cantidad de tiempo que se tarda en ejecutar las distintas partes del código.

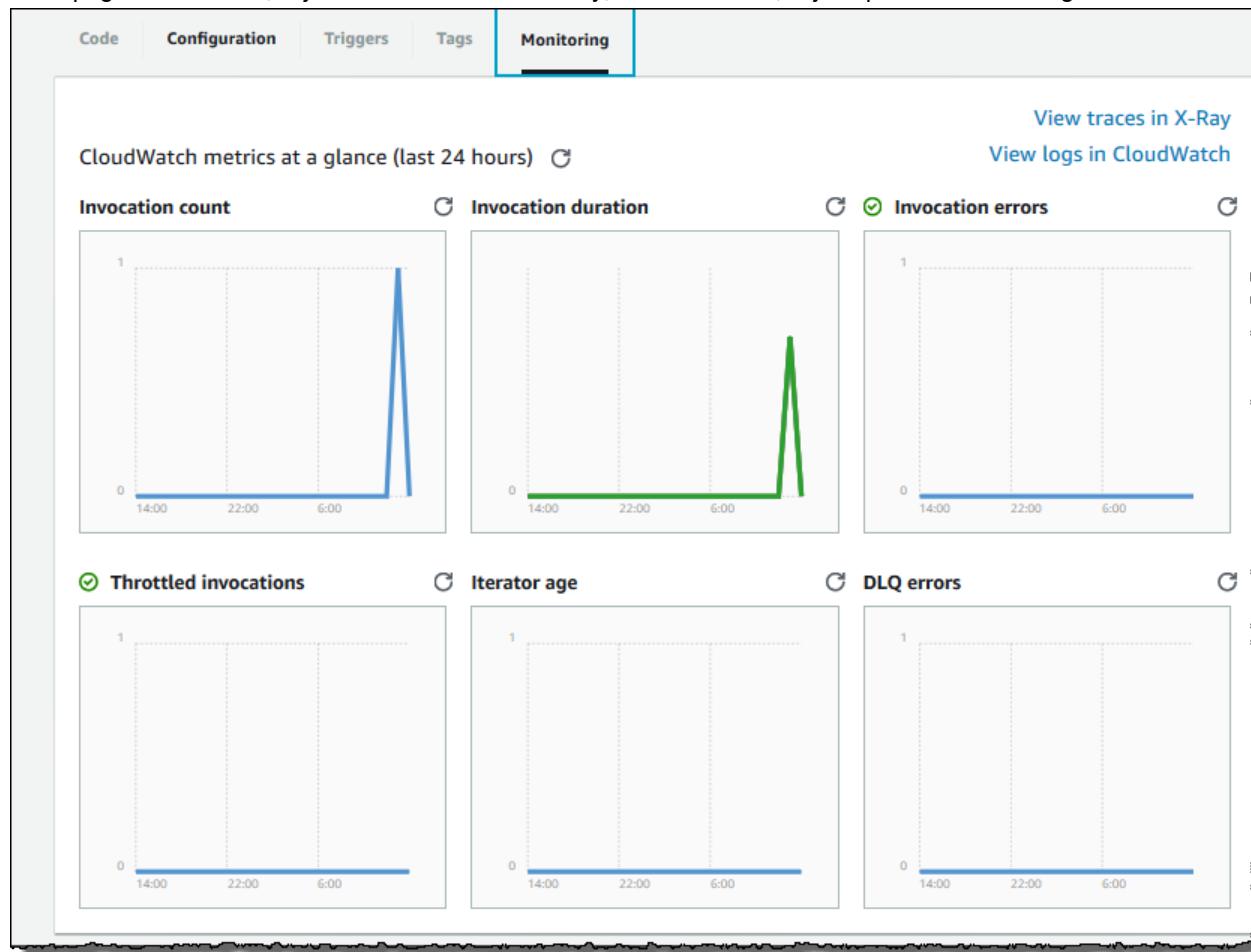
Acceso a las métricas de Amazon CloudWatch para AWS Lambda

AWS Lambda monitoriza automáticamente las funciones en su nombre, e informa sobre las métricas a través de Amazon CloudWatch. Estas métricas incluyen el total de solicitudes, la latencia y las tasas de errores. Para obtener más información acerca de las métricas de Lambda, consulte [Métricas de AWS Lambda \(p. 125\)](#). Para obtener más información acerca de CloudWatch, consulte la [Guía del usuario de Amazon CloudWatch](#).

Puede monitorizar las métricas de Lambda y ver los logs mediante la consola de Lambda, la consola de CloudWatch, la AWS CLI o la API de CloudWatch. Los siguientes procedimientos le muestran cómo obtener acceso a las métricas a través de los distintos métodos descritos a continuación.

Para obtener acceso a las métricas mediante la consola de Lambda

1. Inicie sesión en la Consola de administración de AWS y abra la consola de AWS Lambda en <https://console.aws.amazon.com/lambda/>.
2. Si todavía no ha creado una función de Lambda, consulte [Introducción \(p. 199\)](#).
3. En la página Functions , elija el nombre de la función y, a continuación, elija la pestaña Monitoring.



Se muestra una representación gráfica de las métricas de la función de Lambda.

4. Para ver los logs, elija View logs in CloudWatch.

Para obtener acceso a las métricas mediante la consola de CloudWatch

1. Abra la consola de CloudWatch en <https://console.aws.amazon.com/cloudwatch/>.
2. En la barra de navegación, elija una región.
3. En el panel de navegación, seleccione Metrics.
4. En el panel CloudWatch Metrics by Category, elija Lambda Metrics.
5. (Opcional) En el panel de gráficos, seleccione una estadística y un periodo de tiempo y, a continuación, cree una alarma de CloudWatch utilizando estos ajustes.

Para obtener acceso a las métricas mediante la AWS CLI

Utilice los comandos [list-metrics](#) y [get-metric-statistics](#).

Para obtener acceso a las métricas mediante la CLI de CloudWatch

Utilice los comandos [mon-list-metrics](#) y [mon-get-stats](#).

Para obtener acceso a las métricas mediante la API de CloudWatch

Utilice las operaciones [ListMetrics](#) y [GetMetricStatistics](#).

Acceso a los logs de Amazon CloudWatch para AWS Lambda

AWS Lambda monitoriza automáticamente las funciones de Lambda en su nombre, e informa sobre las métricas a través de Amazon CloudWatch. Para ayudarle a solucionar problemas relacionados con los errores de una función, Lambda registra todas las solicitudes gestionadas por la función, y almacena automáticamente los logs generados por el código a través de Amazon CloudWatch Logs.

Puede introducir instrucciones de registro en el código comprobar que el código está funcionando según lo previsto. Lambda se integra automáticamente con CloudWatch Logs y envía todos los logs generados por el código a un grupo de CloudWatch Logs asociado a una función de Lambda que se denomina /aws/lambda/<*nombre de la función*>. Para obtener más información acerca de los grupos de logs y cómo acceder a ellos a través de la consola de CloudWatch, consulte [Monitoring System, Application, and Custom Log Files](#) en la Guía del usuario de Amazon CloudWatch.

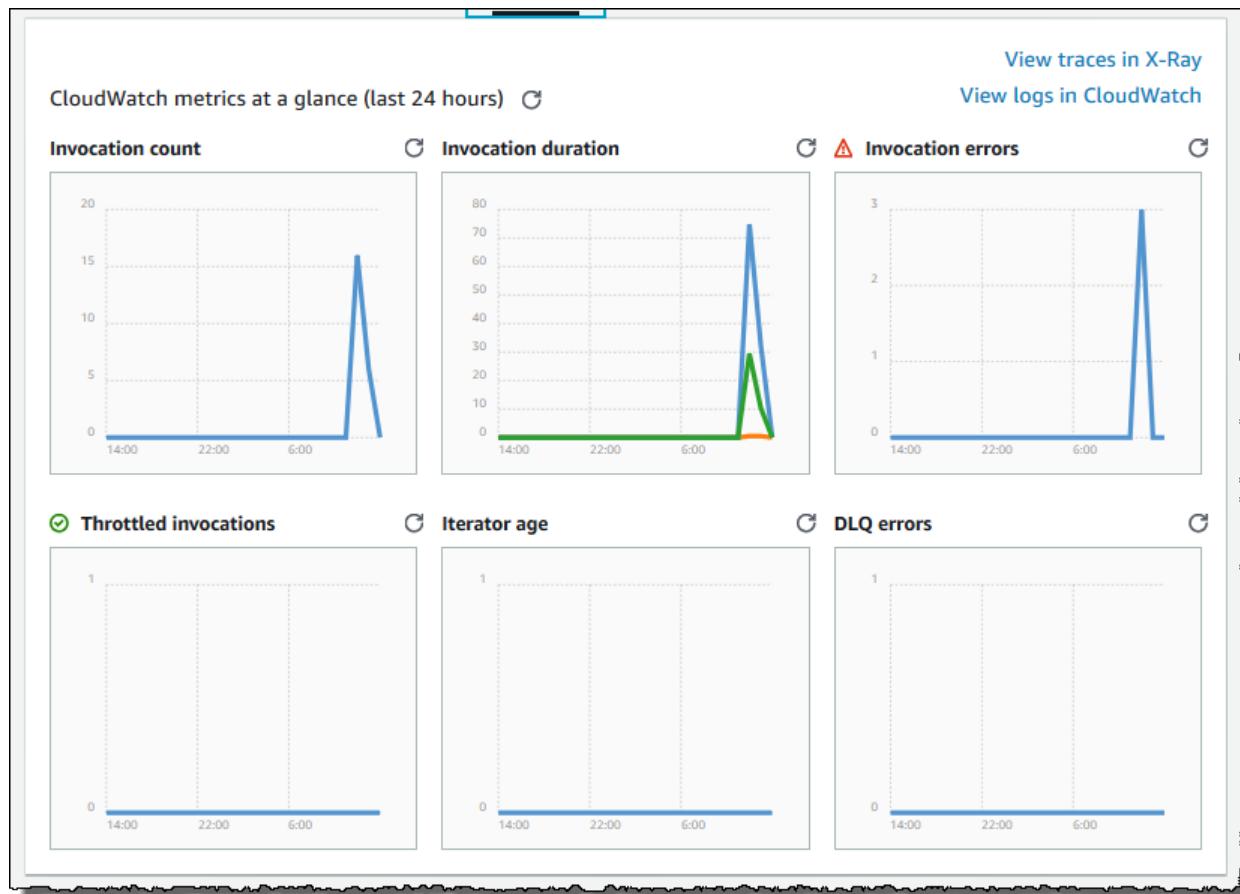
Puede ver los logs de Lambda mediante la consola de Lambda, la consola de CloudWatch, la AWS CLI o la API de CloudWatch. El siguiente procedimiento muestra cómo ver los logs utilizando la consola de Lambda.

Note

No se aplican cargos adicionales por utilizar los logs de Lambda; no obstante, sí se aplican los cargos estándar de CloudWatch Logs. Para obtener más información, consulte los [precios de CloudWatch](#).

Para ver los logs mediante la consola de Lambda

1. Inicie sesión en la Consola de administración de AWS y abra la consola de AWS Lambda en <https://console.aws.amazon.com/lambda/>.
2. Si todavía no ha creado una función de Lambda, consulte [Introducción \(p. 199\)](#).
3. En la página Functions , elija el nombre de la función y, a continuación, elija la pestaña Monitoring.



Se muestra una representación gráfica de las métricas de la función de Lambda.

4. Para ver los logs, elija View logs in CloudWatch.

Para obtener más información acerca de cómo tener acceso a CloudWatch Logs, consulte las siguientes guías:

- [Guía del usuario de Amazon CloudWatch](#)
- [Amazon CloudWatch Logs API Reference](#)
- [Monitorización de los archivos de log](#) en la Guía del usuario de Amazon CloudWatch

Métricas de AWS Lambda

En este tema se describen el espacio de nombres, las métricas y las dimensiones de AWS Lambda. AWS Lambda monitoriza automáticamente las funciones en su nombre, e informa sobre las métricas a través de Amazon CloudWatch. Estas métricas incluyen el total de invocaciones, los errores, la duración, las limitaciones controladas, los errores de DLQ (colas de mensajes fallidos) y la edad del iterador para las invocaciones basada en flujos.

CloudWatch es básicamente un repositorio de métricas. Una métrica es el concepto fundamental de CloudWatch y representa un conjunto de puntos de datos ordenados cronológicamente. El usuario (o los servicios de AWS) publican los puntos de datos de las métricas en CloudWatch y el usuario recupera estadísticas sobre esos puntos de datos en un conjunto ordenado de series temporales.

Las métricas se definen de forma exclusiva mediante un nombre, un espacio de nombres y una o varias dimensiones. Cada punto de datos tiene una marca temporal y, opcionalmente, una unidad de medida. Cuando se solicitan estadísticas, el flujo de datos devuelto se identifica mediante el espacio de nombres, el nombre de la métrica y la dimensión. Para obtener más información sobre CloudWatch, consulte la [Guía del usuario de Amazon CloudWatch](#).

Métricas de CloudWatch de AWS Lambda

El espacio de nombres de AWS Lambda incluye las siguientes métricas.

| Métrica | Descripción |
|--------------------------|--|
| Invocations | <p>Mide el número de veces que se invoca una función en respuesta a un evento o a una llamada a una API. Esta métrica sustituye a la métrica RequestCount obsoleta. Incluye las llamadas que se realizaron correcta e incorrectamente, pero no incluye los intentos limitados. Equivale a las solicitudes facturadas para la función. Tenga en cuenta que AWS Lambda solo envía estas métricas a CloudWatch si tienen un valor distinto de cero.</p> <p>Unidades: recuento</p> |
| Errors | <p>Mide el número de invocaciones que no se pudieron realizar debido a un error en la función (código de respuesta 4XX). Sustituye a la métrica ErrorCount obsoleta. Las invocaciones con error pueden activar un reinicio que prospere. Esto incluye:</p> <ul style="list-style-type: none"> • Excepciones controladas (por ejemplo, context.fail(error)) • Excepciones no controladas que hacen que termine el código • Excepciones de memoria insuficiente • Tiempos de espera • Errores de permisos <p>No incluye las invocaciones que produjeron error debido a que el número de invocaciones superó los límites simultáneos predeterminados (código de error 429) o debido a errores internos del servicio (código de error 500).</p> <p>Unidades: recuento</p> |
| Dead Letter Error | <p>Se incrementa cuando Lambda no puede escribir la carga del evento con error en las colas de mensajes fallidos configuradas. Esto podría deberse a lo siguiente:</p> <ul style="list-style-type: none"> • Errores de permisos • Limitaciones de servicios posteriores • Recursos configurados incorrectamente • Tiempos de espera <p>Unidades: recuento</p> |
| Duration | <p>Mide el tiempo de reloj que ha transcurrido desde que el código de la función empieza a ejecutarse como resultado de una invocación hasta que deja de ejecutarse. Sustituye a la métrica Latency obsoleta. El valor máximo del punto de datos posible es el valor de tiempo de espera de la función. La duración facturada se redondeará al valor superior más cercano a los 100</p> |

| Métrica | Descripción |
|--------------------|---|
| | <p>milisegundos. Tenga en cuenta que AWS Lambda solo envía estas métricas a CloudWatch si tienen un valor distinto de cero.</p> <p>Unidades: milisegundos</p> |
| Throttles | <p>Mide el número de intentos de invocación a la función Lambda que se limitaron debido a que el número de invocaciones superó los límites simultáneos del cliente (código de error 429). Las invocaciones con error pueden activar un reintento que prospere.</p> <p>Unidades: recuento</p> |
| IteratorAge | <p>Emitido solo para las invocaciones basadas en el flujo (funciones activadas por un flujo de Amazon DynamoDB o un flujo de Kinesis). Mide la antigüedad del último registro de cada lote de registros procesado. La antigüedad es la diferencia entre el momento en que Lambda recibió el lote y el momento en que el último registro del lote se escribió en el flujo.</p> <p>Unidades: milisegundos</p> |

Proporción de errores/invocaciones

Cuando se calcula el porcentaje de errores en invocaciones de funciones Lambda, es importante distinguir entre una solicitud de invocación y una invocación real. Es posible que el porcentaje de errores supere el número de invocaciones de funciones Lambda facturadas. Lambda registra una métrica de invocación solo si se ejecuta el código de la función Lambda. Si la solicitud de invocación produce una limitación u otro error de inicialización que impide llamar a la función Lambda, Lambda registrará un error, pero no una métrica de invocación.

- Lambda emite `Invocations=1` cuando se ejecuta la función. Si la función Lambda no se ejecuta, no se emite nada.
- Lambda emite un punto de datos de `Errors` para cada solicitud de invocación. `Errors=0` indica que no hay ningún error de ejecución de la función. `Errors=1` indica que hay un error de ejecución de la función.
- Lambda emite un punto de datos de `Throttles` para cada solicitud de invocación. `Throttles=0` indica que no hay ninguna limitación de invocación. `Throttles=1` indica que hay una limitación de invocación.

Dimensiones de CloudWatch de AWS Lambda

Puede utilizar las dimensiones de la siguiente tabla para ajustar las métricas devueltas para sus funciones Lambda.

| Dimensión | Descripción |
|-------------------------------|--|
| <code>FunctionName</code> | Filtrá los datos de métricas por función Lambda. |
| <code>Resource</code> | Filtrá los datos de las métricas del recurso de la función Lambda, como, por ejemplo, la versión o el alias de la función. |
| <code>Executed Version</code> | Filtrá los datos de métricas por versiones de función Lambda. Esto solo se aplica a las invocaciones de alias. |

Colas de mensajes fallidos

De forma predeterminada, una función de Lambda invocada de forma asíncrona que genera errores se reintenta dos veces y, a continuación, el evento se descarta. En lugar de esto, puede utilizar colas de mensajes fallidos (DLQ) para indicar a Lambda que los eventos sin procesar deben enviarse a una cola de Amazon SQS o a un tema de Amazon SNS, donde se pueden realizar otras acciones.

Puede configurar una DLQ especificando un nombre de recurso de Amazon (ARN) en el parámetro `DeadLetterConfig` de una función de Lambda del tema de Amazon SNS o de la cola de Amazon SQS donde desea que se entregue la carga del evento, tal como se muestra en el siguiente código. Para obtener más información acerca de la creación de un tema de Amazon SNS, consulte [Creación de un tema de SNS](#). Para obtener más información acerca de cómo crear una cola de Amazon SQS, consulte [Tutorial: Creación de una cola de Amazon SQS](#).

```
{  
    "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 dirige los eventos que no se pueden procesar al tema de Amazon SNS o a la cola de Amazon SQS que se ha configurado para la función de Lambda. Las funciones que no tienen asociada una DLQ descartan los eventos tras agotar sus intentos. Para obtener más información acerca de las políticas de reintentos, consulte [Reintentos si se producen errores \(p. 192\)](#). Debe proporcionar de forma explícita el acceso read/publish/sendMessage al recurso de DLQ como parte del rol de ejecución para la función de Lambda. La carga que se escribe en el ARN de destino de la DLQ es la carga del evento original sin modificaciones en el cuerpo del mensaje. Los atributos del mensaje, que se describen a continuación, contienen información que le ayudará a entender por qué no se ha procesado el evento:

| Nombre | Tipo | Valor |
|--------------|--------|-------------------------------------|
| RequestID | Cadena | Identificador único de la solicitud |
| ErrorCode | Número | Código de error HTTP de 3 dígitos |
| ErrorMessage | Cadena | Mensaje de error (truncado a 1 KB) |

Si por alguna razón, la carga del evento no consigue llegar al ARN de destino, Lambda incrementa una métrica de CloudWatch denominada `DeadLetterErrors` y, a continuación, elimina la carga del evento.



Creación de aplicaciones con AWS Lambda

Cuando se crean aplicaciones en AWS Lambda, incluidas las aplicaciones [sin servidor](#), los componentes principales son las funciones de Lambda y los orígenes de eventos. Un origen de eventos es el servicio de AWS o la aplicación personalizada que publica los eventos, y una función de Lambda es el código personalizado que los procesa. A modo de ejemplo, considere las siguientes situaciones:

- Procesamiento de archivos: supongamos que tiene una aplicación de uso compartido de fotografías. Los usuarios utilizan la aplicación para cargar fotografías, y esta las almacena en un bucket de Amazon S3. A continuación, la aplicación crea una versión en miniatura las fotografías de cada usuario y las muestra en la página de perfil del usuario. En este caso, puede optar por crear una función de Lambda para crear una miniatura automáticamente. Amazon S3 es uno de los orígenes de eventos de AWS que pueden publicar eventos de creación de objetos e invocar una función de Lambda. El código de la función de Lambda puede leer el objeto de fotografía del bucket de S3, crear una versión en miniatura y guardarla en otro bucket de S3.
- Datos y análisis: supongamos que está creando una aplicación de análisis, y que almacena los datos sin procesar en una tabla de DynamoDB. Cuando se escriben, actualizan o eliminan elementos de una tabla, los flujos de DynamoDB pueden publicar eventos de actualización de elementos en un flujo asociado a la tabla. En este caso, los datos del evento proporcionan la clave del elemento, el nombre del evento (por ejemplo, insert, update y delete) y otros detalles. Puede escribir una función de Lambda que genere métricas personalizadas agregando los datos sin procesar.
- Sitios web: supongamos que está creando un sitio web y que desea alojar la lógica del backend en Lambda. Puede invocar la función de Lambda a través de HTTP utilizando Amazon API Gateway como punto de enlace HTTP. A continuación, el cliente web puede invocar la API y, por último, API Gateway puede dirigir la solicitud a Lambda.
- Aplicaciones móviles: supongamos que tiene una aplicación móvil personalizada que produce eventos. Puede crear una función de Lambda para procesar los eventos publicados por la aplicación personalizada. Por ejemplo, en este caso, puede configurar una función de Lambda para procesar los clics que se realizan en la aplicación móvil personalizada.

Cada uno de estos orígenes de eventos utiliza un formato específico para los datos de los eventos. Para obtener más información, consulte [Eventos de muestra publicados por los orígenes de eventos \(p. 146\)](#). Cuando se invoca una función de Lambda, esta recibe el evento como parámetro de la función de Lambda.

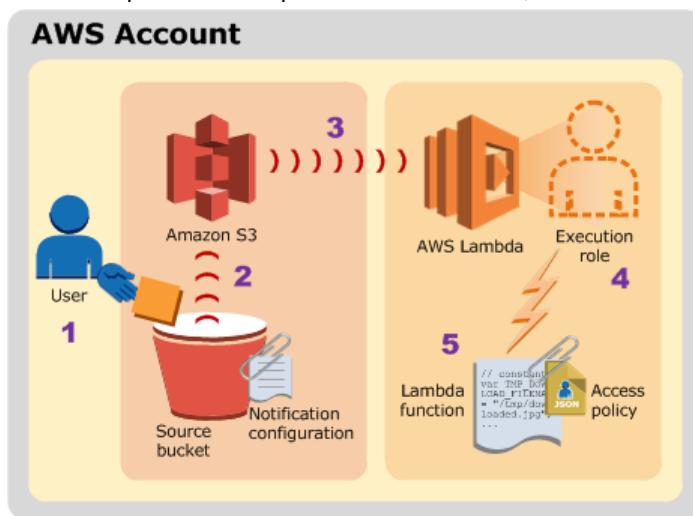
AWS Lambda admite muchos servicios de AWS como orígenes de eventos. Para obtener más información, consulte [Orígenes de eventos admitidos \(p. 138\)](#). Cuando se configuran estos orígenes de eventos para activar una función de Lambda, la función de Lambda se invoca automáticamente cuando se producen los eventos. Debe definir el mapeo de orígenes de eventos, que es la forma de identificar los eventos cuyo seguimiento se desea realizar y la función de Lambda que se debe invocar.

Además de los servicios de AWS admitidos, las aplicaciones de los usuarios también pueden generar eventos, lo que les permite crear sus propios orígenes de eventos personalizados. Los orígenes de eventos personalizados invocan una función de Lambda utilizando la operación [Invoke \(p. 423\)](#) de AWS Lambda. Las aplicaciones de los usuarios, como las aplicaciones cliente, móviles o web, pueden publicar eventos e invocar funciones de Lambda bajo demanda mediante los SDK de AWS o los AWS Mobile SDK, como el SDK de AWS Mobile para Android.

A continuación se muestran ejemplos introductorios de orígenes de eventos y del funcionamiento de la experiencia integral.

Ejemplo 1: Amazon S3 envía eventos e invoca una función de Lambda

Amazon S3 puede publicar eventos de diferentes tipos, como, por ejemplo, los eventos de objetos PUT, POST, COPY y DELETE en un bucket. Mediante la característica de notificaciones de los buckets, puede configurar un mapeo de origen de eventos que indique a Amazon S3 que invoque una función de Lambda cuando se produzca un tipo concreto de evento, tal como se muestra en la siguiente ilustración.



El diagrama ilustra la siguiente secuencia:

1. El usuario crea un objeto en un bucket.
2. Amazon S3 detecta el evento de creación de objeto.
3. Amazon S3 invoca la función de Lambda utilizando los permisos proporcionados por el rol de ejecución. Para obtener más información acerca de los roles de ejecución, consulte [Autenticación y control de acceso de AWS Lambda \(p. 337\)](#). Amazon S3 sabe cuál es la función de Lambda que debe invocar basándose en el mapeo de origen de eventos almacenado en la configuración de notificaciones del bucket.
4. AWS Lambda ejecuta la función de Lambda, especificando el evento como parámetro.

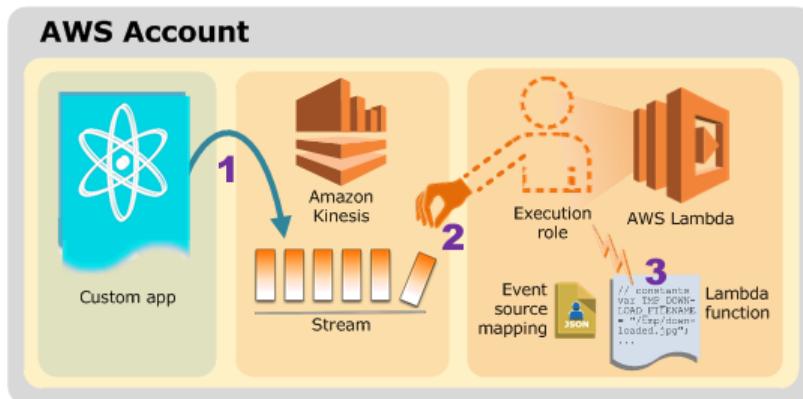
Tenga en cuenta lo siguiente:

- El mapeo de origen de eventos se mantiene en el servicio del origen de eventos, en este caso, Amazon S3. Esto se aplica a todos los orígenes de eventos compatibles de AWS, excepto los orígenes basados en flujos (flujos de Kinesis y DynamoDB). En el siguiente ejemplo se explican los orígenes de eventos basados en flujos.
- El origen de eventos (Amazon S3) invoca la función de Lambda (esto se denomina modelo de inserción). De nuevo, esto se aplica a todos los servicios compatibles de AWS, excepto los orígenes de eventos basados en flujos.
- Para que el origen de eventos (Amazon S3) pueda invocar la función de Lambda, debe conceder permisos a través de la política de permisos asociada a la función de Lambda.

Ejemplo 2: AWS Lambda obtiene los eventos desde un flujo de Kinesis e invoca una función de Lambda

Para los orígenes de eventos basados en flujos, AWS Lambda sondea el flujo e invoca la función de Lambda cuando se detectan registros en el flujo. Estos orígenes de flujos son especiales porque la información de mapeo de orígenes de eventos se almacena en Lambda. AWS Lambda proporciona una API para crear y administrar estos mapeos de orígenes de eventos.

En el siguiente diagrama se muestra cómo una aplicación personalizada escribe registros en un flujo de Kinesis.



El diagrama ilustra la siguiente secuencia:

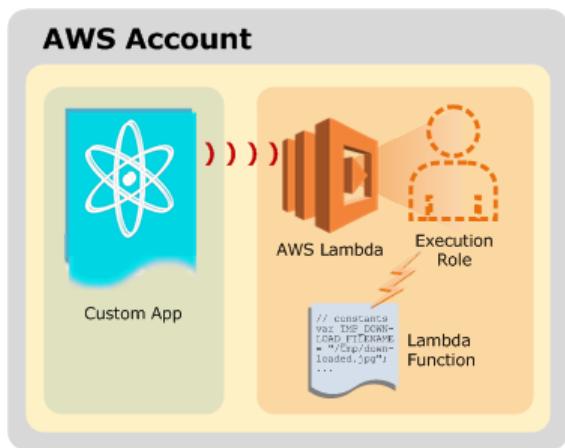
1. La aplicación personalizada escribe registros en un flujo de Kinesis.
2. AWS Lambda sondea constantemente el flujo, e invoca la función de Lambda cuando el servicio detecta registros nuevos en el flujo. AWS Lambda sabe cuál es el flujo que debe sondear y la función de Lambda que debe invocar basándose en el mapeo de origen de eventos que se ha creado en Lambda.
3. La función de Lambda se invoca con el evento entrante.

Tenga en cuenta lo siguiente:

- Cuando se trabaja con orígenes de eventos basados en flujos, se cumple lo siguiente:
 - Los mapeos de orígenes de eventos se crean en AWS Lambda.
 - AWS Lambda invoca la función de Lambda (esto se denomina modelo de extracción).
- AWS Lambda no necesita permiso para invocar la función de Lambda; por lo tanto, no es necesario añadir permisos a la política de permisos asociada a la función de Lambda.
- El rol de Lambda necesita permiso para leer los datos del flujo.

Ejemplo 3: Una aplicación personalizada publica eventos e invoca una función de Lambda

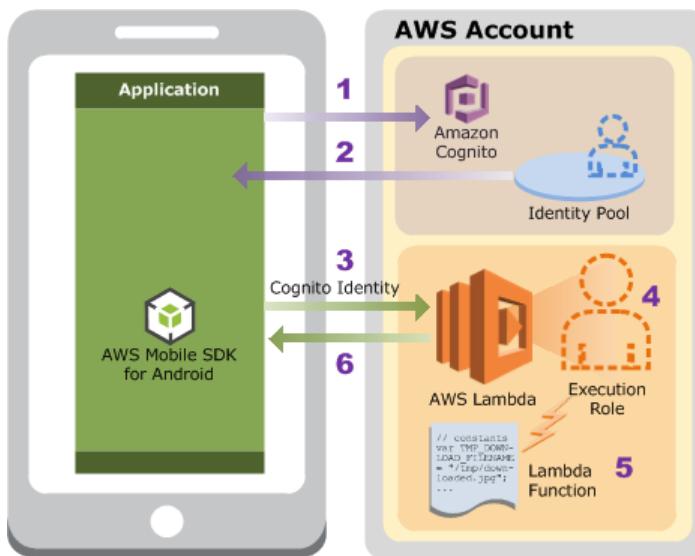
En el siguiente diagrama se muestra cómo una aplicación personalizada de una cuenta invoca una función de Lambda.



El diagrama ilustra la siguiente secuencia:

1. La aplicación personalizada invoca la función de Lambda mediante el SDK de AWS.
2. La función de Lambda se invoca con el evento entrante.

En el siguiente diagrama se muestra cómo una aplicación móvil personalizada invoca una función de Lambda.



1. La aplicación móvil envía una solicitud a Amazon Cognito con un ID del grupo de identidades en la solicitud (el grupo de identidades se crea durante la configuración).
2. Amazon Cognito devuelve unas credenciales de seguridad temporales a la aplicación.

Amazon Cognito asume el rol asociado al grupo de identidades para generar las credenciales temporales.

3. La aplicación móvil invoca la función de Lambda utilizando las credenciales temporales (identidad de Cognito).
4. AWS Lambda asume el rol de ejecución para ejecutar la función de Lambda en su nombre.
5. La función de Lambda se ejecuta.

6. AWS Lambda devuelve los resultados a la aplicación móvil, suponiendo que la aplicación invocó la función de Lambda utilizando el tipo de invocación RequestResponse (esto se denomina invocación síncrona).

Tenga en cuenta lo siguiente:

- Este es un ejemplo de una invocación bajo demanda de una función de Lambda. Para las invocaciones bajo demanda, no es necesario preconfigurar un mapeo de origen de eventos como se hace para los servicios de AWS.
- En este ejemplo, como la aplicación personalizada utiliza las mismas credenciales de cuenta que la cuenta a la que pertenece la función de Lambda, no necesita permisos adicionales para invocar la función.

Lectura recomendada

Si es la primera vez que utiliza AWS Lambda, le recomendamos que lea todos los temas de esta sección para obtener más información. También conviene examinar primero el ejercicio [Introducción \(p. 199\)](#) para obtener experiencia práctica en la creación y prueba de una función de Lambda y, a continuación, leer los temas de este capítulo.

Además, el tema [Creación de funciones de Lambda \(p. 5\)](#) también proporciona información introductoria que podría resultarle de utilidad antes de profundizar en la tecnología.

Mapeo de orígenes de eventos

Las funciones de Lambda y los orígenes de eventos son los componentes fundamentales de AWS Lambda. Para obtener más información, consulte [AWS Lambda: funcionamiento \(p. 188\)](#). Un origen de eventos es la entidad que publica los eventos, y una función de Lambda es el código personalizado que los procesa. Se conocen como orígenes de eventos compatibles los servicios de AWS que se pueden preconfigurar para que funcionen con AWS Lambda. La configuración se conoce como mapeo de origen de eventos, que mapea un origen de eventos a una función de Lambda. Permite invocar automáticamente la función de Lambda cuando se producen eventos.

Cada mapeo de origen de eventos identifica el tipo de eventos que se publican y la función de Lambda que se debe invocar cuando se producen los eventos. A continuación, la función específica de Lambda recibe la información del evento como parámetro, y el código de la función de Lambda puede procesar el evento.

Tenga en cuenta lo siguiente acerca de los orígenes de eventos. Los orígenes de eventos pueden ser cualquiera de los siguientes:

- Servicios de AWS: son los servicios compatibles de AWS que pueden preconfigurarse para que funcionen con AWS Lambda. Puede agrupar estos servicios como servicios normales de AWS o servicios basados en flujos. Amazon Kinesis Data Streams y Amazon DynamoDB Streams son orígenes de eventos basados en flujos; ninguno de los demás servicios de AWS utiliza orígenes de eventos basados en flujos. El lugar donde se mantiene el mapeo de origen de eventos y la forma en que se invoca la función de Lambda dependen de si se está utilizando o no un origen de eventos basado en flujos.
- Aplicaciones personalizadas: puede hacer que sus aplicaciones personalizadas publiquen eventos e invoquen una función de Lambda.

Es posible que se pregunte dónde se guarda la información de mapeo de eventos. Si se guarda en el origen de eventos o en AWS Lambda. En las secciones siguientes se explica el mapeo de orígenes de eventos para cada una de estas categorías de orígenes de eventos. En estas secciones también se explica

cómo se invoca la función de Lambda y cómo se administran los permisos para permitir la invocación de la función de Lambda.

Temas

- [Mapeo de orígenes de eventos para los servicios de AWS \(p. 135\)](#)
- [Mapeo de orígenes de eventos para los servicios de AWS basados en flujos \(p. 136\)](#)
- [Mapeo de orígenes de eventos para las aplicaciones personalizadas \(p. 137\)](#)

Mapeo de orígenes de eventos para los servicios de AWS

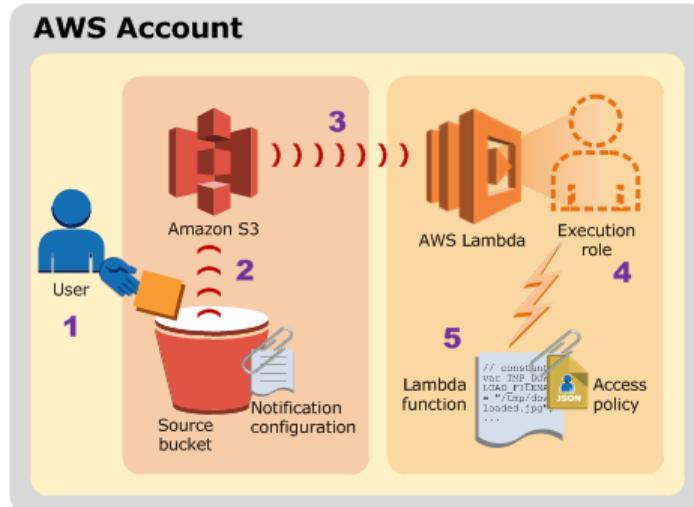
Salvo los servicios de AWS basados en flujos (Amazon Kinesis Data Streams y DynamoDB), los demás servicios compatibles de AWS publican eventos y también pueden invocar una función de Lambda (esto se denomina modelo de inserción). En el modelo de inserción, tenga en cuenta lo siguiente:

- Los mapeos del origen de eventos se mantienen en el origen de eventos. El uso de las API adecuadas en los orígenes de eventos permite crear y administrar mapeos de orígenes de eventos. Por ejemplo, Amazon S3 proporciona la API de configuración de notificaciones de bucket. Gracias a esta API, puede configurar un mapeo de origen de eventos que identifique los eventos de bucket que se deben publicar y la función de Lambda que se debe invocar.
- Dado que los orígenes de eventos invocan la función de Lambda, debe conceder al origen de eventos los permisos necesarios mediante una política basada en recursos (esto se denomina política de función de Lambda). Para obtener más información, consulte [Modelo de permisos de AWS Lambda \(p. 193\)](#).

En el siguiente ejemplo se ilustra el funcionamiento de este modelo.

Example Amazon S3 envía los eventos e invoca una función de Lambda

Supongamos que desea que se invoque una función de AWS Lambda para cada evento de bucket object-created. Debe añadir el mapeo de origen de eventos necesario a la configuración de notificaciones del bucket.



El diagrama ilustra el flujo:

1. El usuario crea un objeto en un bucket.
2. Amazon S3 detecta el evento de creación de objeto.

3. Amazon S3 invoca la función de Lambda correspondiente al mapeo de origen de eventos que se describe en la configuración de notificaciones del bucket.
4. AWS Lambda verifica la política de permisos asociada a la función de Lambda para asegurarse de que Amazon S3 tiene los permisos necesarios. Para obtener más información acerca de las políticas de permisos, consulte [Autenticación y control de acceso de AWS Lambda \(p. 337\)](#).
5. Una vez que AWS Lambda verifica la política de permisos asociada, ejecuta la función de Lambda. Recuerde que la función de Lambda recibe el evento como parámetro.

Mapeo de orígenes de eventos para los servicios de AWS basados en flujos

Los flujos de Amazon Kinesis Data Streams y DynamoDB son servicios basados en flujos que pueden preconfigurarse para su uso con AWS Lambda. Cuando se ha hecho el mapeo de origen de eventos necesario, AWS Lambda sondea los flujos e invoca la función de Lambda (esto se denomina modelo de extracción). En el modelo de extracción, tenga en cuenta lo siguiente:

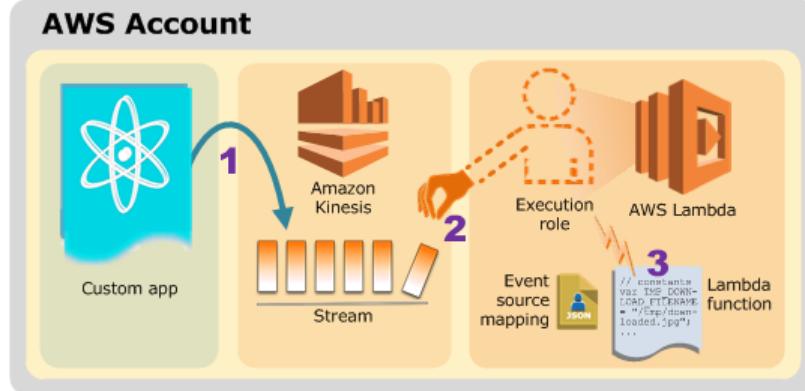
- Los mapeos del origen de eventos se mantienen en AWS Lambda. AWS Lambda proporciona las API necesarias para crear y administrar mapeos de orígenes de eventos. Para obtener más información, consulte [CreateEventSourceMapping \(p. 380\)](#).
- Debe conceder permiso a AWS Lambda para sondear el flujo y leer los registros. Puede conceder estos permisos a través del rol de ejecución, utilizando la política de permisos asociada al rol que especifique al crear la función de Lambda. AWS Lambda no necesita ningún permiso para invocar la función de Lambda.

En el siguiente ejemplo se ilustra el funcionamiento de este modelo.

Example AWS Lambda obtiene los eventos desde un flujo de Kinesis e invoca una función de Lambda

En el siguiente diagrama se muestra una aplicación personalizada que escribe registros en un flujo de Kinesis y cómo AWS Lambda sondea el flujo. Cuando AWS Lambda detecta un registro nuevo en el flujo, invoca la función de Lambda.

Suponga que tiene una aplicación personalizada que escribe registros en un flujo de Kinesis. Desea invocar una función de Lambda cuando se detecten registros nuevos en el flujo. Debe crear una función de Lambda y el mapeo de origen de eventos necesario en AWS Lambda.



El diagrama ilustra la siguiente secuencia:

1. La aplicación personalizada escribe registros en un flujo de Kinesis.

2. AWS Lambda sondea constantemente el flujo, e invoca la función de Lambda cuando el servicio detecta registros nuevos en el flujo. AWS Lambda sabe cuál es el flujo que debe sondear y la función de Lambda que debe invocar basándose en el mapeo de origen de eventos que se ha creado en AWS Lambda.
3. Suponiendo que la política de permisos asociada, que permite a AWS Lambda sondear el flujo, está verificada, AWS Lambda ejecuta la función de Lambda. Para obtener más información acerca de las políticas de permisos, consulte [Autenticación y control de acceso de AWS Lambda \(p. 337\)](#).

El ejemplo utiliza un flujo de Kinesis, pero también se aplica cuando se utiliza un flujo de DynamoDB.

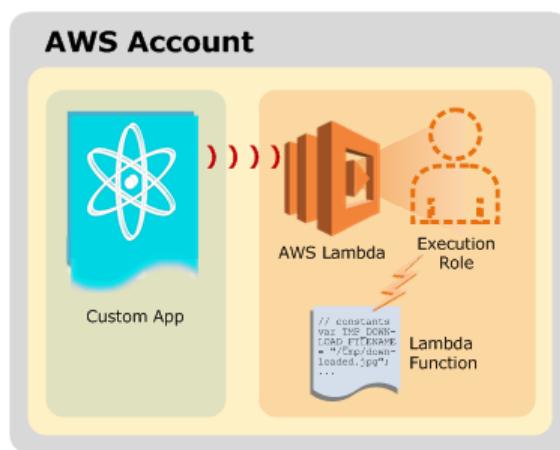
Mapeo de orígenes de eventos para las aplicaciones personalizadas

Si tiene aplicaciones personalizadas que publican y procesan eventos, puede crear una función de Lambda para procesar estos eventos. En este caso, no se necesita ninguna preconfiguración (no es necesario configurar un mapeo de origen de eventos). En lugar de ello, el origen de eventos utiliza la API `Invoke` de AWS Lambda. Si la aplicación y la función de Lambda pertenecen a cuentas de AWS distintas, la cuenta de AWS a la que pertenece la función de Lambda debe incluir permisos entre cuentas en la política de permisos asociada a la función de Lambda.

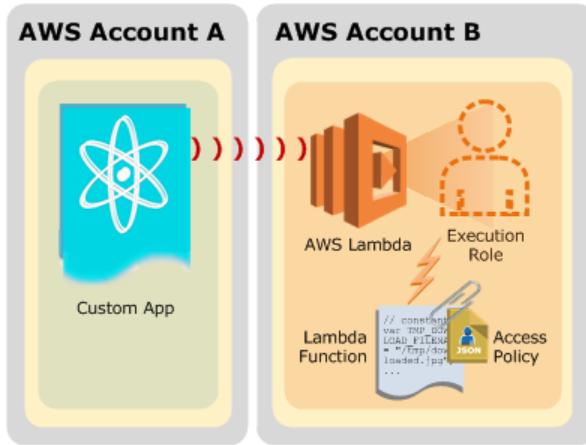
El siguiente ejemplo ilustra cómo funciona.

Example Una aplicación personalizada publica eventos e invoca una función de Lambda

En el siguiente diagrama se muestra cómo una aplicación personalizada de una cuenta puede invocar una función de Lambda. En este ejemplo, la aplicación personalizada utiliza las mismas credenciales de cuenta que la cuenta a la que pertenece la función de Lambda, por lo que no necesita permisos adicionales para invocar la función.



En el siguiente ejemplo, la aplicación del usuario y la función de Lambda pertenecen a cuentas de AWS distintas. En este caso, la cuenta de AWS a la que pertenece la función de Lambda debe tener permisos entre cuentas en la política de permisos asociada a la función de Lambda. Para obtener más información, consulte [Modelo de permisos de AWS Lambda \(p. 193\)](#).



Lectura recomendada

Si es la primera vez que utiliza AWS Lambda, le sugerimos que lea todos los temas de la sección “Funcionamiento” para familiarizarse con Lambda. El siguiente tema es [Orígenes de eventos admitidos \(p. 138\)](#).

Después de leer todos los temas de la sección Funcionamiento, le recomendamos que lea [Creación de funciones de Lambda \(p. 5\)](#), que pruebe el ejercicio de [Introducción \(p. 199\)](#) y, a continuación, que examine los [Casos de uso \(p. 214\)](#). Cada caso de uso proporciona instrucciones paso a paso para configurar la experiencia integral.

Orígenes de eventos admitidos

En este tema se indican los servicios compatibles de AWS que se pueden configurar como orígenes de eventos para las funciones de AWS Lambda. Después de preconfigurar el mapeo de origen de eventos, la función de Lambda se invoca automáticamente cuando estos orígenes de eventos detectan eventos. Para obtener más información acerca de los modos de invocación, consulte [Mapeo de orígenes de eventos \(p. 134\)](#).

Tenga en cuenta lo siguiente para todos los orígenes de eventos que se indican en este tema:

- El mapeo de origen de eventos se guarda en los orígenes de eventos, salvo para los servicios basados en flujos (Amazon Kinesis Data Streams y Amazon DynamoDB Streams). Para los servicios basados en flujos, AWS Lambda guarda el mapeo de origen de eventos. AWS Lambda proporciona la operación [CreateEventSourceMapping \(p. 380\)](#) para crear y administrar el mapeo de origen de eventos. Para obtener más información, consulte [Mapeo de orígenes de eventos \(p. 134\)](#).
- El tipo de invocación que utilizan estos orígenes de eventos cuando se invoca una función de Lambda también está preconfigurado. Por ejemplo, Amazon S3 siempre invoca una función de Lambda forma asíncrona y Amazon Cognito invoca una función de Lambda de forma síncrona. La única forma de controlar el tipo de invocación es cuando se invoca una función de Lambda utilizando la operación [Invoke \(p. 423\)](#) (por ejemplo, si se invoca una función de Lambda bajo demanda desde una aplicación personalizada).
- Para procesar eventos de AWS, es posible que sea necesario incluir bibliotecas adicionales en las funciones de Lambda, dependiendo del lenguaje de programación utilizado para crearlas. Las funciones

escritas en Node.js o Python no requieren bibliotecas adicionales. Para C#, es necesario incluir [AWS Lambda para .NET Core](#). Para Java, es necesario incluir [aws-lambda-java-libs](#).

Important

Cada uno de los paquetes incluidos debe utilizarse sin modificaciones. Si se eliminan dependencias, se añaden dependencias conflictivas o se incluyen clases de los paquetes de forma selectiva, puede producirse un comportamiento inesperado.

También puede invocar una función de Lambda bajo demanda. Para obtener más información, consulte [Otros orígenes de eventos: invocación de una función de Lambda bajo demanda \(p. 146\)](#).

Para ver ejemplos de los eventos que publican estos orígenes de eventos, consulte [Eventos de muestra publicados por los orígenes de eventos \(p. 146\)](#).

Temas

- [Amazon S3 \(p. 139\)](#)
- [Amazon DynamoDB \(p. 140\)](#)
- [Amazon Kinesis Data Streams \(p. 140\)](#)
- [Amazon Simple Notification Service \(p. 140\)](#)
- [Amazon Simple Email Service \(p. 141\)](#)
- [Amazon Cognito \(p. 141\)](#)
- [AWS CloudFormation \(p. 142\)](#)
- [Amazon CloudWatch Logs \(p. 142\)](#)
- [Amazon CloudWatch Events \(p. 142\)](#)
- [AWS CodeCommit \(p. 143\)](#)
- [Eventos programados \(basados en Amazon CloudWatch Events\) \(p. 143\)](#)
- [AWS Config \(p. 143\)](#)
- [Amazon Alexa \(p. 144\)](#)
- [Amazon Lex \(p. 144\)](#)
- [Amazon API Gateway \(p. 144\)](#)
- [Botón de AWS IoT \(p. 145\)](#)
- [Amazon CloudFront \(p. 145\)](#)
- [Amazon Kinesis Data Firehose \(p. 145\)](#)
- [Otros orígenes de eventos: invocación de una función de Lambda bajo demanda \(p. 146\)](#)
- [Eventos de muestra publicados por los orígenes de eventos \(p. 146\)](#)

Amazon S3

Puede escribir funciones de Lambda para procesar eventos de bucket de S3, como por ejemplo, eventos ObjectCreated u ObjectDeleted. Por ejemplo, cuando un usuario cargue una fotografía en un bucket de Amazon S3, es posible que desee invocar una función de Lambda que lea la imagen y cree una miniatura de la fotografía.

Puede utilizar la característica de configuración de notificaciones de bucket de Amazon S3 para configurar el mapeo de origen de eventos, identificar los eventos del bucket que desea que publique Amazon S3 y la función de Lambda que se debe invocar.

Para ver un ejemplo de evento de Amazon S3, consulte [Estructura de mensaje de evento, Evento de muestra Put de Amazon S3 \(p. 151\)](#) y [Evento de muestra Delete de Amazon S3 \(p. 152\)](#). Para ver un ejemplo de caso de uso, consulte [Uso de AWS Lambda con Amazon S3 \(p. 214\)](#).

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. Amazon S3 invoca la función de Lambda de forma asíncrona. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Amazon DynamoDB

Puede utilizar funciones de Lambda como disparadores para una tabla de Amazon DynamoDB. Los disparadores son acciones personalizadas que se llevan a cabo en respuesta a actualizaciones realizadas en la tabla de DynamoDB. Para crear un disparador, primero debe habilitar Amazon DynamoDB Streams para la tabla. AWS Lambda sondea el flujo y la función de Lambda procesa las actualizaciones publicadas en él.

Este es un origen de eventos basado en flujos. Para un servicio basado en flujos, debe crear el mapeo de origen de eventos en AWS Lambda e identificar el flujo que se debe sondear y la función de Lambda que se debe invocar.

Para ver un ejemplo de evento de DynamoDB, consulte [Paso 2.3.2: Prueba de la función de Lambda \(invocación manual\) \(p. 249\)](#) y [Evento de muestra de actualización de Amazon DynamoDB \(p. 149\)](#).

Para ver el formato general, consulte [GetRecord](#) en la Amazon DynamoDB API Reference. Para ver un ejemplo de caso de uso, consulte [Uso de AWS Lambda con Amazon DynamoDB \(p. 243\)](#).

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. DynamoDB es un origen de eventos basado en flujos. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Amazon Kinesis Data Streams

Puede configurar AWS Lambda para que sondee automáticamente un flujo y procese los registros nuevos, como secuencias de clics de sitios web, transacciones financieras, fuentes de redes sociales, logs de TI y eventos de seguimiento de ubicación. A continuación, AWS Lambda sondea el flujo de forma periódica (una vez por segundo) buscando registros nuevos.

Para un servicio basado en flujos, debe crear el mapeo de origen de eventos en AWS Lambda e identificar el flujo que se debe sondear y la función de Lambda que se debe invocar.

Para ver un ejemplo de evento, consulte [Paso 2.3: Creación y prueba manual de la función de Lambda \(p. 238\)](#) y [Evento de muestra de Amazon Kinesis Data Streams \(p. 151\)](#). Para ver un ejemplo de caso de uso, consulte [Uso de AWS Lambda con Kinesis \(p. 232\)](#).

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. Amazon Kinesis Data Streams es un origen de eventos basado en flujos. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Amazon Simple Notification Service

Puede escribir funciones de Lambda para procesar通知 de Amazon Simple Notification Service. Cuando se publique un mensaje en un tema de Amazon SNS, el servicio puede invocar la función de Lambda pasando la carga del mensaje como parámetro. El código de la función de Lambda puede procesar el evento, por ejemplo, publicar el mensaje en otros temas de Amazon SNS o enviarlo a otros servicios de AWS.

Esto también le permite activar una función de Lambda en respuesta a las alarmas de Amazon CloudWatch y otros servicios de AWS que utilizan Amazon SNS.

El mapeo de orígenes de eventos de Amazon SNS se configura a través de la configuración de suscripciones del tema. Para obtener más información, consulte [Invoking Lambda functions using Amazon SNS notifications](#) en la Guía para desarrolladores de Amazon Simple Notification Service.

Para ver un evento de ejemplo, consulte [Appendix: Message and JSON Formats](#) y [Evento de muestra de Amazon SNS \(p. 148\)](#). Para ver un ejemplo de caso de uso, consulte [Uso de AWS Lambda con Amazon SNS desde cuentas distintas \(p. 270\)](#).

Cuando un usuario llame a la API Publish de SNS en un tema al que está suscrita una función de Lambda, Amazon SNS llamará a Lambda para invocar la función de forma asíncrona. A continuación, Lambda devolverá un estado de entrega. Si se produce un error al llamar a Lambda, Amazon SNS intentará invocar la función de Lambda hasta tres veces. Después de tres intentos, si Amazon SNS sigue sin invocar correctamente la función de Lambda, Amazon SNS enviará a CloudWatch un mensaje de error de estado de entrega.

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. Amazon SNS invoca la función de Lambda de forma asíncrona. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Amazon Simple Email Service

Amazon Simple Email Service (Amazon SES) es un servicio de envío de correo electrónico rentable. Con Amazon SES, además de enviar correos electrónicos, también puede utilizar el servicio para recibir mensajes. Para obtener más información acerca de Amazon SES, consulte [Amazon Simple Email Service](#). Si utiliza Amazon SES para recibir mensajes, puede configurar Amazon SES para que llame a una función de Lambda cuando lleguen mensajes. El servicio puede invocar la función de Lambda pasándole como parámetro el evento de correo electrónico entrante, que en realidad es un mensaje de Amazon SES en un evento de Amazon SNS. Para ver escenarios de ejemplo, consulte [Considering Your Use Case for Amazon SES Email Receiving](#).

El mapeo de orígenes de eventos de Amazon SES se configura mediante la configuración de reglas. Los siguientes temas proporcionan información adicional en la Guía para desarrolladores de Amazon Simple Email Service:

- Para ver eventos de ejemplo, consulte [Lambda Action](#) y [Evento de muestra de recepción de correo electrónico de Amazon SES \(p. 147\)](#).
- Para ver ejemplos de funciones de Lambda, consulte [Lambda Function Examples](#).

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. Amazon SES invoca la función de Lambda de forma asíncrona. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Amazon Cognito

La característica de eventos de Amazon Cognito permite ejecutar una función de Lambda como respuesta a eventos de Amazon Cognito. Por ejemplo, puede invocar una función de Lambda para los eventos Sync Trigger, que se publique cada vez que se sincronice un conjunto de datos. Para obtener más información y ver un ejemplo, consulte [Introducing Amazon Cognito Events: Sync Triggers](#) en el blog de desarrollo para móviles.

El mapeo de orígenes de eventos de Amazon Cognito se configura mediante la configuración de suscripciones de eventos. Para obtener información sobre el mapeo de orígenes de eventos y un evento de muestra, consulte [Eventos de Amazon Cognito](#) en la Guía para desarrolladores de Amazon Cognito. Para ver otro ejemplo de evento, consulte [Evento de muestra de disparador de sincronización de Amazon Cognito \(p. 150\)](#).

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. Amazon Cognito está configurado para invocar una función de Lambda de forma síncrona. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

AWS CloudFormation

Como parte de la implementación de pilas de AWS CloudFormation, puede especificar una función de Lambda como recurso personalizado para ejecutar cualquier comando personalizado. Si asocia una función de Lambda a un recurso personalizado, podrá invocar la función de Lambda siempre que cree, actualice o elimine pilas de AWS CloudFormation.

El mapeo de orígenes de eventos de AWS CloudFormation se configura mediante la definición de pilas. Para obtener más información, consulte [Recursos personalizados respaldados por AWS Lambda](#) en la Guía del usuario de AWS CloudFormation.

Para ver un ejemplo de evento, consulte [Evento de muestra de creación de solicitud de AWS CloudFormation \(p. 147\)](#). Tenga en cuenta que este evento es realmente un mensaje de AWS CloudFormation en un evento de Amazon SNS.

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. AWS CloudFormation invoca la función de Lambda de forma síncrona. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Amazon CloudWatch Logs

Puede utilizar funciones de AWS Lambda para realizar análisis personalizados en Amazon CloudWatch Logs utilizando suscripciones de CloudWatch Logs. Las suscripciones de CloudWatch Logs proporcionan acceso a una fuente en tiempo real de eventos de log de CloudWatch Logs y se la envían a la función de AWS Lambda para su procesamiento personalizado, análisis o carga en otros sistemas. Para obtener más información sobre CloudWatch Logs, consulte [Monitorización de archivos de log](#).

El mapeo de origen de eventos se mantiene en Amazon CloudWatch Logs mediante la configuración de suscripciones de log. Para obtener más información, consulte [Procesamiento en tiempo real de datos de registros con suscripciones \(Ejemplo 2: AWS Lambda\)](#) en la Guía del usuario de Amazon CloudWatch.

Para ver un ejemplo de evento, consulte [Evento de muestra de Amazon CloudWatch Logs \(p. 148\)](#).

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. Amazon CloudWatch Logs invoca la función de Lambda de forma asíncrona (al invocar una función de Lambda, no se bloquea la operación de escritura en los logs). Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Amazon CloudWatch Events

[Amazon CloudWatch Events](#) le ayuda a responder a los cambios de estado de los recursos de AWS. Cuando cambia el estado de los recursos, estos envían automáticamente eventos a un flujo de eventos. Puede crear reglas que coincidan con los eventos seleccionados en el flujo y los transfieran a una función de AWS Lambda para realizar alguna acción. Por ejemplo, puede invocar automáticamente una función de AWS Lambda para registrar el estado de una [instancia EC2](#) o un [grupo de Auto Scaling](#).

El mapeo de origen de eventos se mantiene en Amazon CloudWatch Events mediante la utilización de una definición de destino de regla. Para obtener más información, consulte la operación [PutTargets](#) en la Referencia de la API de Amazon CloudWatch Events.

Para ver eventos de muestra, consulte [Tipos de eventos compatibles](#) en la Guía del usuario de Amazon CloudWatch.

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. Amazon CloudWatch Events invoca la función de Lambda de forma asíncrona. Para obtener

más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

AWS CodeCommit

Puede crear un disparador para un repositorio de AWS CodeCommit de modo que los eventos del repositorio invoquen una función de Lambda. Por ejemplo, puede invocar una función de Lambda cuando se crea una ramificación o una etiqueta, o cuando se envían datos a una ramificación existente. Para obtener más información, consulte [Administración de los disparadores de un repositorio de AWS CodeCommit](#).

El mapeo de origen de eventos se mantiene en AWS CodeCommit mediante un disparador de repositorio. Para obtener más información, consulte la operación [PutRepositoryTriggers](#).

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. AWS CodeCommit invoca la función de Lambda de forma asíncrona. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Eventos programados (basados en Amazon CloudWatch Events)

También puede configurar AWS Lambda para que invoque el código de forma periódica y programada utilizando la capacidad de eventos programados de Amazon CloudWatch Events. Para definir una programación, puede especificar una frecuencia fija (un número de horas, días o semanas) o una expresión cron (consulte [Programar expresiones para reglas](#) en la Guía del usuario de Amazon CloudWatch).

El mapeo de origen de eventos se mantiene en Amazon CloudWatch Events mediante la utilización de una definición de destino de regla. Para obtener más información, consulte la operación [PutTargets](#) en la Referencia de la API de Amazon CloudWatch Events.

Para ver un ejemplo de caso de uso, consulte [Uso de AWS Lambda con eventos programados \(p. 302\)](#).

Para ver un ejemplo de evento, consulte [Evento de muestra de evento programado \(p. 148\)](#).

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. Amazon CloudWatch Events está configurado para invocar una función de Lambda de forma asíncrona. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

AWS Config

Puede utilizar funciones de AWS Lambda para evaluar si las configuraciones de los recursos de AWS cumplen sus reglas personalizadas de AWS Config. A medida que se crean, eliminan o cambian recursos, AWS Config registra estos cambios y envía la información a las funciones de Lambda. A continuación, las funciones de Lambda, evalúan los cambios y envían los resultados a AWS Config. A continuación, puede utilizar AWS Config para evaluar la conformidad general de los recursos: puede averiguar qué recursos no son conformes y los atributos de configuración que provocan esta situación.

El mapeo de origen de eventos se mantiene en AWS Config mediante la utilización de una definición de destino de regla. Para obtener más información, consulte la operación [PutConfigRule](#) en la referencia de la API de AWS Config.

Para obtener más información, consulte [Evaluación de recursos con reglas de AWS Config](#). Para ver un ejemplo de definición de una regla personalizada, consulte [Desarrollo de una regla personalizada para](#)

[AWS Config](#). Para ver ejemplos de funciones de Lambda, consulte [Funciones AWS Lambda de ejemplo para reglas de AWS Config \(Node.js\)](#).

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. AWS Config está configurado para invocar una función de Lambda de forma asíncrona. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Amazon Alexa

Puede utilizar funciones de Lambda para crear servicios que aporten nuevas habilidades a Alexa, el asistente de voz de Amazon Echo. Alexa Skills Kit proporciona las API, las herramientas y la documentación para crear estas nuevas habilidades, basadas en sus propios servicios que se ejecutan como funciones de Lambda. Los usuarios de Amazon Echo pueden obtener acceso a estas nuevas habilidades haciendo preguntas a Alexa o realizando solicitudes. Para obtener más información, consulte:

- [Getting Started with Alexa Skills Kit](#).
- [alexa-skills-kit-sdk-for-nodejs](#)
- [alexa-skills-kit-java](#)

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. Amazon Echo está configurado para invocar una función de Lambda de forma síncrona. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Amazon Lex

Amazon Lex es un servicio de AWS para crear interfaces de conversación en aplicaciones utilizando voz y texto. Amazon Lex ofrece integración integrada con AWS Lambda, lo que permite crear funciones de Lambda para utilizarlas como enlace de código con el bot de Amazon Lex. En la configuración de la intención, puede identificar la función de Lambda para realizar la inicialización/validación, cumplimiento o ambos.

Para obtener más información, consulte [Uso de funciones Lambda](#). Para ver un ejemplo de caso de uso, consulte [Ejercicio 1: creación de un bot de Amazon Lex mediante un proyecto](#).

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. Amazon Lex está configurado para invocar una función de Lambda de forma síncrona. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Amazon API Gateway

Puede invocar una función de Lambda a través de HTTPS. Puede hacerlo definiendo un punto de enlace y una API de REST personalizados utilizando Amazon API Gateway. Puede mapear operaciones individuales de la API, como, por ejemplo, GET y PUT, a determinadas funciones de Lambda. Cuando se envía una solicitud HTTPS al punto de enlace de la API, el servicio de Amazon API Gateway invoca la función de Lambda correspondiente.

Para obtener más información, consulte [Realización de llamadas síncronas a funciones de Lambda](#). Para ver un ejemplo de caso de uso, consulte [Uso de AWS Lambda con Amazon API Gateway \(bajo demanda a través de HTTPS\) \(p. 275\)](#).

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. Amazon API Gateway está configurado para invocar una función de Lambda de forma síncrona. Para

obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Además, también puede utilizar funciones de Lambda con otros servicios de AWS que publican datos en uno de los orígenes de eventos de AWS compatibles que se mencionan en este tema. Por ejemplo, puede hacer lo siguiente:

- Activar funciones de Lambda en respuesta a las actualizaciones de CloudTrail, ya que registra todos los eventos de acceso a la API en un bucket de Amazon S3.
- Activar funciones de Lambda en respuesta a las alarmas de CloudWatch, ya que publica eventos de alarma en un tema de Amazon SNS.

Botón de AWS IoT

El botón de AWS IoT es un botón programable basado en el hardware Amazon Dash Button. Este sencillo dispositivo wifi es fácil de configurar y está diseñado para que los desarrolladores comiencen a utilizar AWS Lambda, entre muchos otros servicios de AWS, sin necesidad de escribir código específico para el dispositivo.

Puede codificar la lógica del botón en la nube para configurar los clics del botón de manera que cuenten o sigan elementos, llamen o avisen a alguien, inicien o detengan algo, soliciten servicios o incluso proporcionen comentarios. Por ejemplo, puede hacer clic en el botón para abrir las puertas de un automóvil o ponerlo en marcha, abrir la puerta del garaje, llamar un taxi, llamar a su pareja o a un representante de atención al cliente, supervisar la realización de tareas domésticas comunes o el uso de fármacos o productos, o controlar sus electrodomésticos de forma remota.

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. AWS IoT está configurado para invocar una función de Lambda de forma asíncrona. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Amazon CloudFront

Lambda@Edge le permite ejecutar funciones de Lambda en regiones de AWS y ubicaciones de borde de Amazon CloudFront en respuesta a eventos de CloudFront, sin necesidad de aprovisionar ni administrar servidores. Puede utilizar funciones de Lambda para cambiar las solicitudes y las respuestas de CloudFront en los siguientes puntos:

- Despues de que CloudFront reciba una solicitud de un espectador (solicitud del espectador)
- Antes de que CloudFront reenvíe la solicitud al origen (solicitud al origen)
- Despues de que CloudFront reciba la respuesta del origen (respuesta del origen)
- Antes de que CloudFront reenvíe la respuesta al espectador (respuesta al espectador)

Para obtener más información, consulte [AWS Lambda@Edge \(p. 317\)](#)

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. CloudFront está configurado para invocar una función de Lambda de forma síncrona. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Amazon Kinesis Data Firehose

Amazon Kinesis Data Firehose es la forma más fácil de cargar datos de streaming en AWS. Puede capturar, transformar y cargar datos de streaming en servicios posteriores como Kinesis Data Analytics

o Amazon S3, lo que permite realizar análisis casi en tiempo real con las herramientas de inteligencia empresarial existentes y los paneles que se usan hoy día. Es posible escribir funciones de Lambda para realizar un procesamiento adicional personalizado de los datos antes de enviarlos a los servicios posteriores.

El control de errores de cada origen de eventos depende de cómo se invoca la función de Lambda. Kinesis Data Firehose está configurado para invocar una función de Lambda de forma síncrona. Para obtener más información acerca de cómo se reintentan los errores, consulte [Reintentos si se producen errores \(p. 192\)](#).

Otros orígenes de eventos: invocación de una función de Lambda bajo demanda

Además de invocar funciones de Lambda utilizando orígenes de eventos, también puede invocar una función de Lambda bajo demanda. En este caso, no es necesario preconfigurar ningún mapeo de origen de eventos. Sin embargo, debe asegurarse de que la aplicación personalizada dispone de los permisos necesarios para invocar la función de Lambda.

Por ejemplo, las aplicaciones de los usuarios también pueden generar eventos (crear sus propios orígenes de eventos personalizados). Las aplicaciones de los usuarios, como las aplicaciones cliente, móviles o web, pueden publicar eventos e invocar funciones de Lambda mediante los SDK de AWS o los AWS Mobile SDK, como el SDK de AWS Mobile para Android.

Para obtener más información, consulte [Herramientas para Amazon Web Services](#). Para ver un tutorial de ejemplo, consulte [Uso de AWS Lambda con Amazon API Gateway \(bajo demanda a través de HTTPS\) \(p. 275\)](#).

Eventos de muestra publicados por los orígenes de eventos

A continuación se ofrece una lista de los eventos de muestra publicados por los servicios compatibles de AWS. Para obtener más información acerca de los orígenes de eventos que admite AWS, consulte [Orígenes de eventos admitidos \(p. 138\)](#).

Eventos de muestra

- [Evento de muestra de creación de solicitud de AWS CloudFormation \(p. 147\)](#)
- [Evento de muestra de recepción de correo electrónico de Amazon SES \(p. 147\)](#)
- [Evento de muestra de evento programado \(p. 148\)](#)
- [Evento de muestra de Amazon CloudWatch Logs \(p. 148\)](#)
- [Evento de muestra de Amazon SNS \(p. 148\)](#)
- [Evento de muestra de actualización de Amazon DynamoDB \(p. 149\)](#)
- [Evento de muestra de disparador de sincronización de Amazon Cognito \(p. 150\)](#)
- [Evento de muestra de Amazon Kinesis Data Streams \(p. 151\)](#)
- [Evento de muestra Put de Amazon S3 \(p. 151\)](#)
- [Evento de muestra Delete de Amazon S3 \(p. 152\)](#)
- [Evento de muestra de Amazon Lex \(p. 152\)](#)
- [Evento de solicitud de proxy de API Gateway \(p. 153\)](#)
- [Evento de respuesta de proxy de API Gateway \(p. 154\)](#)
- [Evento de CloudFront \(p. 154\)](#)
- [Evento de AWS Config \(p. 155\)](#)
- [Evento de botón de AWS IoT \(p. 155\)](#)

- Evento de Kinesis Data Firehose (p. 155)

Evento de muestra de creación de solicitud de AWS CloudFormation

```
{  
    "Records": [  
        {  
            "EventVersion": "1.0",  
            "EventSource": "aws:sns",  
            "EventSubscriptionArn": "arn:aws:sns:us-west-2:0000000000:ses_messages:26a58451-3392-4ab6-a829-d65c2968421a",  
            "Sns": {  
                "MessageId": "211121b2-7aa4-5a8f-91ec-705a468b3024",  
                "Signature": "  
askdfjhaslkdfhalskjdfhalyrle234jh213k4jhl12k34h2134uqwf",  
                "Type": "Notification",  
                "TopicArn": "arn:aws:sns:us-west-2:00000000:ses_messages",  
                "MessageAttributes": {},  
                "SignatureVersion": "1",  
                "Timestamp": "2017-07-18T16:40:15.619Z",  
                "SigningCertUrl": "https://sns.us-west-2.amazonaws.com/  
SimpleNotificationService-0000000.pem",  
                "Message": "StackId='arn:aws:cloudformation:us-west-2:00000000:stack/testlin/c6a2d3f0-6bd7-11e7-aed2-50a68a'\nTimestamp='2017-07-18T16:40:15.346Z'\nEventId='c6a3e560-6bd7-11e7-aed2-50012ba'\nLogicalResourceId='testlin'\nNamespace='0000000000'\nPhysicalResourceId='arn:aws:cloudformation:us-west-2:00000000:stack/testlin/c6a2d3f0-6bd7-11e7-aed2-50a68a2012ba'\nPrincipalId='AROAJONWXY64:Isengard'\nResourceProperties='null'\nResourceStatus=Initiated'\nResourceType='AWS::CloudFormation::Stack'\nStackName='testlin'\nClientRequestToken='CoCreateStack-e18403e9-743c-4a8b-96aa-29361'\n",  
                "UnsubscribeUrl": "https://sns.us-west-2.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-west-2:00000000:ses_messages:26a58451-3392-4ab6-a829-d65c29",  
                "Subject": "AWS CloudFormation Notification"  
            }  
        }  
    ]  
}
```

Evento de muestra de recepción de correo electrónico de Amazon SES

```
{  
    "Records": [  
        {  
            "EventVersion": "1.0",  
            "EventSource": "aws:sns",  
            "EventSubscriptionArn": "arn:aws:sns:us-west-2:0000000000:ses_messages:26a58451-3392-4ab6-a829-d65c2968421a",  
            "Sns": {  
                "MessageId": "483eae4c-4fb0-57e5-a5f9-ff9b08612",  
                "Signature": "asdadasdasdasdasdasd",  
                "Type": "Notification",  
                "TopicArn": "arn:aws:sns:us-west-2:0000000000:ses_messages",  
                "MessageAttributes": {},  
                "SignatureVersion": "1",  
                "Timestamp": "2017-07-05T20:01:21.366Z",  
                "SigningCertUrl": "https://sns.us-west-2.amazonaws.com/  
SimpleNotificationService-0000000000.pem",  
                "Message": "{ \"notificationType\": \"Delivery\", \"mail\": {\"timestamp\": \"2017-07-05T20:01:20.773Z\", \"source\": \"example@amazon.com\", \"sourceArn\": \"arn:aws:ses:us-west-2:0000000000:identity/example@amazon.com\", \"sourceIp\": \"205.251.233.183\", \"sendingAccountId\": \"0000000000\", \"messageId\": \"0101017bd85-2ff839b3-c119-4311-b90c-5ce39eff3026-000000\", \"destination\": \"\" } }"}  
            }  
        }  
    ]  
}
```

```
[\"example@amazon.com\"], \"delivery\": {\"timestamp\": \"2017-07-05T20:01:21.302Z\", \"processingTimeMillis\": 529, \"recipients\": [\"example@amazon.com\"], \"smtpResponse\": \"250 ok: Message 122614849 accepted\", \"remoteMtaIp\": \"192.168.1.1\", \"reportingMTA\": \"smtp-out.us-west-2.amazonaws.com\"}}},\n        \"UnsubscribeUrl\": \"https://sns.us-\nwest-2.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-\nwest-2:000000000000:ses_messages:26a58451-3392-4ab6-a829-d65c1a\",\n        \"Subject\": null\n    }\n}\n]
```

Evento de muestra de evento programado

```
{\n    \"account\": \"123456789012\", \n    \"region\": \"us-east-1\", \n    \"detail\": {}, \n    \"detail-type\": \"Scheduled Event\", \n    \"source\": \"aws.events\", \n    \"time\": \"1970-01-01T00:00:00Z\", \n    \"id\": \"cdc73f9d-aea9-11e3-9d5a-835b769c0d9c\", \n    \"resources\": [\n        \"arn:aws:events:us-east-1:123456789012:rule/my-schedule\"\n    ]\n}
```

Evento de muestra de Amazon CloudWatch Logs

```
{\n    \"awslogs\": {\n        \"data\": \"H4sIAAAAAAAAAHWPwQqCQBCGX0Xm7EFtK\n+smZBEUgXoLCdMhFtKV3akI8d0bLYmibvPPN3wz00CJxmQnTO41whwWQRICtmEcB6sQbFC3CjW3XW8kxpOpP\n+OC22d1Wml1qZkQGtoMsScxaczKN3plG8zlaHIta5KqWszoTxw3/djzwhpLwivWFHGpAFe7DL68JlBUk\n+l7KSN7tCOEJ4M3/qOI49vMHj+zCKd1FqLaU2ZHV2a4Ct/an0/ivdX8oYc1UVX860fQDQiMdxRQEAAA==\"\n    }\n}
```

Evento de muestra de Amazon SNS

```
{\n    \"Records\": [\n        {\n            \"EventVersion\": \"1.0\", \n            \"EventSubscriptionArn\": eventsubscriptionarn, \n            \"EventSource\": \"aws:sns\", \n            \"Sns\": {\n                \"SignatureVersion\": \"1\", \n                \"Timestamp\": \"1970-01-01T00:00:00.000Z\", \n                \"Signature\": \"EXAMPLE\", \n                \"SigningCertUrl\": \"EXAMPLE\", \n                \"MessageId\": \"95df01b4-ee98-5cb9-9903-4c221d41eb5e\", \n                \"Message\": \"Hello from SNS!\", \n                \"MessageAttributes\": {\n                    \"Test\": {\n                        \"Type\": \"String\", \n                        \"Value\": \"TestString\"\n                    },\n                    \"TestBinary\": {\n                        \"Type\": \"Binary\", \n                        \"Value\": \"TestBinary\"\n                    }\n                }\n            }\n        }\n    ]\n}
```

```
        },
    },
    "Type": "Notification",
    "UnsubscribeUrl": "EXAMPLE",
    "TopicArn": topicarn,
    "Subject": "TestInvoke"
}
]
}
```

Evento de muestra de actualización de Amazon DynamoDB

```
{
"Records": [
{
    "eventID": "1",
    "eventVersion": "1.0",
    "dynamodb": {
        "Keys": {
            "Id": {
                "N": "101"
            }
        },
        "NewImage": {
            "Message": {
                "S": "New item!"
            },
            "Id": {
                "N": "101"
            }
        },
        "StreamViewType": "NEW_AND_OLD_IMAGES",
        "SequenceNumber": "111",
        "SizeBytes": 26
    },
    "awsRegion": "us-west-2",
    "eventName": "INSERT",
    "eventSourceARN": eventsourcearn,
    "eventSource": "aws:dynamodb"
},
{
    "eventID": "2",
    "eventVersion": "1.0",
    "dynamodb": {
        "OldImage": {
            "Message": {
                "S": "New item!"
            },
            "Id": {
                "N": "101"
            }
        },
        "SequenceNumber": "222",
        "Keys": {
            "Id": {
                "N": "101"
            }
        },
        "SizeBytes": 59,
        "NewImage": {
            "Message": {
                "S": "This item has changed"
            },

```

```
        "Id": {
            "N": "101"
        },
        "StreamViewType": "NEW_AND_OLD_IMAGES"
    },
    "awsRegion": "us-west-2",
    "eventName": "MODIFY",
    "eventSourceARN": sourcearn,
    "eventSource": "aws:dynamodb"
},
{
    "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"
}
]
```

Evento de muestra de disparador de sincronización de Amazon Cognito

```
{
    "datasetName": "datasetName",
    "eventType": "SyncTrigger",
    "region": "us-east-1",
    "identityId": "identityId",
    "datasetRecords": {
        "SampleKey2": {
            "newValue": "newValue2",
            "oldValue": "oldValue2",
            "op": "replace"
        },
        "SampleKey1": {
            "newValue": "newValue1",
            "oldValue": "oldValue1",
            "op": "replace"
        }
    },
    "identityPoolId": "identityPoolId",
    "version": 2
}
```

Evento de muestra de Amazon Kinesis Data Streams

```
"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": eventsourcarn,
    "eventSource": "aws:kinesis",
    "awsRegion": "us-east-1"
  }
]
```

Evento de muestra Put de Amazon S3

```
{
  "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": "EXAMPLE123/5678abcdefghijklmklambdaisawesome/",
        "x-amz-request-id": "EXAMPLE123456789"
      },
      "awsRegion": "us-east-1",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "EXAMPLE"
      },
      "versionId": "EXAMPLE123456789"
    }
  ]
}
```

```
        "eventSource": "aws:s3"
    }
}
```

Evento de muestra Delete de Amazon S3

```
{
"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": "EXAMPLE123/5678abcdefghijklmabaisawesome/
mnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ",
        "x-amz-request-id": "EXAMPLE123456789"
    },
    "awsRegion": "us-east-1",
    "eventName": "ObjectRemoved:Delete",
    "userIdentity": {
        "principalId": "EXAMPLE"
    },
    "eventSource": "aws:s3"
}
]
}
```

Evento de muestra de Amazon Lex

```
{
    "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)"
}
}
```

Evento de solicitud de proxy de API Gateway

```
{
    "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"
}
}
```

Evento de respuesta de proxy de API Gateway

```
{
    "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"
}
```

Evento de CloudFront

```
{
    "Records": [
        {
            "cf": {
                "config": {
                    "distributionId": "EDFDVBD6EXAMPLE"
                },
                "request": {
                    "clientIp": "2001:0db8:85a3:0:0:8a2e:0370:7334",
                    "method": "GET",
                    "uri": "/picture.jpg",
                    "headers": {
                        "host": [
                            {
                                "key": "Host",
                                "value": "d11111abcdef8.cloudfront.net"
                            }
                        ],
                        "user-agent": [
                            {
                                "key": "User-Agent",

```

```
        "value": "curl/7.51.0"
    }
}
}
]
}
```

Evento de AWS Config

```
{
    "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\":{}, \"relationships\":[{\"resourceId\":\"eipalloc-00000000\", \"resourceType\":\"AWS::EIP\", \"name\":\"Is attached to ElasticIp\"}], \"configuration\":{}, \"messageType\":\"ConfigurationItemChangeNotification\"},\n        \"ruleParameters\": {\"myParameterKey\":\"myParameterValue\"},\n        \"resultToken\": \"myResultToken\",\n        \"eventLeftScope\": false,\n        \"executionRoleArn\": \"arn:aws:iam::012345678912:role/config-role\",\n        \"configRuleArn\": \"arn:aws:config:us-east-1:012345678912:config-rule/config-rule-0123456\",\n        \"configRuleName\": \"change-triggered-config-rule\",\n        \"configRuleId\": \"config-rule-0123456\",\n        \"accountId\": \"012345678912\",\n        \"version\": \"1.0\"\n    }
```

Evento de botón de AWS IoT

```
{
    "serialNumber": "ABCDEFG12345",
    "clickType": "SINGLE",
    "batteryVoltage": "2000 mV"
}
```

Evento de Kinesis Data Firehose

```
{
    "invocationId": "invoked123",
    "deliveryStreamArn": "aws:lambda:events",
    "region": "us-west-2",
    "records": [
        {
            "data": "SGVsbtG8gV29ybGQ=",
            "recordId": "record1",
            "approximateArrivalEpoch": 123456789,
            "approximateArrivalTimestamp": "2017-01-01T00:00:00.000Z",
            "kinesisRecordMetadata": {
                "shardId": "shardId-000000000000",
                "partitionKey": "4d1ad2b9-24f8-4b9d-a088-76e9947c317a",
                "approximateArrivalTimestamp": "2012-04-23T18:25:43.511Z",
                "sequenceNumber": "49546986683135544286507457936321625675700192471156785154",
                "subsequenceNumber": ""
            }
        }
    ]
}
```

```
        }
    },
{
    "data": "SGVsbG8gV29ybGQ=",
    "recordId": "record2",
    "approximateArrivalEpoch": 123456789101112,
    "approximateArrivalTimestamp": "2018-01-01T00:00:00.000Z",
    "kinesisRecordMetadata": {
        "shardId": "shardId-000000000001",
        "partitionKey": "4d1ad2b9-24f8-4b9d-a088-76e9947c318a",
        "approximateArrivalTimestamp": "2012-04-23T19:25:43.511Z",
        "sequenceNumber": "49546986683135544286507457936321625675700192471156785155",
        "subsequenceNumber": ""
    }
}
]
```

Implementación de aplicaciones basadas en Lambda

Las aplicaciones basadas en Lambda (también denominadas aplicaciones sin servidor) contienen funciones activadas por eventos. Una aplicación sin servidor típica consta de una o varias funciones activadas por eventos, como cargas de objetos a Amazon S3, notificaciones de Amazon SNS y acciones de la API. Estas funciones pueden ser independientes o utilizar otros recursos como tablas de DynamoDB o buckets de Amazon S3. La aplicación sin servidor más básica es simplemente una función.

AWS Lambda proporciona operaciones de la API que puede utilizar para crear y actualizar funciones de Lambda proporcionando un paquete de implementación en un archivo ZIP. Sin embargo, este mecanismo podría no ser conveniente para automatizar los pasos de implementación de las funciones, o para coordinar implementaciones y actualizaciones con otros elementos de una aplicación sin servidor (como los orígenes de eventos y los recursos posteriores). Por ejemplo, para implementar un disparador de Amazon SNS, debe actualizar la función, el tema de Amazon SNS, el mapeo entre la función y el tema, y cualquier otro recurso posterior que necesite la función como, por ejemplo, una tabla de DynamoDB.

Implementación de aplicaciones sin servidor mediante AWS CloudFormation

Puede utilizar AWS CloudFormation para especificar, implementar y configurar aplicaciones sin servidor. AWS CloudFormation es un servicio que le ayuda a modelar y configurar recursos de AWS, por lo que podrá dedicar menos tiempo a la administración de dichos recursos y más tiempo a centrarse en las aplicaciones que se ejecutan en AWS. Puede crear una plantilla que describa todos los recursos de AWS que desea (como funciones de Lambda y tablas de DynamoDB), y AWS CloudFormation se encargará del aprovisionamiento y la configuración de dichos recursos. No es necesario crear y configurar individualmente los recursos de AWS ni averiguar qué depende de qué. AWS CloudFormation se encarga de todo eso. Para obtener más información, consulte [Conceptos de AWS CloudFormation](#) en la Guía del usuario de AWS CloudFormation.

Uso de AWS Serverless Application Model (AWS SAM)

AWS Serverless Application Model (AWS SAM) es un modelo para definir aplicaciones sin servidor. AWS CloudFormation admite de forma nativa AWS SAM y define una sintaxis simplificada para expresar

recursos sin servidor. La especificación actualmente incluye las API, las funciones de Lambda y las tablas de Amazon DynamoDB. La especificación está disponible con licencia Apache 2.0 para que los socios y los clientes de AWS la adopten y amplíen dentro de sus propios conjuntos de herramientas. Para obtener información detallada sobre la especificación, consulte [AWS Serverless Application Model](#).

Recursos sin servidor de AWS CloudFormation

AWS SAM admite tipos de recursos especiales que simplifican la forma de expresar funciones, API, mapeos y tablas de DynamoDB para aplicaciones sin servidor, así como algunas de características de estos servicios, como las variables de entorno. La descripción de estos recursos en AWS CloudFormation se ajusta al [AWS Serverless Application Model](#). Para implementar la aplicación, solo tiene que especificar los recursos que necesita que formen parte de ella, junto con sus políticas de permisos asociadas en un archivo de plantilla de AWS CloudFormation (escrito en JSON o YAML), empaquetar sus artefactos de implementación e implementar la plantilla.

Una plantilla de AWS CloudFormation con recursos sin servidor adaptada al modelo AWS SAM se denomina plantilla o archivo SAM.

Los siguientes ejemplos ilustran cómo utilizar AWS SAM para declarar los componentes comunes de una aplicación sin servidor. Tenga en cuenta que los valores de los parámetros `Handler` y `Runtime` deben coincidir con los que utilizó cuando creó la función en la sección anterior.

Función de Lambda

A continuación se muestra la notación que se utiliza para describir una función de Lambda:

```
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
```

El valor `handler` de la propiedad `Handler` apunta al módulo que contiene el código que se ejecutará cuando se invoque la función de Lambda. El valor `index` de la propiedad `Handler` indica el nombre del archivo que contiene el código. Puede declarar tantas funciones como necesite la aplicación sin servidor.

También puede declarar variables de entorno, que son opciones de configuración que puede definir para la aplicación. A continuación se muestra un ejemplo de una aplicación sin servidor con dos funciones de Lambda y una variable de entorno que apunta a una tabla de DynamoDB. Puede actualizar las variables de entorno sin necesidad de modificar, ni volver a empaquetar o implementar el código de la función de Lambda. Para obtener más información, consulte [Variables de entorno \(p. 97\)](#).

```
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
```

Tenga en cuenta la notación del principio:

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

Es necesaria para incluir los objetos definidos por AWS Serverless Application Model en una plantilla de AWS CloudFormation.

SimpleTable

SimpleTable es un recurso que crea una tabla de DynamoDB con una clave principal de un solo atributo. Puede utilizar esta versión simplificada si para acceder a los datos con los que interactúa la aplicación sin servidor solo se necesita una clave de un solo valor. Puede actualizar el ejemplo anterior para utilizar un recurso SimpleTable, como se muestra a continuación:

```
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
```

Eventos

Los eventos son recursos de AWS que activan la función de Lambda, como, por ejemplo, un punto de enlace de Amazon API Gateway o una notificación Amazon SNS. La propiedad `Events` es una matriz, lo que le permite definir varios eventos para cada función. A continuación se muestra la notación que se utiliza para describir una función de Lambda con una tabla de DynamoDB como origen de eventos:

```
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
```

Como se mencionó antes, puede configurar varios orígenes de eventos que activarán la función de Lambda. En el siguiente ejemplo se muestra una función de Lambda que puede activarse mediante un evento PUT o POST de HTTP.

API

Existen dos maneras de definir una API mediante AWS SAM. A continuación se utiliza Swagger para configurar los recursos subyacentes de Amazon API Gateway:

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  Api:
    Type: AWS::Serverless::Api
    Properties:
      StageName: prod
      DefinitionUri: swagger.yml
```

En el siguiente ejemplo, el tipo de recurso `AWS::Serverless::Api` se añade implícitamente a partir de la unión de los eventos de la API definidos en los recursos `AWS::Serverless::Function`.

```
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
```

En el ejemplo anterior, AWS CloudFormation generará automáticamente una API de Amazon API Gateway con la ruta "/resource/{resourceId}" y con los métodos GET, PUT y DELETE.

Permisos

Puede proporcionar un nombre de recurso de Amazon (ARN) para el rol de AWS Identity and Access Management (IAM) que se utilizará como rol de ejecución de esta función, como se muestra a continuación:

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  FunctionName:
    Type: AWS::Serverless::Function
    Properties:
      Role:role arn
```

También puede proporcionar una o varias políticas administradas para el recurso de la función de Lambda. A continuación, AWS CloudFormation creará un rol nuevo con las políticas administradas, junto con la política de ejecución básica predeterminada de Lambda.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources:
  FunctionName:
    Type: AWS::Serverless::Function
    Properties:
      Policies: AmazonDynamoDBFullAccess
```

Si no se proporciona ninguno de estos elementos, se creará un rol de ejecución predeterminado con los permisos de ejecución básicos de Lambda.

Note

Además de los recursos sin servidor, también puede utilizar la sintaxis convencional de AWS CloudFormation para expresar recursos de la misma plantilla. Cualquier recurso que no esté incluido en el modelo SAM actual se puede crear en la plantilla de AWS CloudFormation utilizando la sintaxis de AWS CloudFormation. Además, puede utilizar la sintaxis de AWS CloudFormation para expresar recursos sin servidor como alternativa a utilizar el modelo SAM. Para obtener más información acerca de cómo especificar una función de Lambda utilizando la sintaxis convencional de CloudFormation como parte de la plantilla de SAM, consulte [AWS::Lambda::Function](#) en la Guía del usuario de AWS CloudFormation.

Para obtener una lista completa de ejemplos de aplicaciones sin servidor, consulte [Ejemplos de cómo utilizar AWS Lambda \(p. 214\)](#).

Paso siguiente

[Creación de su propia aplicación sin servidor \(p. 161\)](#)

Creación de su propia aplicación sin servidor

En el siguiente tutorial, creará una aplicación sin servidor sencilla con una sola función que devuelve el nombre de un bucket de Amazon S3 especificado como una variable de entorno. Siga estos pasos:

1. Copie y pegue lo siguiente en un archivo de texto y guárdelo como `index.js`.

```
var AWS = require('aws-sdk');

exports.handler = function(event, context, callback) {
  var bucketName = process.env.S3_BUCKET;
```

```
    callback(null, bucketName);
}
```

2. Pegue lo siguiente en un archivo de texto y guárdelo como `example.yaml`. Observe que el parámetro `Runtime` utiliza `nodejs6.10` pero puede especificar `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. Cree una carpeta denominada `examplefolder` y coloque el archivo `example.yaml` y el archivo `index.js` dentro de la carpeta.

La carpeta `examplefolder` contiene los dos archivos siguientes que puede utilizar para empaquetar la aplicación sin servidor:

- `example.yaml`
- `index.js`

Empaquetado e implementación

Después de crear el controlador de la función de Lambda y el archivo `example.yaml`, puede utilizar la AWS CLI para empaquetar e implementar la aplicación sin servidor.

Embalaje

Para empaquetar la aplicación, cree el bucket de Amazon S3 que el comando `package` utilizará para cargar el paquete de implementación ZIP (si no ha especificado uno en el archivo `example.yaml`). Puede utilizar el siguiente comando para crear el bucket de Amazon S3:

```
aws s3 mb s3://bucket-name --region region
```

A continuación, abra un símbolo del sistema y escriba lo siguiente:

```
aws cloudformation package \
--template-file example.yaml \
--output-template-file serverless-output.yaml \
--s3-bucket s3-bucket-name
```

El comando `package` devuelve una plantilla de AWS SAM, en este caso `serverless-output.yaml`, que contiene la propiedad `CodeUri` que apunta al zip de implementación situado en el bucket de Amazon S3 especificado. Esta plantilla representa la aplicación sin servidor. Ahora ya puede implementarla.

Implementación

Para implementar la aplicación, ejecute el siguiente comando:

```
aws cloudformation deploy \
--template-file serverless-output.yaml \
```

```
--stack-name new-stack-name \
--capabilities CAPABILITY_IAM
```

Observe que el valor especificado para el parámetro `--template-file` es el nombre de la plantilla de SAM que devolvió el comando `package`. Además, el parámetro `--capabilities` es opcional. El recurso `AWS::Serverless::Function` creará de forma implícita un rol para ejecutar la función de Lambda si no se ha especificado uno en la plantilla. El parámetro `--capabilities` se utiliza para reconocer explícitamente que AWS CloudFormation tiene permiso para crear roles en su nombre.

Cuando se ejecuta el comando `aws cloudformation deploy`, crea una `ChangeSet` de AWS CloudFormation, que es una lista de cambios para la pila de AWS CloudFormation y, a continuación, la implementa. Puede haber plantillas de pila que incluyan recursos que puedan afectar a los permisos de la cuenta de AWS, por ejemplo, creando usuarios nuevos de AWS Identity and Access Management (IAM). Debe reconocer explícitamente las capacidades de estas pilas especificando el parámetro `--capabilities`. Para obtener más información, consulte [CreateChangeSet](#) en la AWS CloudFormation API Reference.

Para verificar los resultados, abra la consola de AWS CloudFormation para ver la pila de AWS CloudFormation que acaba de crear y la consola de Lambda para ver la función.

Para obtener una lista completa de ejemplos de aplicaciones sin servidor, consulte [Ejemplos de cómo utilizar AWS Lambda \(p. 214\)](#).

Exportación de una aplicación sin servidor

Puede utilizar la consola de Lambda para exportar una aplicación sin servidor y volver a implementarla, por ejemplo, en otra región de AWS o en otra etapa de desarrollo. Cuando se exporta una función de Lambda, se obtiene un paquete de implementación ZIP y una plantilla de SAM que representa la aplicación sin servidor. A continuación, puede utilizar los comandos `package` y `deploy` descritos en la sección anterior para volver a realizar la implementación.

También puede seleccionar uno de los proyectos de Lambda para crear un paquete ZIP para realizar el empaquetado y la implementación. Para ello, siga estos pasos:

Para exportar una aplicación sin servidor mediante la consola de Lambda

1. Inicie sesión en la Consola de administración de AWS y abra la consola de AWS Lambda en <https://console.aws.amazon.com/lambda/>.
2. Realice uno de los siguientes procedimientos:
 - Cree una función mediante un proyecto de Lambda: elija un proyecto y siga los pasos para crear una función de Lambda. Para ver un ejemplo, consulte [Paso 2.1: Creación de una función Hello World de Lambda \(p. 203\)](#). Cuando llegue a la página Review, elija Export function.
 - Cree una función: elija Create function y, a continuación, cree la función. Después de crear la función de Lambda, puede seleccionarla para exportarla. Elija Actions, seguido de Export function.
 - Abra una función de Lambda existente: abra la función seleccionando nombre de la función, y elija Actions, seguido de Export function.
3. En la ventana Export your function, dispone de las siguientes opciones:
 - Elija Download AWS SAM file, que define la función de Lambda y los demás recursos que componen la aplicación sin servidor.
 - Elija Download deployment package, que contiene el código de la función de Lambda y todas las bibliotecas dependientes.

Utilice el archivo de AWS SAM y el paquete de implementación ZIP, y siga las instrucciones de [Empaquetado e implementación \(p. 162\)](#) para volver a implementar la aplicación sin servidor.

Automatización de la implementación de aplicaciones basadas en Lambda

En la sección anterior, ha aprendido a crear una plantilla de SAM, a generar el paquete de implementación y a utilizar la AWS CLI para implementar la aplicación sin servidor manualmente. En esta sección, utilizará los siguientes servicios de AWS para automatizar totalmente el proceso de implementación.

- CodePipeline: CodePipeline se utiliza para modelar, visualizar y automatizar los pasos necesarios para lanzar la aplicación sin servidor. Para obtener más información, consulte [What is AWS CodePipeline?](#)
- CodeBuild: CodeBuild se utiliza para crear, probar localmente y empaquetar aplicaciones sin servidor. Para obtener más información, consulte [¿Qué es AWS CodeBuild?](#)
- AWS CloudFormation: AWS CloudFormation se utiliza para implementar la aplicación. Para obtener más información consulte [¿Qué es AWS CloudFormation?](#)

Creación de una canalización para la aplicación sin servidor

En el siguiente tutorial, creará una AWS CodePipeline que automatiza la implementación de la aplicación sin servidor. En primer lugar, debe configurar una etapa de origen para activar la canalización. Para este tutorial:

- Utilizaremos GitHub. Para obtener instrucciones sobre cómo crear un repositorio de GitHub, consulte [Create a Repository in GitHub](#).
- Tendrá que crear un rol de AWS CloudFormation y añadirle la política AWSLambdaExecute, tal como se describe a continuación:
 1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.
 2. Siga los pasos de [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM para crear un rol de IAM (rol de ejecución) y vaya a la sección Para crear un rol para un servicio de AWS. Cuando siga los pasos para crear un rol, tenga en cuenta lo siguiente:
 - En Select Role Type, elija AWS Service Roles, seguido de AWS CloudFormation. Esto concede al servicio de AWS CloudFormation permisos para asumir el rol.
 - En Attach Policy, elija AWSLambdaExecute.
 - En Role Name, utilice un nombre exclusivo dentro de la cuenta de AWS (por ejemplo, cloudformation-lambda-ejecución-role) y, a continuación, elija Create role.
 - Abra el rol que acaba de crear y, en la pestaña Permissions, expanda Inline Policies y, a continuación, elija el enlace click here.
 - Elija Custom Policy, seguido de Select.
 - En Policy Name, introduzca un nombre para la política personalizada y, a continuación, pegue lo siguiente en el campo Policy Document:

```
{  
    "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:account-id:function:/*"
        ],
        "Effect": "Allow"
    },
    {
        "Action": [
            "apigateway:/*"
        ],
        "Resource": [
            "arn:aws:apigateway:region::/*"
        ],
        "Effect": "Allow"
    },
    {
        "Action": [
            "iam:GetRole",
            "iam:CreateRole",
            "iam:DeleteRole"
        ],
        "Resource": [
            "arn:aws:iam::account-id:role/*"
        ],
        "Effect": "Allow"
    },
    {
        "Action": [
            "iam:AttachRolePolicy",
            "iam:DetachRolePolicy"
        ],
        "Resource": [
            "arn:aws:iam::account-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"
    }
],
```

```
        "Version": "2012-10-17"  
    }
```

- Elija Validate Policy, seguido de Apply Policy.

Paso 1: Configuración del repositorio

Para configurar el repositorio, haga lo siguiente:

- Añada un archivo index.js que contenga el código siguiente:

```
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(),  
    });  
};
```

- Añada un archivo samTemplate.yaml con el contenido siguiente. Es la plantilla de SAM que define los recursos de la aplicación. Esta plantilla de SAM define una función de Lambda activada por API Gateway. Observe que el parámetro runtime utiliza nodejs6.10, pero también puede especificar nodejs4.3. Para obtener más información acerca de AWS SAM, consulte [AWS Serverless Application Model](#).

```
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
```

- Añada un archivo buildspec.yml. Una especificación de compilación es un conjunto de comandos de compilación y opciones relacionadas, en formato YAML, que AWS CodeBuild utiliza para ejecutar una compilación. Para obtener más información, consulte [Referencia de especificaciones de compilación para AWS CodeBuild](#). En este ejemplo, la acción de compilación será:
 - Utilizar npm para instalar el paquete de tiempo.
 - Ejecutar el comando Package para preparar el paquete de implementación para los pasos de implementación siguientes de la canalización. Para obtener más información sobre el comando package, consulte [Carga de artefactos locales en un bucket de S3](#).

```
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:
    - NewSamTemplate.yaml
```

Tenga en cuenta que debe proporcionar el valor del parámetro `--s3-bucket` con el nombre del bucket de Amazon S3, algo parecido a lo que haría si empaquetara manualmente el paquete de implementación con SAM, tal como se explicó en el paso [Embalaje \(p. 162\)](#) del tutorial anterior.

Paso 2: Creación de la canalización

Para crear la AWS CodePipeline, siga los pasos que se describen a continuación.

1. Inicie sesión en la Consola de administración de AWS y abra la consola de AWS CodePipeline.
2. Elija Get Started Now.
3. En Pipeline name, introduzca un nombre para la canalización y, a continuación, elija Next step.
4. En Source provider, elija GitHub.
5. Elija Connect to GitHub y, a continuación, rellene los campos Repository y Branch para especificar el destino de la conexión. Cada vez que ejecute un comando git push con destino a esa rama, se activará la canalización. Elija Next Step.
6. Elija AWS CodeBuild como Build provider.
7. Elija Create a new build project e introduzca un nombre de proyecto.
8. Elija Ubuntu como sistema operativo.
9. Elija Node.js como tiempo de ejecución.
10. En Version, elija `aws/codebuild/nodejs:4.3.2`.
11. Elija Save build project.

Note

Se creará automáticamente un rol de servicio para AWS CodeBuild en su nombre.

- Elija Next Step.
12. En Deployment provider, elija AWS CloudFormation.

Al seleccionar esta opción, se utilizarán comandos de AWS CloudFormation para implementar la plantilla de SAM. Para obtener más información, consulte [Recursos sin servidor de AWS CloudFormation \(p. 157\)](#).

13. En Action mode, elija create or replace a change set.
14. En Stack name, escriba MyBetaStack.
15. En Change set name, escriba MyChangeSet.
16. En Template file, escriba NewSamTemplate.yaml.
17. En Capabilities, elija CAPABILITY_IAM.
18. En Role, seleccione el rol de AWS CloudFormation que ha creado al principio de este tutorial y, a continuación, elija Next step.
19. Elija Create role. Elija Next, seguido de Allow. Elija Next Step.
20. Revise la canalización y, a continuación, elija Create pipeline.

Paso 3: Actualización de la política de servicio generada

Siga los pasos que se describen a continuación para permitir que CodeBuild cargue los artefactos de compilación en el bucket de Amazon S3.

1. Vaya a la consola de administración de IAM.
2. Elija Roles.
3. Abra el rol de servicio que se ha generado para el proyecto, normalmente code-build-**nombre-del-proyecto**-service-role.
4. En la pestaña Permissions, elija Attach Policy.
5. En la pestaña Permissions, elija Create Role Policy.
6. Elija Policy Generator, seguido de Select.
7. En la lista AWS Service, elija Amazon S3.
8. En Actions, elija PutObject.
9. En Amazon Resource Name (ARN), escriba `arn:aws:s3:::bucket-name*`.
10. Elija Add Statement, seguido de Next Step.
11. En Review Policy, elija Validate Policy, seguido de Apply Policy.

Paso 4: Finalización de la etapa de implementación Beta

Siga los pasos que se describen a continuación para completar la etapa Beta.

1. Elija Edit.
2. Elija el icono 
3. En la etapa Beta, elija el icono + Action que se encuentra junto a la acción existente.
4. En Category, elija Deploy.
5. En Action, introduzca execute_cs.
6. En Deployment provider, elija AWS CloudFormation.
7. En Action mode , elija execute a changeset. Esto es similar a la operación que realizaría si implementara el paquete manualmente, tal como se explicó en el paso [Implementación \(p. 162\)](#) del tutorial anterior. CreateChangeSet transforma la plantilla de SAM para que tenga el formato de AWS CloudFormation completo, y deployChangeSet implementa la plantilla de AWS CloudFormation.
8. En Stack name, escriba MyBetaStack.
9. En Change set name, escriba MyChangeSet.
10. Elija Add action.
11. Elija Save pipeline changes.
12. Elija Save and continue.

La canalización está lista. Cualquier comando git push cuyo destino sea la rama conectada a esta canalización activará una implementación. Para probar la canalización e implementar la aplicación por primera vez, realice una de las siguientes operaciones:

- Ejecute un comando git push con destino a la rama conectada a la canalización.
- Vaya a la consola de AWS CodePipeline, elija el nombre de la canalización que ha creado y, a continuación, elija Release change.

Prueba local de aplicaciones sin servidor mediante SAM Local (en versión beta pública)

Esta característica está disponible formando parte de una versión beta pública y está sujeta a cambios en cualquier momento.

Tal como se ha descrito anteriormente, [AWS SAM](#) es una forma rápida y sencilla de implementar aplicaciones sin servidor, lo que permite escribir plantillas sencillas para describir funciones y sus orígenes de eventos (Amazon API Gateway, Amazon S3, Kinesis, etc.). SAM Local es una herramienta de la AWS CLI que se basa en AWS SAM y que proporciona un entorno para desarrollar, probar y analizar localmente aplicaciones sin servidor antes de cargarlas en el tiempo de ejecución de Lambda. Puede utilizar SAM Local para crear un entorno de pruebas local que simule el entorno en tiempo de ejecución de AWS para desarrollos en Linux, Mac o Microsoft Windows. Esto le ayuda a abordar cuestiones como el desempeño. El uso de SAM Local también permite acelerar el desarrollo iterativo del código de una función de Lambda, ya que no es necesario volver a implementar el paquete de la aplicación en el tiempo de ejecución AWS Lambda. Para obtener más información, consulte [Creación de una sencilla con SAM Local \(p. 173\)](#).

SAM Local funciona con [AWS SAM](#), lo que permite invocar funciones definidas en las plantillas de SAM, ya sea directamente o a través de puntos de enlace de API Gateway. Las características de SAM Local le permiten analizar el desempeño de una aplicación sin servidor en su propio entorno de pruebas y actualizarla según sea necesario. Los siguientes ejemplos ilustran las ventajas adicionales de utilizar SAM Local con código de operación de muestra. Por ejemplo, puede hacer lo siguiente:

- Generar cargas de muestra para las funciones (por ejemplo, un evento de Amazon S3).

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

- Probar localmente una carga de muestra con funciones de Lambda.

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

- Generar un API Gateway local para probar la funcionalidad de solicitud y respuesta HTTP. La característica de recarga en caliente le permite probar e iterar funciones sin tener que reiniciarlas ni volver a cargarlas en el tiempo de ejecución de AWS.

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

SAM Local encontrará automáticamente todas las funciones existentes en una plantilla de SAM en las que se hayan definido orígenes de eventos de la API, y las montará en las rutas HTTP definidas. En el ejemplo siguiente, la función Ratings montaría ratings.py:handler() en /ratings para las solicitudes GET.

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

Method: get

De forma predeterminada, SAM Local utiliza la [integración de proxy](#) y espera que la respuesta de la función de Lambda incluya uno o varios de los elementos siguientes: statusCode, headers y/o body. Por ejemplo:

```
// 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"
  });
}
```

Si la función de Lambda no devuelve una respuesta válida de la [integración de proxy](#), recibirá una respuesta HTTP 500 (Internal Server Error) al acceder a la función. SAM Local también imprimirá el siguiente mensaje de error de log que le ayudará a diagnosticar el problema:

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

- Compruebe que se cumplan las limitaciones del tiempo de ejecución, como, por ejemplo, los límites de tiempo de espera o de uso máximo de memoria para las invocaciones de la función de Lambda.
- Examine los logs del tiempo de ejecución de AWS Lambda, así como cualquier salida de registro personalizada especificada en el código de la función de Lambda (por ejemplo, `console.log`). SAM Local muestra automáticamente esta salida. A continuación se muestra un ejemplo.

```
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
```

- Respete las credenciales de seguridad que ha establecido mediante la AWS CLI. Al hacerlo, la función de Lambda puede realizar llamadas remotas a los servicios de AWS que componen la aplicación sin servidor. Si no ha instalado la AWS CLI, consulte [Instalación de la AWS Command Line Interface](#).

Al igual que sucede con la AWS CLI y los SDK, SAM Local buscará las credenciales en el siguiente orden:

- Variables de entorno (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`)
- Archivo de credenciales de AWS, situado en `~/.aws/credentials` en Linux, MacOS o Unix, o en `C:\Users\USERNAME\.aws\credentials` en Windows
- Credenciales de perfil de la instancia, en caso de que se ejecute en una instancia de Amazon EC2 con un rol de instancia asignado

Tiempos de ejecución admitidos

SAM Local admite los siguientes tiempos de ejecución de AWS:

- node.js 4.3
- node.js 6.10

- python 2.7
- python 3.6
- java8

Requisitos para usar SAM Local

Para utilizar SAM Local, debe instalar Docker y SAM Local.

Instalar Docker

Docker es una plataforma de contenedores de software de código abierto que permite crear, administrar y probar aplicaciones, independientemente de que ejecute en Linux, Mac o Windows. Para obtener más información e instrucciones e descarga, consulte [Docker](#).

Cuando se ha instalado Docker, SAM Local proporciona automáticamente una imagen personalizada de Docker denominada `docker-lambda`. Esta imagen está diseñada específicamente por un socio de AWS para simular el entorno de ejecución real de AWS Lambda. Este entorno incluye software instalado, bibliotecas, permisos de seguridad, variables de entorno y otras características que se describen en [Entorno de ejecución de Lambda y bibliotecas disponibles \(p. 195\)](#).

Puede invocar la función de Lambda localmente utilizando `docker-lambda`. En este entorno, las aplicaciones sin servidor se ejecutan y funcionan de forma muy similar a como lo hacen en el tiempo de ejecución de AWS Lambda, sin que tenga que volver a implementar el tiempo de ejecución. Su ejecución y desempeño de este entorno tienen en cuenta aspectos como los tiempos de espera y el uso de la memoria.

Important

Dado que se trata de un entorno simulado, no se garantiza que los resultados de las pruebas locales coincidan exactamente con los del tiempo de ejecución real de AWS.

Para obtener más información, consulte [Docker Lambda](#) en [GitHub](#). (Si no tiene una cuenta de GitHub, puede crearla de forma gratuita y, a continuación, acceder a Docker Lambda).

Instalación de SAM Local

Puede ejecutar SAM Local en entornos Linux, Mac y Windows. La forma más sencilla de instalar SAM Local es utilizar [NPM](#).

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

A continuación, compruebe que la instalación se ha realizado correctamente.

```
sam --version
```

Si NPM no le funciona, puede descargar el archivo binario más reciente y comenzar a utilizar SAM Local inmediatamente. Puede encontrar los archivos binarios en la sección Releases del [repositorio de SAM Local en GitHub](#).

Introducción al uso de SAM Local

SAM Local está formado por las siguientes operaciones de la CLI:

- **start-api:** crea un servidor HTTP local que aloja todas las funciones de Lambda. Cuando accede a él a través de un navegador o de la CLI, esta operación lanza un contenedor de Docker localmente para invocar la función. Lee la propiedad `CodeUri` del recurso `AWS::Serverless::Function` para encontrar la ruta del sistema de archivos que contiene el código de la función de Lambda. Esta ruta puede ser el directorio raíz del proyecto para los lenguajes interpretados como Node.js o Python, un directorio de compilación que almacena los artefactos compilados o, para Java, un archivo `.jar`.

Si utiliza un lenguaje interpretado, las modificaciones locales estarán disponibles en el mismo contenedor de Docker. Este enfoque significa que puede volver a invocar la función de Lambda sin necesidad de volver a realizar la implementación. Para los lenguajes compilados o los proyectos que requieran un soporte de empaquetado complejo, recomendamos que ejecute su propia solución de compilación y que AWS SAM incluya referencias al directorio que contiene los archivos de dependencias de compilación necesarios.

- **invoke:** invoca una función de Lambda local una vez y termina cuando se completa la invocación.

```
# 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:** genera eventos sin servidor simulados. Estos permiten desarrollar y probar localmente funciones que responden a eventos asíncronos, como los de Amazon S3, Kinesis y DynamoDB. A continuación se muestran las opciones de los comandos disponibles para la operación `generate-event`.

```
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:** valida la plantilla con la versión oficial de la especificación de [AWS Serverless Application Model](#). A continuación se muestra un ejemplo.

```
$ 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 y deploy: sam package y sam deploy realizan una llamada implícita a los comandos [package](#) y [deploy](#) de AWS CloudFormation. Para obtener más información sobre el empaquetado y la implementación de aplicaciones de SAM, consulte [Empaquetado e implementación \(p. 162\)](#).

A continuación se muestra cómo utilizar los comandos package y deploy en SAM Local.

```
# 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
```

Creación de una sencilla con SAM Local

Supongamos que desea crear una operación sencilla de la API RESTful, que crea, lee, actualiza y elimina una lista de productos. Comience creando la siguiente estructura de directorios:

dir/products.js

dir/template.yaml

El archivo template.yaml es la plantilla de AWS SAM que describe una sola función de Lambda que gestiona todas las solicitudes de la API.

Note

De forma predeterminada, los comandos start-api e invoke buscan el archivo template.yaml en el directorio de trabajo. Si hace referencia a un archivo template.yaml que se encuentra en otro directorio, añada el parámetro -t o --template a estas operaciones y pase una ruta absoluta o relativa a ese archivo.

Copie y pegue lo siguiente en el archivo template.yaml.

```
AWSTemplateFormatVersion : '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
```

El ejemplo anterior configura los siguientes puntos de enlace de la API RESTful:

- Crear un producto nuevo con una solicitud `PUT` para `/products`.
- Listar todos los productos con una solicitud `GET` para `/products`.
- Leer, actualizar o eliminar un producto con una solicitud `GET`, `PUT` o `DELETE` para `/products/{product}`.

A continuación, copie y pegue el siguiente código en el archivo `products.js`.

```
'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})

    }
}
```

Inicie una copia local de las operaciones de la API llamando al comando `start-api`.

```
$ 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.

A continuación, puede probar el punto de enlace de la API localmente mediante un navegador o mediante la CLI.

```
$ 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"
```

Para ver más ejemplos, consulte [aws sam local/samples](#).

Registro local

Los comandos `invoke` y `start-api` permiten canalizar a un archivo los logs generados por la invocación de la función de Lambda. Este enfoque resulta útil si ejecuta pruebas automatizadas con SAM Local y desea capturar los logs para su análisis. A continuación se muestra un ejemplo.

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

Mediante un archivo de variables de entorno

Si la función de Lambda utiliza [Variables de entorno \(p. 97\)](#), SAM Local proporciona un argumento `--env-vars` para los comandos `invoke` y `start-api`. Este argumento le permite utilizar un archivo JSON que contiene valores para las variables de entorno definidas en la función. La estructura del archivo JSON debe ser similar a la siguiente.

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

Seguidamente, puede acceder al archivo JSON utilizando el siguiente comando:

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

Mediante un entorno de shell

Las variables definidas en el entorno del shell se pasan al contenedor de Docker si se mapean a una variable de la función de Lambda. Todas las funciones pueden acceder globalmente a las variables del shell. Por ejemplo, suponga que tiene dos funciones, `MyFunction1` y `MyFunction2`, que tienen una variable denominada `TABLE_NAME`. En este caso, el valor de `TABLE_NAME` proporcionado a través del entorno del shell está disponible para ambas funciones.

El siguiente comando establece el valor de `TABLE_NAME` en `myTable` para ambas funciones.

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

Note

Para una mayor flexibilidad, puede utilizar una combinación de variables del shell y un archivo JSON externo que contenga las variables de entorno. Si una variable se define en los dos sitios, la del archivo externo tiene prioridad sobre la versión del shell. A continuación se indica el orden de prioridad, mayor a menor:

- Archivo de variables de entorno
- Entorno del shell
- Valores fijos contenidos en la plantilla de SAM

Depuración con SAM Local

Tanto `sam local invoke` como `sam local start-api` admiten la depuración local de funciones. Para ejecutar SAM Local con el soporte de depuración activado, especifique `--debug-port` o `-d` en la línea de comandos.

```
# 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

Si utiliza `sam local start-api`, la API Gateway local expone todas las funciones de Lambda. Sin embargo, debido a que solo se puede especificar un puerto de depuración, las funciones solo se pueden depurar de una en una.

Depuración de funciones escritas en Python

A diferencia de Node.js o Java, Python requiere la activación de la depuración remota en el código de la función de Lambda. Si activa la depuración (utilizando las opciones `--debug-port` o `-d` mencionadas anteriormente) para una función que utiliza uno de los tiempos de ejecución de Python (2.7 o 3.6), SAM Local realiza el mapeo desde la máquina host al contenedor de Lambda a través de ese puerto. Para activar la depuración remota, utilizar un paquete de Python como `remote-pdb`.

Important

Al configurar el host, el depurador escucha el código, por lo que debe utilizar `0.0.0.0` y no `127.0.0.1`.

Solución de problemas de aplicaciones basadas en Lambda

Una aplicación basada en Lambda típica consta de una o varias funciones activadas por eventos, como cargas de objetos a Amazon S3, notificaciones de Amazon SNS y acciones de la API. Una vez activadas, dichas funciones suelen llamar a recursos posteriores, como, por ejemplo, tablas de DynamoDB o buckets de Amazon S3, o realizar otras llamadas a la API. AWS Lambda utiliza Amazon CloudWatch para emitir automáticamente métricas y logs para todas las invocaciones de la función. Sin embargo, es posible que este mecanismo no sea conveniente para rastrear el origen del evento que invocó la función de Lambda, o para rastrear las llamadas posteriores realizadas por la función. Si desea ver una descripción general completa del funcionamiento del rastreo, consulte [AWS X-Ray](#).

Rastreo de aplicaciones basadas en Lambda con AWS X-Ray

AWS X-Ray es un servicio de AWS que permite detectar, analizar y optimizar problemas de desempeño en las aplicaciones de AWS Lambda. X-Ray recopila metadatos del servicio de Lambda y de cualquier servicio anterior o posterior que forme parte de la aplicación. X-Ray utiliza estos metadatos para generar un gráfico de servicios detallado que ilustra los cuellos de botella de desempeño, picos de latencia y otros problemas que afectan al desempeño de una aplicación de Lambda.

Después de utilizar el [Mapa de servicio de Lambda en AWS X-Ray \(p. 179\)](#) para identificar un recurso o componente problemático, puede ampliar el nivel de detalle y ver una representación visual de la solicitud. Esta representación visual cubre el intervalo de tiempo desde que un origen de eventos activa una función de Lambda hasta que se completa la ejecución de la función. X-Ray proporciona un desglose de las operaciones de la función, como, por ejemplo, información sobre las llamadas posteriores que la función de Lambda ha realizado a otros servicios. Además, la integración de X-Ray con Lambda proporciona visibilidad de la sobrecarga de servicios de AWS Lambda. Lo hace mostrando detalles como el tiempo de permanencia de la solicitud y el número de invocaciones.

Note

Los servicios que se integran actualmente con X-Ray son los únicos que se muestran como rastros independientes, ajenos al rastro de Lambda. Para obtener una lista de los servicios que admiten X-Ray actualmente, consulte [Integración de AWS X-Ray con los servicios de AWS](#).

Configuración de AWS X-Ray con Lambda

A continuación, puede encontrar información detallada sobre cómo configurar X-Ray con Lambda.

Antes de empezar

Para activar el rastreo en una función de Lambda mediante la CLI de Lambda, primero debe añadir permisos de rastreo al rol de ejecución de la función. Para ello, siga estos pasos:

- Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.
- Busque el rol de ejecución de la función de Lambda.
- Asocie la siguiente política administrada: `AWSXrayWriteOnlyAccess`.

Para obtener más información sobre estas políticas, consulte [AWS X-Ray](#).

Si cambia el modo de rastreo a activo utilizando la consola de Lambda, los permisos de rastreo se añaden automáticamente, como se explica en la siguiente sección.

Tracing

La ruta que recorre una solicitud a través de la aplicación se rastrea mediante un ID de rastro. Un rastro recopila todos los segmentos generados por una única solicitud, por lo general, una solicitud GET o POST de HTTP.

Existen dos modos de rastreo para una función de Lambda:

- Pass Through: es el valor predeterminado para todas las funciones de Lambda si ha añadido permisos de rastreo al rol de ejecución de la función. Este enfoque significa que la función de Lambda solo se rastrea si X-Ray se ha activado en un servicio ascendente, como AWS Elastic Beanstalk.
- Active: cuando una función de Lambda tiene esta configuración, Lambda muestrea automáticamente las solicitudes de invocación, basándose en el algoritmo de muestreo especificado por X-Ray.

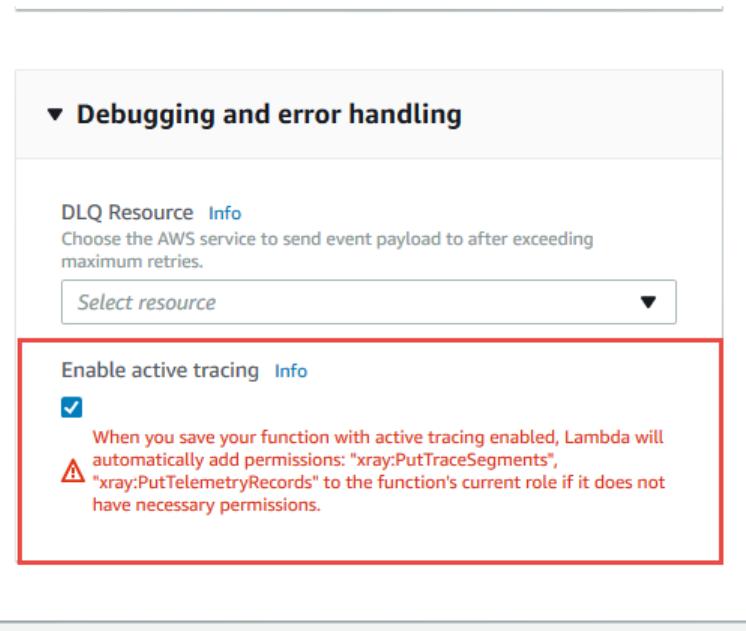
Note

X-Ray aplica un algoritmo de muestreo para garantizar que el rastreo sea eficiente, al tiempo que proporciona una muestra representativa de las solicitudes a las que sirve la aplicación. El algoritmo de muestreo predeterminado es de 1 solicitud por minuto, con un 5 por ciento de las solicitudes muestreadas pasado ese límite. Sin embargo, si el volumen de tráfico a la función es bajo, es posible que observe un aumento de la velocidad de muestreo.

Puede cambiar el modo de rastreo de una función de Lambda mediante la consola de administración de Lambda o las acciones de la API [CreateFunction \(p. 385\)](#) o [UpdateFunctionConfiguration \(p. 472\)](#) de Lambda.

Si utiliza la consola de Lambda, tenga en cuenta lo siguiente:

- Cuando se cambia el modo de seguimiento de una función a active, los permisos de seguimiento se asocian automáticamente al rol de ejecución de la función. Si recibe un error que indica que Lambda no ha podido añadir la política AWSXrayWriteOnlyAccess al rol de ejecución de la función, inicie sesión en la consola de IAM en <https://console.aws.amazon.com/iam/> y añada manualmente la política.
- Para activar el rastreo activo, vaya a la pestaña Configuration de la función y seleccione la casilla Enable active tracing.



Si utiliza las acciones [CreateFunction \(p. 385\)](#) o [UpdateFunctionConfiguration \(p. 472\)](#) de la API de Lambda:

- Si desea que el modo de rastreo esté activo, establezca la propiedad Mode del parámetro TracingConfig en Active. Recuerde que, de forma predeterminada, cualquier función nueva tiene el modo de rastreo configurado como PassThrough.
- Cualquier función de Lambda nueva o actualizada tiene la versión \$LATEST establecida en el valor que se especifique.

Note

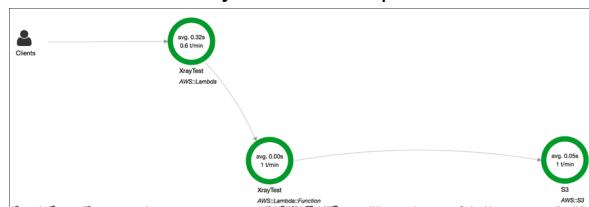
Se mostrará un error si no se han añadido permisos de rastreo al rol de ejecución de la función. Para obtener más información, consulte [Antes de empezar \(p. 177\)](#).

Mapa de servicio de Lambda en AWS X-Ray

X-Ray muestra tres tipos de nodos en el mapa de servicio para las solicitudes atendidas por Lambda:

- Servicio de Lambda (AWS::Lambda): este tipo de nodo representa el tiempo que la solicitud ha pasado en el servicio de Lambda. El tiempo comienza a contar cuando Lambda recibe la solicitud por primera vez, y finaliza cuando la solicitud sale del servicio de Lambda.
- Función de Lambda (AWS::Lambda::Function): este tipo de nodo representa el tiempo de ejecución de la función de Lambda.
- Llamadas de servicio posteriores: en este tipo, cada llamada de servicio posterior desde la función de Lambda se representa mediante su propio nodo.

En el diagrama siguiente, los nodos representan (de izquierda a derecha): el servicio de Lambda, la función del usuario y una llamada posterior a Amazon S3:



Para obtener más información, consulte [Uso del mapa de servicio](#).

Lambda como rastro de AWS X-Ray

Desde el mapa de servicio, puede ampliar el nivel de detalle para ver una vista de rastros de la función de Lambda. El rastro mostrará información detallada sobre las invocaciones de la función, representada como segmentos y subsegmentos:

- Segmento de servicio de Lambda: la información que representa este segmento depende del origen de eventos utilizado para invocar la función:
 - Orígenes de eventos síncronos y de flujo: el segmento de servicio mide el tiempo a partir del momento en que el servicio de Lambda recibe la solicitud o el evento, y finaliza cuando la solicitud abandona el servicio de Lambda (después de que se haya completado la invocación final de la solicitud).
 - Asíncronos: el segmento de servicio representa el tiempo de respuesta, es decir, el tiempo que tardó el servicio de Lambda en devolver una respuesta 202 al cliente.

El segmento de servicio de Lambda puede incluir dos tipos de subsegmentos:

- Tiempo de permanencia (solo para las invocaciones asíncronas): representa el tiempo que pasa la función en el servicio de Lambda antes de su invocación. Este subsegmento comienza cuando el servicio de Lambda recibe la solicitud o el evento y finaliza cuando se invoca la función de Lambda por primera vez.
- Intento: representa un solo intento de invocación, incluyendo cualquier sobrecarga introducida por el servicio de Lambda. Algunos ejemplos de sobrecarga son el tiempo dedicado a inicializar el código de la función y el tiempo de ejecución de la función.
- Segmento de la función de Lambda: representa el tiempo de ejecución de la función para un intento de invocación determinado. Se inicia cuando el controlador de la función comienza a ejecutarse y finaliza cuando la función termina. Este segmento puede incluir tres tipos de subsegmentos:
 - Inicialización: el tiempo empleado en ejecutar el código `initialization` de la función, que es el código que no pertenece al controlador de la función de Lambda ni a los inicializadores estáticos.
 - Llamadas posteriores: llamadas realizadas a otros servicios de AWS desde el código de la función de Lambda.

- Subsegmentos personalizados: anotaciones del usuario o subsegmentos personalizados que pueden añadirse al segmento de la función de Lambda utilizando el SDK de X-Ray.

Note

Para cada invocación rastreada, Lambda emite el segmento de servicio de Lambda y todos sus subsegmentos. Estos segmentos se emiten independientemente del tiempo de ejecución y no necesitan cambios en el código.

Emisión de segmentos de rastro desde una función de Lambda

Para cada invocación rastreada, Lambda emitirá el segmento de servicio de Lambda y todos sus subsegmentos. Además, Lambda emitirá el segmento de la función de Lambda y el subsegmento de inicialización. Estos segmentos se emitirán independientemente del tiempo de ejecución de la función y sin que se necesiten otras bibliotecas ni cambios en el código. Si desea que los rastros de X-Ray de la función de Lambda incluyan segmentos personalizados, anotaciones o subsegmentos para las llamadas posteriores, es posible que necesite incluir otras bibliotecas y anotar el código. Tenga en cuenta que cualquier instrumentación debe implementarse dentro del controlador de la función de Lambda, y no como parte del código de inicialización. Solo puede anotar los subsegmentos que haya creado, y no el segmento raíz de la función de Lambda.

Note

En la actualidad, estas funciones adicionales (segmentos personalizados, anotaciones y subsegmentos para las llamadas posteriores) están disponibles para los tiempos de ejecución de Node.js, Python y Java. Tenga en cuenta que cualquier instrumentación debe implementarse dentro del controlador de la función de Lambda, y no como parte del código de inicialización.

Node.js

En Node.js, puede hacer que Lambda emita subsegmentos a X-Ray para mostrar información acerca de las llamadas posteriores a otros servicios de AWS realizadas por la función. Para ello, primero debe incluir el [AWS X-Ray SDK for Node.js](#) en el paquete de implementación. Además, encapsule la instrucción `require` del SDK de AWS como se indica a continuación:

```
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));
```

A continuación, utilice la variable de AWS definida en el ejemplo anterior para inicializar cualquier cliente del servicio que desee rastrear con X-Ray, por ejemplo:

```
s3Client = AWS.S3();
```

Después de seguir estos pasos, cualquier llamada realizada desde la función utilizando `s3Client` da como resultado un subsegmento de X-Ray que representa esa llamada. Por ejemplo, puede ejecutar la función de Node.js siguiente para ver el rastro en X-Ray:

```
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));
```

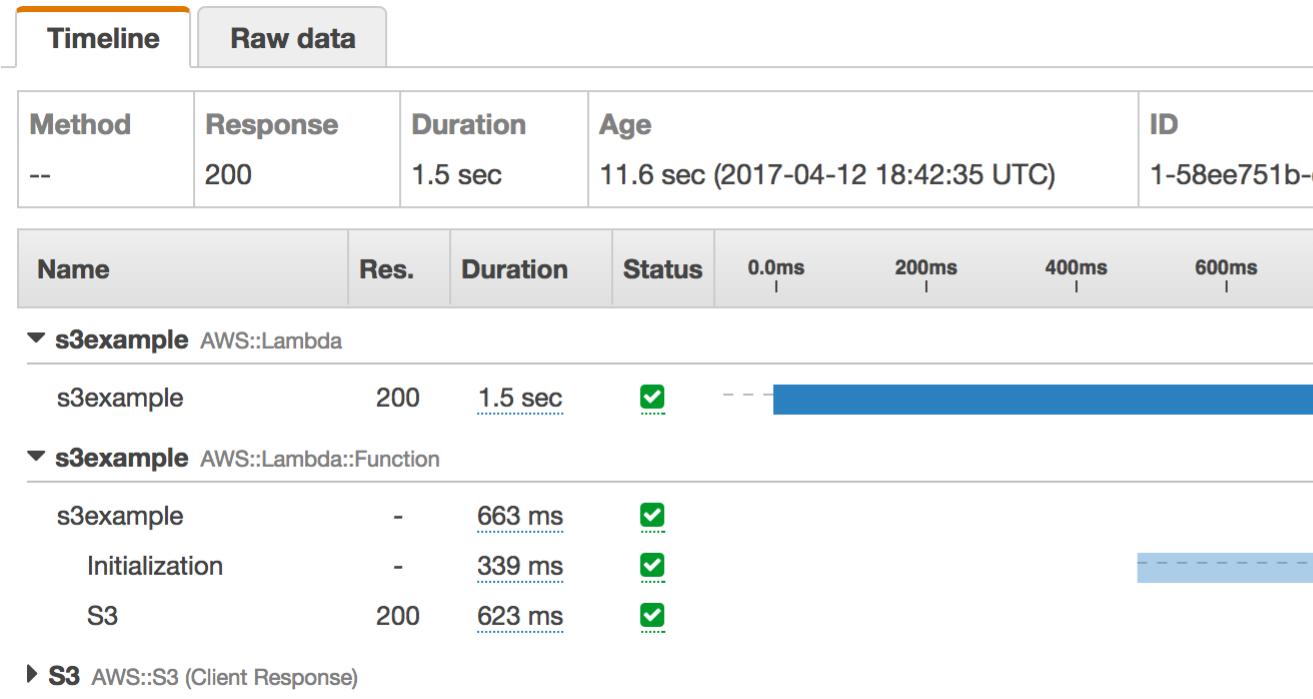
```
s3 = new AWS.S3({signatureVersion: 'v4'});

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!')
        }
    });
};
```

A continuación se muestra cómo se visualiza un rastro emitido por el código anterior (invocación síncrona):



Java

En Java, puede hacer que Lambda emita subsegmentos a X-Ray para mostrar información relacionada con las llamadas posteriores a otros servicios de AWS realizadas por la función. Para aprovechar esta función, incluya el [AWS X-Ray SDK para Java](#) en el paquete de implementación. No es necesario cambiar el código. Siempre que esté utilizando la versión 1.11.48 y anteriores del SDK de AWS, no es necesario que añada líneas de código adicionales para las llamadas posteriores desde la función que se va a rastrear.

El SDK de AWS importará dinámicamente el SDK de X-Ray para emitir subsegmentos para las llamadas posteriores realizadas por la función. Si utiliza el SDK de X-Ray para Java, puede instrumentar el código para que emita subsegmentos personalizados o añada anotaciones a los segmentos de X-Ray.

En el siguiente ejemplo se utiliza el SDK de X-Ray para Java con objeto de instrumentar una función de Lambda para emitir un subsegmento personalizado y enviar anotaciones personalizadas a 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);

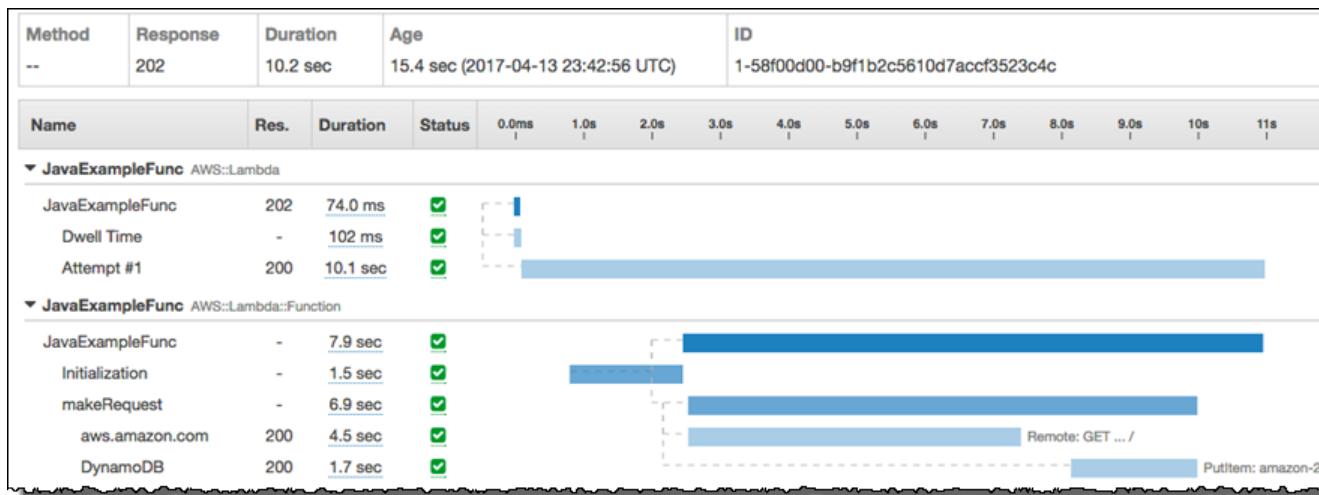
    private static final AmazonDynamoDB dynamoClient;
    private static final HttpClient httpClient;

    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);
        });
    }
}
```

A continuación se muestra cómo se visualiza un rastro emitido por el código anterior (invocación síncrona):



Python

En Python, puede hacer que Lambda emita subsegmentos a X-Ray para mostrar información acerca de las llamadas posteriores a otros servicios de AWS realizadas por la función. Para ello, primero debe incluir el [AWS X-Ray SDK for Python](#) en el paquete de implementación. Además, puede aplicar parches a boto3 (o botocore si está usando sesiones), de modo que cualquier cliente que cree para obtener acceso a otros servicios de AWS será rastreado automáticamente por X-Ray.

```
import boto3
from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.core import patch

patch(['boto3'])
```

Una vez que haya aplicado el parche al módulo que está utilizando para crear clientes, puede usarlo para crear clientes rastreados, como Amazon S3 en el ejemplo siguiente:

```
s3_client = boto3.client('s3')
```

El SDK de X-Ray para Python crea un subsegmento para la llamada y registra la información de la solicitud y la respuesta. Puede utilizar `aws_xray_sdk.core.xray_recorder` para crear subsegmentos automáticamente, modificando las funciones de Lambda, o manualmente, llamando a `xray_recorder.begin_subsegment()` y `xray_recorder.end_subsegment()` dentro de la función, tal como se muestra en la siguiente función de Lambda.

```
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)
```

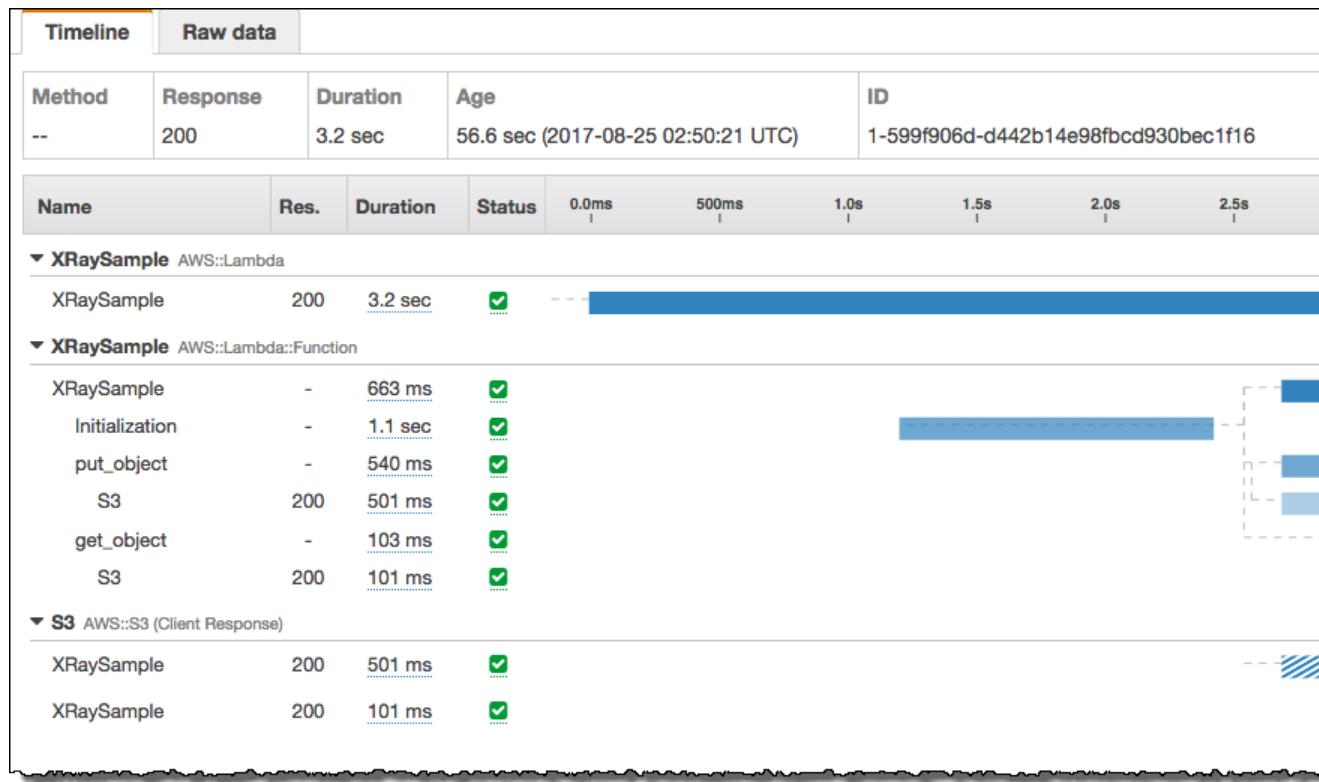
Note

El SDK de X-Ray para Python le permite aplicar parches a los siguientes módulos:

- botocore
- boto3
- requests
- sqlite3
- mysql

Puede utilizar `patch_all()` para aplicar parches a todos a la vez.

A continuación se muestra cómo se visualiza un rastro emitido por el código anterior (invocación síncrona):



El demonio de AWS X-Ray en el entorno de Lambda

El demonio de AWS X-Ray es una aplicación de software que recopila datos de los segmentos sin procesar y se los transmite al servicio de AWS X-Ray. El demonio funciona junto con los SDK de AWS X-Ray para que los datos que envíen los SDK puedan llegar al servicio de X-Ray.

Cuando se rastrea una función de Lambda, el demonio de X-Ray se ejecuta automáticamente en el entorno de Lambda para recopilar datos de rastreo y enviarlos a X-Ray. Cuando se realiza el rastreo, el demonio de X-Ray consume un máximo de 16 MB o el 3 por ciento de la asignación de memoria de la función. Por ejemplo, si asigna 128 MB de memoria a la función de Lambda, el demonio de X-Ray obtiene 16 MB de la asignación de memoria de la función. Si asigna 1 024 MB a la función de Lambda, el demonio de X-Ray tiene asignados 31 MB (el 3 por ciento). Para obtener más información, consulte [El demonio de AWS X-Ray](#).

Note

Lambda intentará terminar el demonio de X-Ray para evitar que se supere la cantidad máxima de memoria de la función. Por ejemplo, supongamos que ha asignado 128 MB a la función de Lambda, lo que significa que el demonio de X-Ray tendrá asignados 16 MB. Esto deja a la función de Lambda con una asignación de memoria de 112 MB. Sin embargo, si la función supera los 112 MB, el demonio de X-Ray se terminará para evitar que se produzca un error de memoria.

Uso de variables de entorno para comunicarse con AWS X-Ray

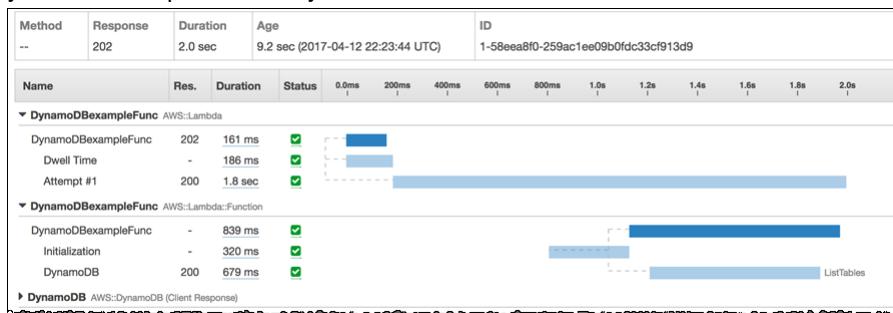
AWS Lambda genera automáticamente tres variables de entorno para facilitar la comunicación con el demonio de X-Ray y definir la configuración del SDK de X-Ray:

- `_X_AMZN_TRACE_ID`: contiene el encabezado de rastreo, que incluye la decisión de muestreo, el ID de rastro y el ID del segmento principal. (Para obtener más información acerca de estas propiedades, consulte [Encabezado de rastreo](#)). Si Lambda recibe un encabezado de rastreo cuando se invoca la función, ese encabezado se utilizará para llenar la variable de entorno `_X_AMZN_TRACE_ID`. Si no se ha recibido un encabezado de rastreo, Lambda generará uno automáticamente.
- `AWS_XRAY_CONTEXT_MISSING`: el SDK de X-Ray utiliza esta variable para determinar su comportamiento en caso de que la función intente registrar datos de X-Ray, pero no haya un encabezado de rastreo de datos disponible. Lambda establece este valor como `LOG_ERROR` de forma predeterminada.
- `AWS_XRAY_DAEMON_ADDRESS`: esta variable de entorno expone la dirección del demonio de X-Ray con el siguiente formato: `DIRECCIÓN_IP:PUERTO`. Puede utilizar la dirección del demonio de X-Ray para enviar datos de rastreo al demonio de X-Ray directamente, sin usar el SDK de X-Ray.

Ejemplos de rastros de Lambda en la consola de AWS X-Ray

A continuación se muestran rastros de Lambda para dos funciones de Lambda distintas. Cada rastro muestra una estructura de rastro para un tipo de invocación: asíncrona y síncrona.

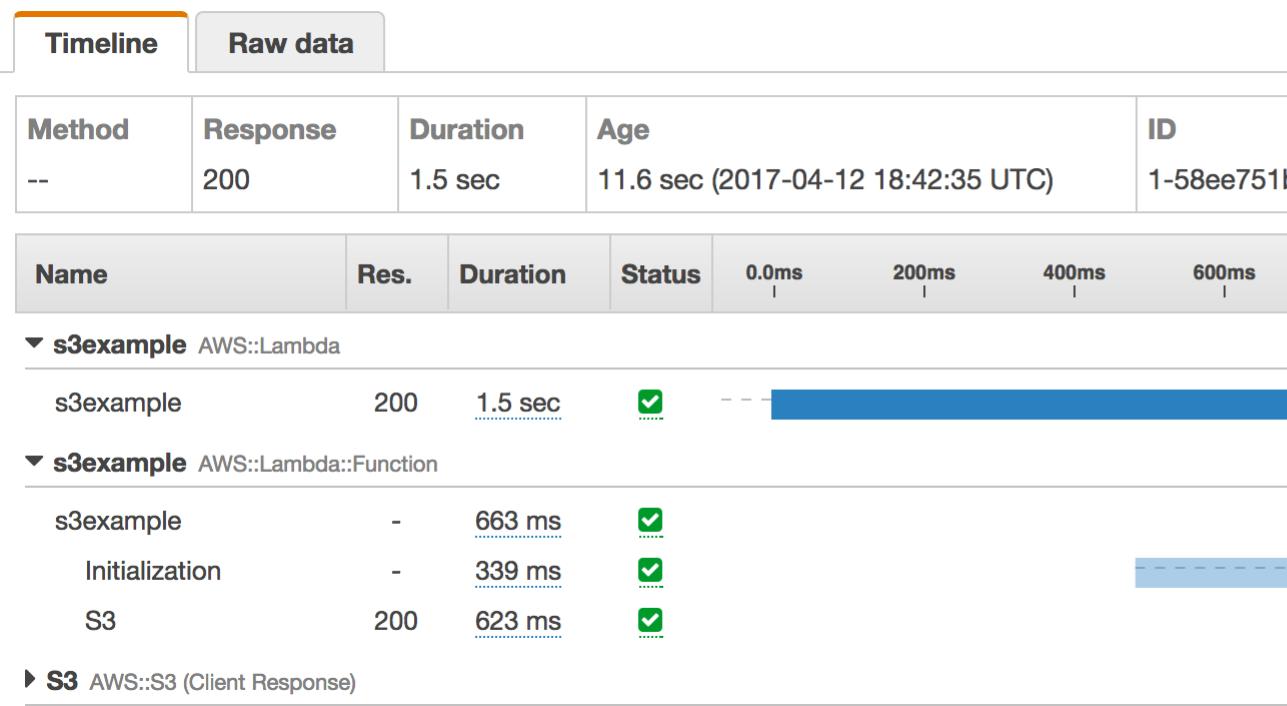
- Asíncrona: el siguiente ejemplo muestra una solicitud de Lambda asíncrona con una invocación correcta y una llamada posterior a DynamoDB.



El segmento de servicio de Lambda encapsula el tiempo de respuesta, que es el tiempo que tarda en devolver una respuesta (por ejemplo, 202) al cliente. Incluye subsegmentos para el tiempo empleado en la cola del servicio de Lambda (tiempo de permanencia) y en cada intento de invocación. (En el ejemplo anterior, solo aparece un intento de invocación). Cada subsegmento de intento del segmento de servicio tendrá su correspondiente segmento de función del usuario. En este ejemplo, el segmento de función del usuario contiene dos subsegmentos: el subsegmento de inicialización, que representa el código de inicialización de la función que se ejecuta antes del controlador, y un subsegmento de llamada posterior que representa una llamada `ListTables` a DynamoDB.

Se muestran los códigos de estado y los mensajes de error para cada subsegmento de invocación y cada llamada posterior.

- Síncrona: el ejemplo siguiente muestra una solicitud síncrona con una llamada posterior a Amazon S3.



El segmento de servicio de Lambda captura todo el tiempo que la solicitud pasa en el servicio de Lambda. El segmento de servicio tendrá su correspondiente segmento de función del usuario. En este ejemplo, el segmento de función del usuario contiene un subsegmento que representa el código de inicialización de la función (el código que se ejecuta antes del controlador) y un subsegmento que representa la llamada PutObject a Amazon S3.

Note

Si desea rastrear las llamadas HTTP, debe utilizar un cliente HTTP. Para obtener más información, consulte [Rastreo de llamadas a servicios web HTTP posteriores con X-Ray SDK for Java](#) o [Rastreo de llamadas a servicios web HTTP posteriores utilizando X-Ray SDK for Node.js](#).

AWS Lambda: funcionamiento

¿Cómo ejecuta AWS Lambda mi código? El modelo de contenedores

Cuando AWS Lambda ejecuta una función de Lambda en su nombre, se encarga de aprovisionar y administrar los recursos necesarios para ejecutarla. Al crear una función de Lambda, se debe especificar información de configuración, como la cantidad de memoria y el tiempo máximo de ejecución asignados a la función. Al invocar una función de Lambda, AWS Lambda lanza un contenedor (es decir, un entorno de ejecución) en función de las opciones de configuración proporcionadas.

Note

El contenido de esta sección es meramente informativo. AWS Lambda se encarga de administrar la creación y eliminación de contenedores; no existe ninguna API de AWS Lambda que permita al usuario administrar contenedores.

Se necesita tiempo para configurar un contenedor y realizar el proceso de arranque necesario, lo que añade cierta latencia cada vez que se invoca una función de Lambda. Normalmente, esta latencia aparece cuando se invoca una función de Lambda por primera vez o después de actualizarla, ya que AWS Lambda intenta reutilizar el contenedor para las invocaciones posteriores de la función.

Una vez ejecutada la función de Lambda, AWS Lambda mantiene el contenedor durante cierto tiempo, anticipándose a otra posible invocación de la función. A efectos prácticos, el servicio congela el contenedor una vez finalizada la función de Lambda y, si AWS Lambda decide reutilizarlo cuando se invoque de nuevo la función, lo descongela para su reutilización. Este enfoque de reutilización de contenedores tiene las siguientes implicaciones:

- Las declaraciones del código de la función de Lambda (aparte del código `handler`, consulte [Modelo de programación \(p. 8\)](#)) permanecen inicializadas, proporcionando una optimización adicional al invocar de nuevo la función. Por ejemplo, si la función de Lambda establece una conexión con una base de datos, en lugar de volver a establecer la conexión, se utiliza la conexión original en posteriores invocaciones. Puede añadir lógica en el código que compruebe si ya existe una conexión antes de crear una nueva.
- Cada contenedor proporciona 500 MB de espacio en disco adicional en el directorio `/tmp`. El contenido del directorio se conserva al congelar el contenedor, proporcionando una caché transitoria que se puede utilizar para varias invocaciones. Puede agregar código adicional para comprobar si la caché dispone de los datos que se almacenaron. Para obtener información acerca de los límites de implementación, consulte [Límites de AWS Lambda \(p. 335\)](#).
- Los procesos en segundo plano o las devoluciones de llamada iniciadas por la función de Lambda que no se completaron al finalizar la función se reanudan si AWS Lambda decide reutilizar el contenedor. Asegúrese de que los procesos en segundo plano o las devoluciones de llamada (en el caso de Node.js) del código se completen antes de que este finalice.

Note

Al escribir el código de la función de Lambda no presuponga que AWS Lambda siempre reutiliza el contenedor, porque es posible que no sea así. Dependiendo de distintos factores, AWS Lambda puede simplemente crear un nuevo contenedor en lugar de reutilizar uno ya existente.

Ejecuciones simultáneas de la función de Lambda

Las ejecuciones simultáneas hacen referencia al número de ejecuciones del código de la función que tienen lugar en un momento dado. Es posible hacer un cálculo aproximado del número de ejecuciones simultáneas, pero esta cifra variará en función de si la función de Lambda está procesando o no eventos de un origen de eventos basado en flujos.

- **Orígenes de eventos basados en flujos:** si crea una función de Lambda que procesa eventos de servicios basados en flujos (flujos de Amazon Kinesis Data Streams o de DynamoDB), el número de fragmentos por flujo es la unidad de concurrencia. Si el flujo tiene 100 fragmentos activos, habrá como máximo 100 invocaciones a la función de Lambda ejecutándose simultáneamente. A continuación, cada función de Lambda procesa eventos de un fragmento en el orden en el que llegan.
- **Orígenes de eventos que no están basados en flujos:** si crea una función de Lambda para procesar eventos provenientes de orígenes de eventos que no están basados en flujos (por ejemplo, Amazon S3 o API Gateway), cada uno de los eventos publicados es una unidad de trabajo. Por lo tanto, el número de eventos (o solicitudes) que estos orígenes de eventos publican influye en la concurrencia.

Puede utilizar la siguiente fórmula para calcular el número aproximado de invocaciones simultáneas a la función de Lambda:

```
events (or requests) per second * function duration
```

Por ejemplo, imagine una función de Lambda que procesa eventos de Amazon S3. Supongamos que la función de Lambda tarda en ejecutarse un promedio de tres segundos y que Amazon S3 publica 10 eventos por segundo. Como resultado, dispondrá de 30 ejecuciones simultáneas de la función de Lambda.

Velocidad de solicitudes

La velocidad de solicitudes hace referencia a la velocidad a la que se invoca la función de Lambda. Para todos los servicios, a excepción de los servicios basados en flujos, la velocidad de solicitudes es la velocidad a la que los orígenes de eventos generan los eventos. Para los servicios basados en flujos, AWS Lambda calcula dicha velocidad como se indica a continuación:

```
request rate = number of concurrent executions / function duration
```

Por ejemplo, si hay cinco fragmentos activos en un flujo (es decir, hay cinco funciones de Lambda ejecutándose en paralelo) y la función de Lambda tarda alrededor de dos segundos, la velocidad de solicitudes es de 2,5 solicitudes por segundo.

Límite de ejecuciones simultáneas

De forma predeterminada, AWS Lambda limita a 1 000 el número total de ejecuciones simultáneas en todas las funciones de una región determinada. Para obtener instrucciones acerca de cómo aumentar ese límite, siga leyendo.

Las invocaciones que provocan que la ejecución simultánea de la función supere el límite de seguridad se limitan de forma controlada. En este caso, la invocación no ejecuta la función. Cada una de las invocaciones limitadas aumenta la métrica `Throttles` de Amazon CloudWatch para la función.

La invocación limitada se gestiona de forma distinta en función de cómo se invoque la función:

- Orígenes de eventos que no están basados en flujos: algunos de estos orígenes de eventos invocan una función de Lambda de forma síncrona, y otros la invocan de forma asíncrona. La forma de gestionarlos es diferente para cada una de ellas:
 - Invocación síncrona: si la función se invoca de forma síncrona y se limita de forma controlada, la aplicación que la invoca recibe un error 429 y es responsable de los reintentos. Estos orígenes de eventos pueden tener reintentos adicionales incluidos en la integración. Por ejemplo, CloudWatch Logs reintenta el lote con errores hasta cinco veces, con retrasos entre los reintentos. Para obtener una lista de los orígenes de eventos admitidos y los tipos de invocación que utilizan, consulte [Orígenes de eventos admitidos \(p. 138\)](#).

Si invoca una función de Lambda a través de Amazon API Gateway, asegúrese de mapear los errores de respuesta de Lambda a códigos de error de API Gateway. Si invoca la función directamente, por ejemplo, a través de los SDK de AWS mediante el modo de invocación `RequestResponse` o a través de API Gateway, el cliente recibe el error 429 y es posible reintentar la invocación.

- Invocación asíncrona: si la función de Lambda se invoca de forma asíncrona y se limita de forma controlada, AWS Lambda reintenta automáticamente el evento limitado durante un máximo de seis horas, con retrasos entre los reintentos. Los eventos asíncronos se ponen en cola antes de que se utilicen para invocar la función de Lambda.
- Orígenes de eventos basados en flujos: en los orígenes de eventos basados en flujos (flujos de Amazon Kinesis Data Streams y Amazon DynamoDB), AWS Lambda sondea el flujo e invoca la función de Lambda. Cuando la función de Lambda se limita de forma controlada, AWS Lambda intenta procesar el lote de registros limitado hasta el momento en que los datos caducan. Este periodo de tiempo puede ser de hasta siete días para Amazon Kinesis Data Streams. La solicitud limitada se trata como un bloqueo por fragmento, y Lambda no lee ningún registro nuevo del fragmento hasta que el lote de registros limitado caduque o se procese correctamente. Si existe más de un fragmento en el flujo, Lambda continúa invocando los fragmentos no limitados hasta que uno de ellos consiga procesarse.

Para solicitar un aumento del límite de ejecuciones simultáneas

1. Abra la página [AWS Support Center](#), inicie sesión si es necesario y, a continuación, elija `Create case`.
2. En `Regarding`, seleccione `Service Limit Increase`.
3. En `Limit Type`, elija `Lambda`, rellene los campos necesarios en el formulario y elija el botón correspondiente al método de contacto que prefiera en la parte inferior de la página.

Note

AWS puede aumentar automáticamente el límite de ejecuciones simultáneas para permitir que la función se adapte a la tasa de eventos entrantes. Un ejemplo es el caso en el que la función se activa desde un bucket de Amazon S3.

Escalado

AWS Lambda escalará dinámicamente la capacidad en respuesta al aumento del tráfico, de acuerdo con el [Límite de ejecuciones simultáneas \(p. 189\)](#) de su cuenta. Para gestionar cualquier ráfaga de aumento de tráfico, Lambda aumentará inmediatamente las funciones que se están ejecutando simultáneamente en una cantidad predeterminada, en función de la región en la que se ejecuta (consulte la tabla siguiente).

Si el valor predeterminado de Aumento de concurrencia inmediato que se indica en la tabla siguiente no es suficiente para satisfacer la oleada de tráfico, Lambda seguirá aumentando el número de ejecuciones simultáneas de la función en 500 por minuto hasta alcanzar el límite de seguridad de la cuenta o hasta que el número de funciones que se ejecutan simultáneamente sea suficiente para procesar correctamente el incremento de carga.

Note

Dado que Lambda depende de Amazon EC2 para proporcionar interfaces de red elásticas para las funciones de Lambda habilitadas para VPC, estas funciones también están sujetas a los límites de velocidad de Amazon EC2 a medida que se escalan. Si los límites de velocidad de Amazon EC2 impiden que las funciones habilitadas para VPC añadan 500 invocaciones simultáneas por minuto, solicite un aumento del límite siguiendo las instrucciones de la sección anterior en [Para solicitar un aumento del límite de ejecuciones simultáneas](#).

Más allá de esta velocidad (es decir, para las aplicaciones que utilizan todo el aumento de concurrencia inmediato), la aplicación debe ser capaz de gestionar la limitación controlada de Amazon EC2 (502 EC2ThrottledException) a través del reintento y el retardo del lado del cliente. Para obtener más información, consulte [Reintentos de error y retardo exponencial en AWS](#).

En la tabla siguiente se describe el aumento de concurrencia inmediato por región:

| Región | Aumento de concurrencia inmediato (ejecuciones de funciones) |
|------------------------------------|--|
| Asia Pacífico (Tokio) | 1 000 |
| Asia Pacífico (Seúl) | 500 |
| Asia Pacífico (Mumbai) | 500 |
| Asia Pacífico (Singapur) | 500 |
| Asia Pacífico (Sídney) | 500 |
| Canadá (Central) | 500 |
| UE (Fráncfort) | 1 000 |
| UE (Londres) | 500 |
| UE (Irlanda) | 3 000 |
| AWS GovCloud (US) | 500 |
| EE.UU. Este (Ohio) | 500 |
| EE.UU. Oeste (Norte de California) | 500 |
| EE.UU. Oeste (Oregón) | 3 000 |
| US East (N. Virginia) | 3 000 |
| América del Sur (São Paulo) | 500 |

Lectura recomendada

Si es la primera vez que utiliza AWS Lambda, le sugerimos que lea todos los temas de la sección “Funcionamiento” para familiarizarse con Lambda. El siguiente tema es [Reintentos si se producen errores \(p. 192\)](#).

Después de leer todos los temas de la sección Funcionamiento, le recomendamos que lea [Creación de funciones de Lambda \(p. 5\)](#), que pruebe el ejercicio de [Introducción \(p. 199\)](#) y, a continuación, que examine los [Casos de uso \(p. 214\)](#). Cada caso de uso proporciona instrucciones paso a paso para configurar la experiencia integral.

Reintentos si se producen errores

Una función de Lambda puede presentar un error por cualquiera de las razones siguientes:

- El tiempo de espera de la función se agota al intentar alcanzar un punto de enlace.
- La función no consigue analizar correctamente los datos de entrada.
- La función experimenta restricciones de recursos, como errores de memoria insuficiente u otros tiempos de espera.

Si se produce cualquiera de estos errores, la función generará una excepción. El modo en que se gestiona la excepción depende de cómo se ha invocado la función de Lambda:

- Orígenes de eventos que no están basados en flujos: algunos de estos orígenes de eventos están configurados para invocar una función de Lambda de forma síncrona, y otros para invocarla de forma asíncrona. En consecuencia, las excepciones se gestionan de la siguiente manera:
 - Invocación síncrona: la aplicación que realiza la invocación recibe un error 429, y es responsable de los reintentos. Para obtener una lista de los orígenes de eventos admitidos y los tipos de invocación que utilizan, consulte [Orígenes de eventos admitidos](#). Estos orígenes de eventos pueden tener reintentos adicionales incluidos en la integración.

Si invocó la función de Lambda directamente a través de los SDK de AWS o a través de API Gateway, el cliente recibe el error y puede optar por volver a intentarlo. Si invoca una función de Lambda a través de API Gateway, debe asegurarse de mapear los errores de respuesta de Lambda a códigos de error de API Gateway.

- Invocación asíncrona: los eventos asíncronos se ponen en cola antes de utilizarlos para invocar la función de Lambda. Si AWS Lambda no puede procesar completamente el evento, reintentará automáticamente la invocación dos veces, con retrasos entre los reintentos. Si ha especificado una cola de mensajes fallidos para la función, el evento con error se envía a la cola de Amazon SQS o al tema de Amazon SNS especificados. Si no especifica una cola de mensajes fallidos (DLQ), lo que no es necesario y es la configuración predeterminada, el evento se descartará. Para obtener más información, consulte [Colas de mensajes fallidos \(p. 128\)](#).
- Orígenes de eventos basados en flujos: en los orígenes de eventos basados en flujos (flujos de Amazon Kinesis Data Streams y DynamoDB), AWS Lambda sondea el flujo e invoca la función de Lambda. Por lo tanto, si una función de Lambda falla, AWS Lambda intenta procesar el lote de registros con errores hasta que los datos caduquen, lo que puede tardar hasta siete días para Amazon Kinesis Data Streams. La excepción se trata como un bloqueo, y AWS Lambda no leerá ningún registro nuevo del flujo hasta que el lote de registros con errores caduque o se procese correctamente. Esto garantiza que AWS Lambda procesa los eventos de flujo por orden.

Para obtener más información acerca de los modos de invocación, consulte [Mapeo de orígenes de eventos \(p. 134\)](#).

Lectura recomendada

Si es la primera vez que utiliza AWS Lambda, le sugerimos que lea todos los temas de la sección “Funcionamiento” para familiarizarse con Lambda. El siguiente tema es [Modelo de permisos de AWS Lambda \(p. 193\)](#).

Después de leer todos los temas de la sección Funcionamiento, le recomendamos que lea [Creación de funciones de Lambda \(p. 5\)](#), que pruebe el ejercicio de [Introducción \(p. 199\)](#) y, a continuación, que examine los [Casos de uso \(p. 214\)](#). Cada caso de uso proporciona instrucciones paso a paso para configurar la experiencia integral.

Modelo de permisos de AWS Lambda

Para que las aplicaciones integrales basadas en AWS Lambda funcionen, es necesario administrar diferentes permisos. Por ejemplo:

- En el caso de los orígenes de eventos, excepto para los servicios basados en flujos (flujos de Amazon Kinesis Data Streams y de DynamoDB), debe conceder al origen de eventos permisos para invocar la función de AWS Lambda.
- En los orígenes de eventos basados en flujos (flujos de Amazon Kinesis Data Streams y de DynamoDB), AWS Lambda sondea los flujos en su nombre y lee nuevos registros en el flujo, por lo que es necesario que se le concedan permisos para realizar las acciones de flujo pertinentes.
- Cuando la función de Lambda se ejecuta, puede acceder a los recursos de AWS de su cuenta (por ejemplo, para leer un objeto del bucket de S3). AWS Lambda ejecuta la función de Lambda en su nombre asumiendo el rol que se especificó en el momento de crear la función de Lambda. Por lo tanto, debe conceder al rol los permisos que la función de Lambda necesita, por ejemplo, para que las acciones de Amazon S3 puedan leer un objeto.

En las secciones siguientes se describe la administración de permisos.

Temas

- [Administración de permisos mediante un rol de IAM \(rol de ejecución\) \(p. 193\)](#)
- [Administración de permisos mediante una política de función de Lambda \(p. 194\)](#)
- [Lectura recomendada \(p. 195\)](#)

Administración de permisos mediante un rol de IAM (rol de ejecución)

Cada función de Lambda dispone de un rol de IAM (rol de ejecución) asociado. Este rol se especifica al crear la función de Lambda. Los permisos que concede a este rol determinarán lo que AWS Lambda podrá hacer cuando asuma dicho rol. Existen dos tipos de permisos que se pueden conceder al rol de IAM:

- Si el código de la función de Lambda obtiene acceso a otros recursos de AWS, por ejemplo, para leer un objeto de un bucket de S3 o escribir registros en CloudWatch Logs, debe conceder permisos al rol para las acciones de Amazon S3 y CloudWatch pertinentes.
- Si el origen de eventos está basado en flujos (flujos de Amazon Kinesis Data Streams y de DynamoDB), AWS Lambda sondea estos flujos en su nombre. AWS Lambda necesita permisos para sondear el flujo

y leer nuevos registros en él, por lo que es necesario que se le concedan los permisos pertinentes a este rol.

Para obtener más información acerca de los roles de IAM, consulte [Roles de IAM](#) en la Guía del usuario de IAM.

Important

El usuario que crea el rol de IAM está, en realidad, pasando permisos a AWS Lambda para que asuma este rol, lo que requiere que el usuario tenga permisos para la acción `iam:PassRole`. Si un usuario administrador crea este rol, no necesita hacer nada más para configurar los permisos de la acción `iam:PassRole`, ya que dicho usuario tiene permisos completos, incluida la acción `iam:PassRole`.

Para simplificar el proceso de creación de un rol de ejecución, AWS Lambda le permite utilizar las siguientes políticas de permisos administradas por AWS (predefinidas). Estas políticas incluyen permisos comunes para escenarios específicos:

- **AWSLambdaBasicExecutionRole**: concede permisos a las acciones de Amazon CloudWatch Logs únicamente para escribir logs. Puede utilizar esta política si la función de Lambda no tiene acceso a ningún otro recurso de AWS, excepto la escritura de registros.
- **AWSLambdaKinesisExecutionRole**: concede permisos para las acciones de Amazon Kinesis Data Streams y de CloudWatch Logs. Si va a escribir una función de Lambda para procesar eventos de flujo de Kinesis, puede asociar esta política de permisos.
- **AWSLambdaDynamoDBExecutionRole**: concede permisos para las acciones de flujo de DynamoDB y las acciones de CloudWatch Logs. Si va a escribir una función de Lambda para procesar eventos de flujo de DynamoDB, puede asociar esta política de permisos.
- **AWSLambdaVPCAccessExecutionRole**: concede permisos a las acciones de Amazon Elastic Compute Cloud (Amazon EC2) para administrar interfaces de red elásticas (ENI). Si va a escribir una función de Lambda para obtener acceso a los recursos de una VPC en el servicio de Amazon Virtual Private Cloud (Amazon VPC), puede asociar esta política de permisos. La política también concede permisos a las acciones de CloudWatch Logs para escribir logs.

Puede encontrar estas políticas de permisos administradas de AWS en la consola de IAM. Busque estas políticas para ver los permisos que concede cada una de ellas.

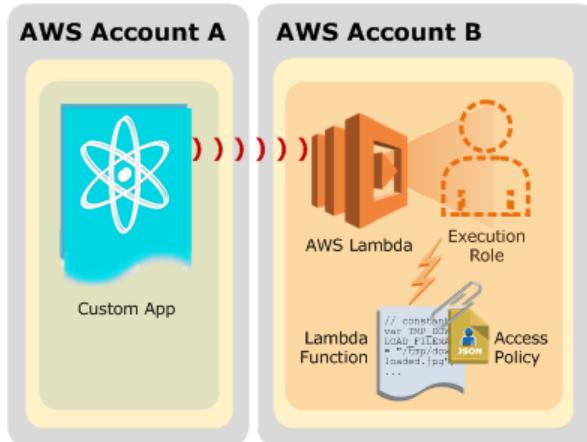
Administración de permisos mediante una política de función de Lambda

Todos los orígenes de eventos admitidos, excepto los servicios basados en flujos (flujos de Kinesis y de DynamoDB), invocan la función de Lambda (el modelo de inserción), siempre que dispongan de los permisos necesarios. Por ejemplo, si desea que Amazon S3 invoque la función de Lambda cuando se crean objetos en un bucket, Amazon S3 necesita permisos para hacerlo.

Puede conceder estos permisos mediante las políticas de funciones. AWS Lambda proporciona API que le permiten administrar permisos en una política de función. Por ejemplo, consulte [AddPermission \(p. 372\)](#).

También puede conceder permisos entre cuentas mediante la política de función. Por ejemplo, si una aplicación definida por el usuario y la función de Lambda a la que invoca pertenecen a la misma cuenta

de AWS, no es necesario conceder permisos explícitos. En caso contrario, la cuenta de AWS a la que pertenece la función de Lambda debe incluir permisos entre cuentas en la política de permisos asociada a la función de Lambda.



Note

En lugar de utilizar una política de función de Lambda, puede crear otro rol de IAM que conceda a los orígenes de eventos (por ejemplo, Amazon S3 o DynamoDB) permisos para invocar la función de Lambda. Sin embargo, es posible que las políticas de recursos resulten más sencillas de configurar y le faciliten el seguimiento de los orígenes de eventos que disponen de permisos para invocar la función de Lambda.

Para obtener más información acerca de las políticas de funciones de Lambda, consulte [Uso de políticas basadas en recursos para AWS Lambda \(políticas de funciones de Lambda\)](#) (p. 355). Para obtener más información acerca de los permisos de Lambda, consulte [Autenticación y control de acceso de AWS Lambda](#) (p. 337).

Lectura recomendada

Si es la primera vez que utiliza AWS Lambda, le sugerimos que lea todos los temas de la sección “Funcionamiento” para familiarizarse con Lambda. El siguiente tema es [Entorno de ejecución de Lambda y bibliotecas disponibles](#) (p. 195).

Después de leer todos los temas de la sección Funcionamiento, le recomendamos que lea [Creación de funciones de Lambda](#) (p. 5), que pruebe el ejercicio de [Introducción](#) (p. 199) y, a continuación, que examine los [Casos de uso](#) (p. 214). Cada caso de uso proporciona instrucciones paso a paso para configurar la experiencia integral.

Entorno de ejecución de Lambda y bibliotecas disponibles

El entorno de ejecución subyacente de AWS Lambda se basa en lo siguiente:

- La versión pública de AMI de Amazon Linux (nombre de AMI: amzn-ami-hvm-2017.03.1.20170812-x86_64-gp2) a la que se puede obtener acceso [aquí](#).

Para obtener información acerca del uso de una AMI, consulte [Imágenes de máquina de Amazon \(AMI\)](#) en la Guía del usuario de Amazon EC2 para instancias de Linux.

- Versión del kernel de Linux: 4.9.43-17.38.amzn1.x86_64

Si está utilizando archivos binarios nativos en el código, asegúrese de que se compilan en este entorno. Tenga en cuenta que AWS Lambda solo admite archivos binarios de 64 bits.

AWS Lambda admite las siguientes versiones del tiempo de ejecución:

- Node.js: v4.3.2 y 6.10.3
- Java: Java 8
- Python: Python 3.6 y 2.7
- .NET Core: .NET Core 1.0.1 (C#)

Note

No todos los tiempos de ejecución están disponibles en la versión pública de la AMI de Amazon Linux o en sus repositorios yum. Es posible que tenga que descargarlos e instalarlos manualmente desde sus respectivos sitios públicos.

Las siguientes bibliotecas están disponibles en el entorno de ejecución de AWS Lambda, independientemente del tiempo de ejecución admitido que utilice, por lo que no es necesario que las incluya:

- AWS SDK: [AWS SDK para JavaScript](#) versión 2.108.0
- AWS SDK para Python 2.7 (Boto 3) versión 3-1.4.7 botocore-1.7.2
 - AWS SDK para Python 3.6 (Boto 3) versión 3-1.4.7 botocore-1.7.2
- Compilación de `java-1.8.0-openjdk` de Amazon Linux para Java.

Variables de entorno disponibles para las funciones de Lambda

A continuación se muestra una lista de las variables de entorno que forman parte del entorno de ejecución de AWS Lambda y que están disponibles para las funciones de Lambda. En la tabla siguiente se indican las que están reservadas por AWS Lambda y no pueden modificarse, así como las que se pueden establecer al crear una función de Lambda. Para obtener más información acerca del uso de las variables de entorno con la función de Lambda, consulte [Variables de entorno \(p. 97\)](#).

Variables de entorno de Lambda

| Clave | Reservada | Valor |
|-------------------|-----------|--|
| LAMBDA_TASK_ROOT | Sí | Contiene la ruta al código de la función de Lambda. |
| AWS_EXECUTION_ENV | Sí | La variable de entorno se establece en una de las siguientes opciones, en función del tiempo de ejecución de la función de Lambda: <ul style="list-style-type: none">• AWS_Lambda_java8• AWS_Lambda_nodejs• AWS_Lambda_nodejs4.3• AWS_Lambda_nodejs6.10• AWS_Lambda_python2.7• AWS_Lambda_python3.6 |

| Clave | Reservada | Valor |
|---|-----------|--|
| | | <ul style="list-style-type: none"> • AWS_Lambda_dotnetcore1.0 |
| LAMBDA_RUNTIME_DIR | Sí | Restringido a los artefactos relacionados con el tiempo de ejecución de Lambda. Por ejemplo, aws-sdk for Node.js y boto3 para Python se pueden encontrar en esta ruta. |
| AWS_REGION | Sí | La región de AWS donde se ejecuta la función de Lambda. |
| AWS_DEFAULT_REGION | Sí | La región de AWS donde se ejecuta la función de Lambda. |
| AWS_LAMBDA_LOG_GROUP_NAME | Sí | El nombre de grupo de Amazon CloudWatch Logs donde se crean los flujos de logs que contienen los logs de la función de Lambda. |
| AWS_LAMBDA_LOG_STREAM_NAME | Sí | Los flujos de Amazon CloudWatch Logs que contienen los logs de la función de Lambda. |
| AWS_LAMBDA_FUNCTION_NAME | Sí | El nombre de la función de Lambda. |
| AWS_LAMBDA_FUNCTION_MEMORY_SIZE | | El tamaño de la función de Lambda en MB. |
| AWS_LAMBDA_FUNCTION_VERSION | Sí | La versión de la función de Lambda. |
| AWS_ACCESS_KEY AWS_ACCESS_KEY_ID AWS_SECRET_KEY AWS_SECRET_ACCESS_KEY AWS_SESSION_TOKEN AWS_SECURITY_TOKEN | Sí | Las credenciales de seguridad necesarias para ejecutar la función de Lambda, en función del tiempo de ejecución utilizado. Los distintos tiempos de ejecución utilizan un subconjunto de estas claves. Se generan mediante un rol de ejecución de IAM que se especifica para la función. |
| PATH | No | Contiene /usr/local/bin, /usr/bin o /bin para la ejecución de ejecutables. |
| LANG | No | Establézcalo en en_US.UTF-8. Esta es la configuración regional del tiempo de ejecución. |
| LD_LIBRARY_PATH | No | Contiene /lib64, /usr/lib64, LAMBDA_TASK_ROOT, LAMBDA_TASK_ROOT/lib. Se utiliza para almacenar bibliotecas auxiliares y código de funciones. |

| Clave | Reservada | Valor |
|------------|-----------|--|
| NODE_PATH | No | Establézcalo para el tiempo de ejecución de Node.js. Contiene LAMBDA_RUNTIME_DIR, LAMBDA_RUNTIME_DIR/node_modules, LAMBDA_TASK_ROOT. |
| PYTHONPATH | No | Establézcalo para el tiempo de ejecución de Python. Contiene LAMBDA_RUNTIME_DIR. |
| TZ | Sí | La hora local actual. El valor predeterminado es UTC . |

Lectura recomendada

Si es la primera vez que utiliza AWS Lambda, le sugerimos que lea todos los temas de la sección Funcionamiento para familiarizarse con Lambda, empezando por [Funcionamiento \(p. 188\)](#).

Después de leer todos los temas de la sección Funcionamiento, le recomendamos que lea [Creación de funciones de Lambda \(p. 5\)](#), que pruebe el ejercicio de [Introducción \(p. 199\)](#) y, a continuación, que examine los [Casos de uso \(p. 214\)](#). Cada caso de uso proporciona instrucciones paso a paso para configurar la experiencia integral.

Introducción

En esta sección Introducción, hará lo siguiente:

- Configurar una cuenta de AWS y una AWS Command Line Interface (AWS CLI). La mayoría de los tutoriales utilizan los comandos de la AWS CLI.
- Crear y probar una sencilla función Hello World de Lambda.

Para comenzar, realice los pasos de los siguientes temas:

Temas

- [Paso 1: Configuración de una cuenta de AWS y de la AWS CLI \(p. 199\)](#)
- [Paso 2: Creación de una función HelloWorld de Lambda y exploración de la consola \(p. 202\)](#)
- [Paso 3: Creación de un microservicio sencillo utilizando Lambda y API Gateway \(p. 211\)](#)

Paso 1: Configuración de una cuenta de AWS y de la AWS CLI

Si aún no lo ha hecho, deberá inscribirse una cuenta de AWS y crear un usuario administrador en dicha cuenta. También deberá configurar la AWS Command Line Interface (AWS CLI). Muchos de los tutoriales utilizan la AWS CLI.

Siga las instrucciones de los temas siguientes para completar la configuración:

Temas

- [Paso 1.1: Configuración de una cuenta de AWS y creación de un usuario administrador \(p. 199\)](#)
- [Paso 1.2: Configuración de la AWS Command Line Interface \(AWS CLI\) \(p. 202\)](#)

Paso 1.1: Configuración de una cuenta de AWS y creación de un usuario administrador

Antes de usar AWS Lambda por primera vez, realice las siguientes tareas:

1. [Suscríbase a AWS \(p. 199\)](#)
2. [Creación de un usuario de IAM \(p. 200\)](#)

Suscríbase a AWS

Cuando se inscriba en Amazon Web Services (AWS), su cuenta de AWS se registrará automáticamente en todos los servicios de AWS, incluido AWS Lambda. Solo se le cobrará por los servicios que utilice.

Con AWS Lambda, solo paga por los recursos que utiliza. Para obtener más información acerca de las tarifas de uso de AWS Lambda, consulte la [página del producto de AWS Lambda](#). Si es un cliente nuevo de AWS, puede empezar a usar AWS Lambda gratuitamente. Para obtener más información, consulte [Capa gratuita de AWS](#).

Si ya dispone de una cuenta de AWS, pase a la siguiente tarea. Si no dispone de una cuenta de AWS, utilice el siguiente procedimiento para crear una.

Para crear una cuenta de AWS

1. Abra <https://aws.amazon.com/> y después elija Create an AWS Account.

Note

Esto podría no estar disponible en el navegador si ha iniciado previamente sesión en la Consola de administración de AWS. En ese caso, seleccione Sign In to the Console y después Create a new AWS account.

2. Siga las instrucciones en línea.

Parte del procedimiento de inscripción consiste en recibir una llamada telefónica e introducir un número PIN con el teclado del teléfono.

Anote su ID de cuenta de AWS porque lo necesitará en la siguiente tarea.

Creación de un usuario de IAM

Al utilizar los servicios de AWS, como AWS Lambda, debe proporcionar las credenciales para que el servicio pueda determinar si tiene los permisos necesarios para obtener acceso a los recursos propiedad de dicho servicio. La consola requiere que especifique la contraseña. Puede crear claves de acceso para su cuenta de AWS para tener acceso a la AWS CLI o API. Sin embargo, no es recomendable que tenga acceso a AWS con las credenciales de su cuenta de AWS. En lugar de ello, le recomendamos que utilice AWS Identity and Access Management (IAM). Cree un usuario de IAM, añada el usuario a un grupo de IAM con permisos administrativos y, a continuación, conceda permisos administrativos al usuario de IAM que ha creado. A continuación, podrá tener acceso a AWS mediante una URL especial y esas credenciales de usuario de IAM.

Si se ha registrado en AWS pero no ha creado un usuario de IAM, puede crear uno mediante la consola de IAM.

En los ejercicios y los tutoriales de la introducción de esta guía se presupone que tiene un usuario (`adminuser`) con privilegios de administrador. Cuando siga el procedimiento, cree un usuario con el nombre `adminuser`.

Para crear un usuario de IAM para sí mismo y agregarlo a un grupo de administradores

1. Utilice la dirección de correo electrónico y la contraseña de su cuenta de AWS para iniciar sesión en la [Consola de administración de AWS](#) como [usuario raíz de la cuenta de AWS](#).
2. En el panel de navegación de la consola, elija Usuarios y, a continuación, elija Añadir usuario.
3. En Nombre de usuario, escriba **Administrador**.
4. Active la casilla de verificación situada junto a Consola de administración de AWS access, seleccione Custom password y escriba la contraseña del nuevo usuario en el cuadro de texto. También puede seleccionar Require password reset para obligar al usuario a seleccionar una nueva contraseña la próxima vez que inicie sesión.
5. Elija Next: Permissions.
6. En la página Set permissions for user, elija Add user to group.
7. Elija Create group.
8. En el cuadro de diálogo Crear grupo, escriba **Administradores**.
9. En Filter, elija Job function.
10. En la lista de políticas, active la casilla de verificación AdministratorAccess. A continuación, elija Create group.

11. Retroceda a la lista de grupos y active la casilla de verificación del nuevo grupo. Elija Refresh si es necesario para ver el grupo en la lista.
12. Elija Next: Review para ver la lista de suscripciones a grupos que se van a añadir al nuevo usuario. Cuando esté listo para continuar, elija Create user.

Puede usar este mismo proceso para crear más grupos y usuarios, y para conceder a los usuarios acceso los recursos de su cuenta de AWS. Para obtener más información sobre cómo usar las políticas para restringir los permisos de los usuarios a recursos de AWS específicos, vaya a [Administración de acceso](#) y [Políticas de ejemplo](#).

Para iniciar sesión con el nuevo usuario de IAM

1. Cierre la sesión de la Consola de administración de AWS.
2. Utilice el siguiente formato de URL para iniciar sesión en la consola:

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

aws_account_number es el ID de la cuenta de AWS, sin guiones. Por ejemplo, si su ID de cuenta de AWS es 1234-5678-9012, su número de cuenta de AWS será 123456789012. Para obtener información acerca de cómo encontrar su número de cuenta, consulte [Su ID de cuenta y alias de AWS](#) en la Guía del usuario de IAM.

3. Especifique el nombre y la contraseña del usuario de IAM que acaba de crear. Cuando haya iniciado sesión, en la barra de navegación se mostrará: *su_nombre_de_usuario @ su_id_de_cuenta_de_aws*.

Si no desea que la dirección URL de la página de inicio de sesión contenga el ID de su cuenta de AWS, puede crear un alias de cuenta.

Para crear o eliminar un alias de cuenta

1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.
2. En el panel de navegación, elija Dashboard.
3. Busque el enlace de inicio de sesión de los usuarios de IAM.
4. Para crear el alias, haga clic en Customize, introduzca el nombre que desea usar para el alias y, a continuación, elija Yes, Create.
5. Para eliminar el alias, elija Customize y, a continuación, elija Yes, Delete. La dirección URL de inicio de sesión volverá a utilizar el ID de cuenta de AWS.

Para iniciar sesión después de crear un alias de cuenta, use la siguiente dirección URL:

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

Para verificar el enlace de inicio de sesión de los usuarios de IAM de su cuenta, abra la consola de IAM y busque IAM users sign-in link: en el panel.

Para obtener más información sobre IAM, consulte lo siguiente:

- [AWS Identity and Access Management \(IAM\)](#)
- [Introducción](#)
- [Guía del usuario de IAM](#)

Paso siguiente

[Paso 1.2: Configuración de la AWS Command Line Interface \(AWS CLI\) \(p. 202\)](#)

Paso 1.2: Configuración de la AWS Command Line Interface (AWS CLI)

En todos los ejercicios de esta guía se presupone que está utilizando las credenciales de usuario administrador (`adminuser`) en su cuenta para realizar las operaciones. Para obtener instrucciones acerca de cómo crear un usuario administrador en su cuenta de AWS, consulte [Paso 1.1: Configuración de una cuenta de AWS y creación de un usuario administrador \(p. 199\)](#) y, a continuación, siga los pasos para descargar y configurar la AWS Command Line Interface (AWS CLI).

Para configurar la AWS CLI

1. Descargue y configure la AWS CLI. Para obtener instrucciones, consulte los siguientes temas de la guía AWS Command Line Interface Guía del usuario.
 - [Configuración inicial de la AWS Command Line Interface](#)
 - [Configuración de la AWS Command Line Interface](#)
2. Añada un perfil con nombre para el usuario administrador en el archivo de configuración de la AWS CLI. Puede utilizar este perfil cuando ejecute los comandos de la AWS CLI.

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

Para ver una lista de las regiones de AWS disponibles, consulte [Regions and Endpoints](#) en la guía Referencia general de Amazon Web Services.

3. Verifique la configuración introduciendo los siguientes comandos en el símbolo del sistema.
 - Pruebe el comando de ayuda para verificar que la AWS CLI está instalada en su equipo:

```
aws help
```

- Pruebe un comando de Lambda para verificar que el usuario puede obtener acceso a AWS Lambda. Este comando muestra las funciones de Lambda de la cuenta, si las hay. La AWS CLI utiliza las credenciales de `adminuser` para autenticar la solicitud.

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

Ahora que ha configurado una cuenta y la AWS CLI, ya puede crear su primera función de Lambda. Para obtener instrucciones, consulte [Paso 2: Creación de una función HelloWorld de Lambda y exploración de la consola \(p. 202\)](#).

Paso 2: Creación de una función HelloWorld de Lambda y exploración de la consola

En este ejercicio de introducción, primero deberá crear una función Hello World de Lambda utilizando la consola de AWS Lambda. A continuación, deberá invocar la función de Lambda de forma manual

utilizando los datos del evento de muestra. AWS Lambda ejecuta la función de Lambda y devuelve los resultados. Seguidamente, podrá verificar los resultados de la ejecución, incluidos los logs creados por la función de Lambda y varias métricas de CloudWatch.

A medida que siga los pasos, también se familiarizará con la consola de AWS Lambda lo que le permitirá:

- Explorar los proyectos. Cada proyecto incluye código y configuraciones de muestra que le permiten crear funciones de Lambda con tan solo unos clics. El ejercicio de introducción utiliza el proyecto hello-world-python.
- Ver y actualizar la información de la configuración de la función de Lambda.
- Invocar una función de Lambda de forma manual y examinar los resultados en la sección Execution results.
- Monitorizar las métricas de CloudWatch en la consola.

Aunque no es necesario, le recomendamos que lea [Funcionamiento \(p. 188\)](#) en primer lugar.

Preparación para la introducción

En primer lugar, inscríbase en una cuenta de AWS y cree un usuario administrador en ella. Para obtener instrucciones, consulte [Paso 1: Configuración de una cuenta de AWS y de la AWS CLI \(p. 199\)](#).

Paso siguiente

[Paso 2.1: Creación de una función Hello World de Lambda \(p. 203\)](#)

Paso 2.1: Creación de una función Hello World de Lambda

Siga los pasos de esta sección para crear una función Hello World de Lambda. En este paso, hará lo siguiente:

- Seleccionar un proyecto: para este ejercicio, utilizará el proyecto hello-world-python. Proporciona código de muestra escrito en Python. En este ejercicio, el lenguaje utilizado para la función de Lambda es irrelevante. Posteriormente podrá crear sus propias funciones de Lambda en cualquiera de los lenguajes admitidos.

Los proyectos proporcionan código de ejemplo que permite realizar un procesamiento mínimo. La mayoría de los proyectos procesan eventos de orígenes de eventos específicos, como Amazon S3, DynamoDB o aplicaciones personalizadas. Por ejemplo, si selecciona un proyecto s3-get-object, este proporciona código de muestra que procesa un evento de creación de objeto publicado por Amazon S3 y que Lambda recibe como parámetro.

- Configurar una función: dado que se selecciona un proyecto para este ejercicio, la consola habrá rellenado previamente parte de la información de configuración. Por ejemplo, preconfigura Python 3.6 o Python 2.7 como tiempo de ejecución, proporciona código de ejemplo, identifica el controlador del código de muestra y otra información de configuración, como la memoria y el tiempo de espera. Para obtener más información acerca de la configuración de funciones, consulte [Funciones de Lambda \(p. 3\)](#). Para obtener más información acerca de los parámetros de configuración de las funciones, consulte [CreateFunction \(p. 385\)](#).

También creará un rol de IAM (denominado rol de ejecución) con los permisos necesarios que AWS Lambda puede asumir para invocar la función de Lambda en su nombre.

Para crear una función Hello World de Lambda

1. Inicie sesión en la Consola de administración de AWS y abra la consola de AWS Lambda.

2. Elija Get Started Now.

The screenshot shows the AWS Lambda landing page. At the top, there's a navigation bar with 'Services', 'Resource Groups', and a star icon. On the right, it shows 'AWS Lambda (w/Lambda) N. Virginia'. Below the navigation is a large orange Lambda logo. The main heading is 'AWS Lambda'. A subtext explains: 'AWS Lambda lets you run code in response to events, without provisioning or managing servers. Just upload your code and Lambda will take care of everything required to run and scale it with high availability.' Below this is a 'Get Started Now' button and a link to 'Learn more about AWS Lambda'. The page then highlights three key features: 'No Servers to Manage' (illustrated with a gear icon), 'Continuous Scaling' (illustrated with a graph icon), and 'Subsecond Metering' (illustrated with a meter icon). Each feature has a brief description. At the bottom, there's a 'Feedback' link, language selection ('English'), copyright information ('© 2008 - 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.'), and links to 'AWS Lambda Documentation and Support' (Getting Started Guide, Documentation, Support, Forums), 'Privacy Policy', and 'Terms'.

Note

Si no se ha creado ninguna función de Lambda, la consola únicamente muestra la página Get Started Now. Si ya ha creado alguna función, verá la página Lambda > Functions. En la página de lista, elija Create a Lambda function para ir a la página Lambda > New function.

3. En la página Select blueprint, examine primero los proyectos disponibles. A continuación, seleccione un proyecto específico para este ejercicio de introducción.
 - a. Revise los proyectos. También puede utilizar Filter para buscar proyectos específicos. Por ejemplo:
 - Introduzca **s3** en Filtro para obtener únicamente la lista de proyectos disponibles para procesar eventos de Amazon S3.
 - Introduzca **dynamodb** en Filtro para obtener una lista de los proyectos disponibles para procesar eventos de Amazon DynamoDB.
 - b. En este ejercicio de introducción, introduzca **hello-world-python** en Filtro y, a continuación, elija el proyecto hello-world-python.
4. En Basic information, haga lo siguiente:
 - En Name*, especifique el nombre de la función de Lambda.
 - En Role*, elija Create new role from template(s).

- Escriba un nombre para el rol en Role Name*.
- En Policy templates, Lambda proporciona una lista de plantillas opcionales que, si se seleccionan, crean automáticamente el rol con los permisos necesarios asociados a dicha política. Para obtener la lista de políticas del campo Policy templates, consulte [Plantillas de política \(p. 362\)](#). Para este tutorial, puede dejar este campo en blanco, ya que la función de Lambda ya tiene el permiso de ejecución básico que necesita.
- En la sección Lambda Function Code, haga lo siguiente:
 - Revise el código de muestra. Tenga en cuenta que la consola guarda este código como `lambda_handler.py`. A continuación, la consola comprime el archivo y lo carga en AWS Lambda para crear la función de Lambda.
 - El código de muestra procesa los eventos entrantes de la siguiente forma:

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

Después de crear la función de Lambda, puede invocarla utilizando los eventos de muestra de este formulario en la siguiente sección.

- Elija Create function.
5. En Configuration, en la sección Lambda function code, observe lo siguiente
 - Runtime es Python 2.7
 - Se proporciona código escrito en Python. Lee datos de eventos entrantes y registra parte de la información en CloudWatch.
 - Handler muestra el valor `lambda_function.lambda_handler`. Se trata de [nombre-de-archivo.función-de-controlador](#). La consola guarda el código de muestra en el archivo `lambda_function.py` y, en el código, `lambda_handler` es el nombre de la función que recibe el evento como parámetro cuando se invoca la función de Lambda. Para obtener más información, consulte [Controlador de funciones de Lambda \(Python\) \(p. 44\)](#).
 6. Elija la pestaña Triggers. En la página Triggers, tiene la opción de elegir un servicio que activa automáticamente la función de Lambda; para ello, seleccione el botón Add trigger y, a continuación, elija el cuadro gris con puntos suspensivos (...) para mostrar una lista de los servicios disponibles.
 - a. En función del servicio que seleccione, el sistema le pedirá que proporcione información pertinente para dicho servicio. Por ejemplo, si selecciona DynamoDB, debe proporcionar lo siguiente:
 - El nombre de la tabla de DynamoDB.
 - Tamaño del lote
 - La posición inicial
 - b. En este ejemplo, no configure un disparador y elija la pestaña Monitoring.
 7. La página Monitoring proporcionará métricas de CloudWatch inmediatas para las invocaciones de la función de Lambda, así como enlaces a otras guías útiles, incluida [Solución de problemas de aplicaciones basadas en Lambda \(p. 176\)](#).
 8. Elija la pestaña Configuration y, a continuación, observe las otras pestañas de la consola:
 - Environment variables: en las funciones de Lambda, le permiten pasar ajustes de forma dinámica al código de la función y a las bibliotecas, sin necesidad de realizar cambios en el código. Para obtener más información, consulte [Variables de entorno \(p. 97\)](#).

- Tags: son pares clave-valor que se asocian a los recursos de AWS para organizarlos mejor. Para obtener más información, consulte [Etiquetado de funciones de Lambda \(p. 105\)](#).
- Execution role: le permite administrar la seguridad de la función, utilizando los roles y las políticas definidas o creando otros nuevos. Para obtener más información, consulte [Autenticación y control de acceso de AWS Lambda \(p. 337\)](#).
- Basic settings: le permite determinar la asignación de memoria y el límite de tiempo de espera de la función de Lambda. Para obtener más información, consulte [Límites de AWS Lambda \(p. 335\)](#).
- Network: le permite seleccionar una VPC a la que obtendrá acceso la función. Para obtener más información, consulte [Configuración de una función de Lambda para acceder recursos en una Amazon VPC \(p. 108\)](#).
- Debugging and error handling: le permite seleccionar un recurso [Colas de mensajes fallidos \(p. 128\)](#) para analizar los reintentos fallidos de invocación de la función. También le permite habilitar el rastreo activo. Para obtener más información, consulte [Solución de problemas de aplicaciones basadas en Lambda \(p. 176\)](#).

Paso siguiente

[Paso 2.2: Invocación manual de la función de Lambda y verificación de los resultados, los logs y las métricas \(p. 206\)](#)

Paso 2.2: Invocación manual de la función de Lambda y verificación de los resultados, los logs y las métricas

Siga estos pasos para invocar la función de Lambda utilizando los datos del evento de muestra que se proporcionan en la consola.

1. En la página Lambda > Functions > HelloWorld, elija Test.
2. En la página Input test event, elija Hello World en la lista Sample event template. En la ventana aparece la siguiente plantilla de evento de muestra.

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

Si lo desea, puede modificar las claves y los valores en el código JSON de muestra, pero no cambie la estructura del evento. Si realiza alguna modificación en las claves o los valores, debe actualizar el código de muestra según sea necesario. Elija Save and test.

3. AWS Lambda ejecuta la función en su nombre. El `handler` de la función de Lambda recibe el evento de muestra y, a continuación, lo procesa.
4. Si la ejecución se realiza correctamente, puede ver los resultados en la consola.

The screenshot shows the AWS Lambda execution history for a function named 'value1'. The execution was successful ('Execution result: succeeded'). The 'Summary' section includes details like Request ID (e306a993-6cbe-11e7-9ab1-335824bf2551), Duration (2.14 ms), Billed duration (100 ms), and Resources configured (128 MB). The 'Log output' section displays CloudWatch logs with entries for START, multiple log entries for processing, and an END event. A red box highlights the REPORT line in the log output, which provides detailed metrics: Duration: 2.14 ms, Billed Duration: 100 ms, Memory Size: 128 MB, and Max Memory Used: 17 MB.

Tenga en cuenta lo siguiente:

- La sección Execution result muestra el estado de ejecución como succeeded y también muestra los resultados de la ejecución de la función devueltos por la instrucción `return`.

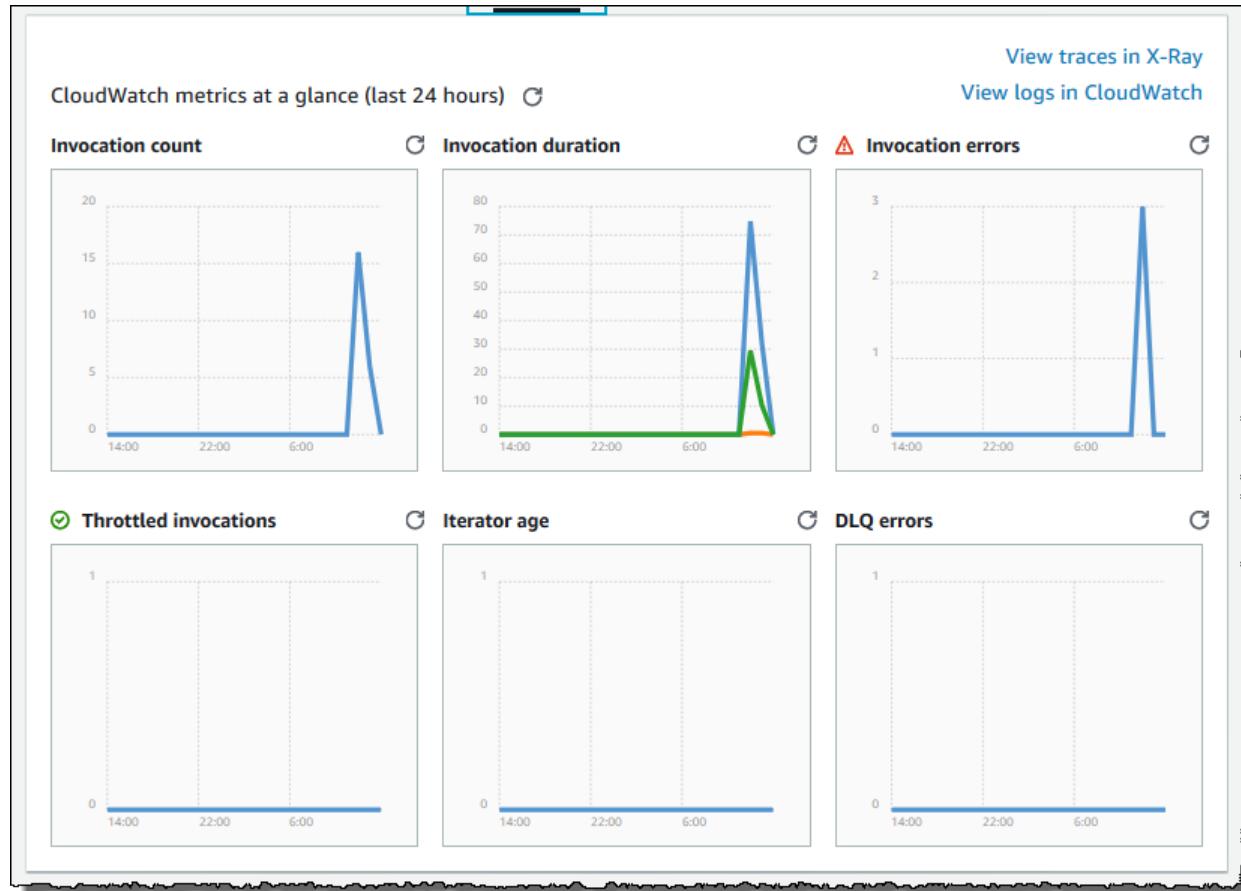
Note

La consola utiliza siempre el tipo de invocación `RequestResponse` (invocación síncrona) al invocar una función de Lambda, lo que hace que AWS Lambda devuelva inmediatamente una respuesta. Para obtener más información, consulte [Tipos de invocación \(p. 4\)](#).

- La sección Summary muestra la información principal proporcionada en la sección Log output (la línea REPORT del log de ejecución).
- La sección Log output muestra los logs que genera AWS Lambda para cada ejecución. Estos son los logs que escribe la función de Lambda en CloudWatch. La consola de AWS Lambda muestra estos logs para su comodidad.

Tenga en cuenta que el enlace [Click here](#) muestra los logs en la consola de CloudWatch. A continuación, la función añade logs a Amazon CloudWatch en el grupo de logs correspondiente a la función de Lambda.

- Ejecute la función de Lambda varias veces para recopilar algunas métricas que podrá ver en el siguiente paso.
- Elija la pestaña Monitoring para ver las métricas de CloudWatch de la función de Lambda. Esta página muestra cuatro métricas de CloudWatch.



Tenga en cuenta lo siguiente:

- El eje X muestra las últimas 24 horas desde la hora actual (por ejemplo, las 14:00 h tal como se muestra en la captura de pantalla).
- Invocation count muestra el número de invocaciones durante este intervalo.
- Invocation duration muestra cuánto tiempo ha tardado la función de Lambda en ejecutarse. Muestra el tiempo de ejecución mínimo, el máximo y el promedio.
- Invocation errors muestra el número de veces que ha fallado la función de Lambda. Puede comparar el número de veces que se ha ejecutado la función y cuántas veces ha fallado (si es que lo ha hecho).
- Las métricas Throttled invocations muestran si AWS Lambda ha restringido la invocación de la función de Lambda. Para obtener más información, consulte [Límites de AWS Lambda \(p. 335\)](#).
- La consola de AWS Lambda muestra estas métricas de CloudWatch para su comodidad. Puede verlas en la consola de Amazon CloudWatch haciendo clic en cualquiera de ellas.

Paso siguiente

[Paso 2.3: Creación de una función de Lambda escrita en Java \(opcional\) \(p. 209\)](#)

Paso 2.3: Creación de una función de Lambda escrita en Java (opcional)

Los proyectos proporcionan código de muestra escrito en Python o en Node.js. Puede modificar fácilmente el ejemplo mediante el editor de código integrado de la consola. Sin embargo, no dispone de proyectos que le permitan escribir código Java para la función de Lambda. Asimismo, tampoco existe un editor de código integrado que le permita escribir código Java en la consola de AWS Lambda.

Esto significa que debe escribir el código Java y crear su paquete de implementación fuera de la consola. Una vez creado el paquete de implementación, puede utilizar la consola para cargarlo en AWS Lambda y crear la función de Lambda. También puede utilizar la consola para probar la función invocándola manualmente.

En esta sección se crea una función de Lambda utilizando el siguiente ejemplo de código Java.

```
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);
    }
}
```

El modelo de programación explica detalladamente cómo escribir código Java; por ejemplo, los tipos de entrada/salida que admite AWS Lambda. Para obtener más información acerca del modelo de programación, consulte [Modelo de programación para crear funciones de Lambda en Java \(p. 25\)](#). De momento, tenga en cuenta lo siguiente sobre este código:

- Al empaquetar y cargar este código para crear la función de Lambda, debe especificar la referencia al método `example.Hello::myHandler` como controlador.
- El controlador de este ejemplo utiliza el tipo `int` para la entrada y el tipo `String` para la salida.

AWS Lambda admite la entrada/salida de tipos serializables en formato JSON y tipos `InputStream`/`OutputStream`. Al invocar esta función, pasará un entero de muestra (por ejemplo, 123).

- En este ejercicio se utiliza la consola para invocar manualmente esta función de Lambda. La consola siempre utiliza el tipo de invocación `RequestResponse` (síncrona) y, por lo tanto, verá la respuesta en ella.
- El controlador incluye el parámetro opcional `Context`. En el código se utiliza la instrucción `LambdaLogger` proporcionada por el objeto `Context` para escribir entradas de log en los logs de CloudWatch. Para obtener información acerca de cómo usar el objeto `Context`, consulte [El objeto context \(Java\) \(p. 36\)](#).

En primer lugar, debe empaquetar este código y todas las dependencias en un paquete de implementación. A continuación, puede utilizar el ejercicio de introducción para cargar el paquete, crear la función de Lambda y probarla en la consola.

Paso siguiente

[Paso 2.4: Creación de una función de Lambda escrita en C# \(opcional\) \(p. 210\)](#)

Paso 2.4: Creación de una función de Lambda escrita en C# (opcional)

Los proyectos de la consola de AWS Lambda proporcionan código de muestra escrito en Python o en Node.js. Puede modificar fácilmente el ejemplo mediante el editor de código integrado de la consola. Sin embargo, no dispone de proyectos que le permitan escribir código C# para la función de Lambda. Asimismo, tampoco existe un editor de código integrado que le permita escribir código C# en la consola de AWS Lambda.

Aunque la consola de Lambda no permite la edición de lenguajes compilados como, por ejemplo, Java y C#, puede utilizar el IDE que desee, como Visual Studio, para crear y empaquetar su código y sus bibliotecas de C#. Una vez empaquetados en un archivo ZIP, puede utilizar la consola de AWS Lambda para cargar y probar funciones de Lambda en C#, así como para ver logs y métricas relacionadas con ellas.

En esta sección se crea una función de Lambda utilizando el siguiente ejemplo de código C#.

```
using Amazon.Lambda.Core;
namespace LambdaFunctionExample{
public class Hello {
    public string MyHandler(int count, ILambdaContext context) {
        var logger = context.Logger;
        logger.Log("received : " + count);
        return count.ToString();
    }
}
}
```

La firma del controlador de la función de Lambda debería tener el formato

Ensamblando: :EspacioDeNombres.NOMBREDeClase: :NombreDeMétodo. El modelo de programación explica detalladamente cómo escribir código C#; por ejemplo, los tipos de entrada/salida que admite AWS Lambda. Para obtener más información acerca del modelo de programación, consulte [Modelo de programación para crear funciones de Lambda en C# \(p. 53\)](#). De momento, tenga en cuenta lo siguiente sobre este código:

- El controlador de este ejemplo utiliza el tipo `int` para la entrada y el tipo `string` para la salida.

Al invocar esta función, pasará un entero de muestra (por ejemplo, 123).

- En este ejercicio se utiliza la consola para probar manualmente esta función de Lambda. La consola siempre utiliza el tipo de invocación `RequestResponse` (síncrona) y, por lo tanto, verá la respuesta en ella.
- El controlador incluye el parámetro opcional `ILambdaContext`. En el código se utiliza el controlador `LambdaLogger` proporcionado por el objeto `Amazon.Lambda.Core.LambdaLogger` para escribir entradas de log en los logs de CloudWatch. Para obtener información acerca de cómo usar el objeto `ILambdaContext`, consulte [El objeto context \(C#\) \(p. 58\)](#).

En primer lugar, debe empaquetar este código y todas las dependencias en un paquete de implementación. A continuación, puede utilizar el ejercicio de introducción para cargar el paquete, crear la función de Lambda y probarla en la consola. Para obtener más información, consulte [Creación de un paquete de implementación \(C#\) \(p. 65\)](#).

Paso siguiente

[Paso 3: Creación de un microservicio sencillo utilizando Lambda y API Gateway \(p. 211\)](#)

Paso 3: Creación de un microservicio sencillo utilizando Lambda y API Gateway

En este ejercicio, utilizará la consola de Lambda para crear una función de Lambda (`MyLambdaMicroservice`) y un punto de enlace de Amazon API Gateway para activar dicha función. Podrá llamar al punto de enlace con cualquier método (GET, POST, PATCH, etc.) para activar la función de Lambda. Una vez realizada la llamada al punto de enlace, se pasará la solicitud completa a la función de Lambda. La acción de la función dependerá del método con el que llame al punto de enlace:

- DELETE: elimina un elemento de una tabla de DynamoDB
- GET: examina la tabla y devuelve todos los elementos
- POST: crea un elemento
- PUT: actualiza un elemento

Paso siguiente

[Paso 3.1: Creación de una API utilizando Amazon API Gateway \(p. 211\)](#)

Paso 3.1: Creación de una API utilizando Amazon API Gateway

Siga los pasos de esta sección para crear una nueva función de Lambda y un punto de enlace de API Gateway para activarla:

1. Inicie sesión en la consola de administración de AWS y abra la consola de AWS Lambda.
2. Seleccione Create Lambda function.
3. En la página Select blueprint, elija el proyecto `microservice-http-endpoint`. Puede utilizar Filter para localizarlo.
4. La página Configure triggers se llenará con un disparador de API Gateway. El nombre predeterminado de la API que se creará es `LambdaMicroservice` (puede cambiarlo mediante el campo API Name si lo desea).

Note

Cuando se completa el asistente y se crea la función, Lambda crea automáticamente un recurso de proxy denominado `MyLambdaMicroservice` (nombre de la función) bajo el nombre de la API que se ha seleccionado. Para obtener más información acerca de los recursos de proxy, consulte [Configuración de la integración del proxy de API Gateway](#). Un recurso de proxy tiene un tipo de integración `AWS_PROXY` y un método catch-all `ANY`. El tipo de integración `AWS_PROXY` aplica una plantilla de mapeo predeterminada para pasar toda la solicitud a la función de Lambda y transforma la salida de esta función en respuestas HTTP. El método `ANY` define la misma configuración de integración para todos los métodos admitidos, incluidos `GET`, `POST`, `PATCH`, `DELETE` y otros.

Una vez revisado el disparador, seleccione Next.

5. En la página Configure function, haga lo siguiente:
 - a. Revise la información de configuración predeterminada de la función de Lambda y asegúrese de que:
 - Runtime es `Node.js 6.10`.
 - Se proporciona código escrito en JavaScript. El código realiza operaciones de DynamoDB en función del método llamado y la carga proporcionada.

- Handler muestra `index.handler`. El formato es el siguiente: `filename.handler-function`
- b. Introduzca el nombre de función `MyLambdaMicroservice` en el campo Name.
- c. En el campo Role, introduzca un nombre de rol para el rol que se va a crear.

Note

El proyecto `microservice-http-endpoint` rellena previamente el campo Policy templates con la plantilla de políticas Simple Microservice permission, información que se añadirá al nuevo rol en el momento de su creación. Esto añade automáticamente al nuevo rol los permisos necesarios asociados a esa política. Para obtener más información, consulte [Plantillas de política \(p. 362\)](#).

6. Elija Create function.

Paso siguiente

[Paso 3.2: Prueba de envío de una solicitud HTTPS \(p. 212\)](#)

Paso 3.2: Prueba de envío de una solicitud HTTPS

En este paso, utilizará la consola para probar la función de Lambda. Además, puede ejecutar un comando `curl` para probar la experiencia integral. Es decir, enviar una solicitud HTTPS a su método de la API y hacer que Amazon API Gateway invoque la función de Lambda. Con el fin de completar los pasos, asegúrese de que ha creado una tabla de DynamoDB denominada “MyTable”. Para obtener más información, consulte [Paso 3.1: Creación de una tabla de DynamoDB con un flujo habilitado \(p. 252\)](#)

1. Con la función `MyLambdaMicroService` aún abierta en la consola, en la pestaña Actions, elija Configure test event.
2. Reemplace el texto existente por el siguiente:

```
{  
  "httpMethod": "GET",  
  "queryStringParameters": {  
    "TableName": "MyTable"  
  }  
}
```

3. Una vez introducido el texto anterior, elija Save and test.

Paso siguiente

[Paso 3.3: Uso de otros proyectos \(opcional\) \(p. 212\)](#)

Paso 3.3: Uso de otros proyectos (opcional)

Si lo desea, puede probar los siguientes ejercicios:

- Ha utilizado el proyecto `hello-world-python` en el ejercicio de introducción. Este proyecto proporciona código de muestra escrito en Python. Existe también el proyecto `hello-world` que proporciona código de función de Lambda similar escrito en Node.js.
- Tanto el proyecto `hello-world-python` como el proyecto `hello-world` procesan eventos personalizados. En este ejercicio de introducción, ha utilizado datos del evento de muestra introducidos manualmente. Puede escribir funciones de Lambda para procesar eventos publicados por orígenes de eventos como Amazon S3 y DynamoDB. Para ello, es necesario configurar orígenes de eventos en la consola.

Por ejemplo, puede escribir una función de Lambda para procesar eventos de Amazon S3. A continuación, puede configurar Amazon S3 como origen de eventos para publicar eventos de creación de objetos en AWS Lambda. Cuando se carga un objeto en el bucket, Amazon S3 detecta el evento e invoca la función de Lambda. La función de Lambda recibe los datos del evento como parámetro. Puede verificar que la función de Lambda se ha ejecutado revisando los logs de CloudWatch, tanto en la consola de Lambda como en la consola de CloudWatch.

La consola de Lambda proporciona un proyecto que permite configurar una función de Lambda de ejemplo para procesar eventos de Amazon S3. Cuando cree una función de Lambda en la consola, en la página Select blueprint, introduzca **s3** en el cuadro Filter para buscar en la lista de proyectos disponibles.

Para obtener más información acerca del uso de distintos orígenes de eventos, consulte [Casos de uso \(p. 214\)](#).

Paso siguiente

[Siguientes pasos \(p. 213\)](#)

Siguientes pasos

Este ejercicio de introducción le ha proporcionado información general acerca de cómo utilizar la consola de AWS Lambda.

Las funciones de AWS Lambda también se pueden invocar automáticamente en respuesta a eventos en otros servicios de AWS, como Amazon S3 y DynamoDB. Las funciones de Lambda también se pueden invocar bajo demanda a través de HTTPS. Asimismo, puede crear sus propios orígenes de eventos personalizados e invocar funciones de Lambda bajo demanda. Para obtener más información, consulte [Funcionamiento \(p. 188\)](#).

En función de su escenario de integración y de si su aplicación necesita invocaciones basadas en eventos o invocaciones bajo demanda de las funciones de Lambda, consulte las siguientes secciones:

- [Uso de AWS Lambda con Amazon S3 \(p. 214\)](#)
- [Uso de AWS Lambda con Kinesis \(p. 232\)](#)
- [Uso de AWS Lambda con Amazon DynamoDB \(p. 243\)](#)
- [Uso de AWS Lambda con AWS CloudTrail \(p. 254\)](#)
- [Uso de AWS Lambda con Amazon API Gateway \(bajo demanda a través de HTTPS\) \(p. 275\)](#)
- [Uso de AWS Lambda como backend de aplicaciones móviles \(origen de eventos personalizados: Android\) \(p. 289\)](#)

La consola ofrece varios proyectos que le permiten configurar rápidamente funciones de Lambda de ejemplo que pueden procesar eventos provenientes de estos orígenes de eventos. Le recomendamos que explore otros proyectos en la consola para comenzar a utilizar funciones de Lambda activadas por estos orígenes de eventos.

Ejemplos de cómo utilizar AWS Lambda

Los casos de uso de AWS Lambda pueden agruparse en las categorías siguientes:

- Uso de AWS Lambda con servicios de AWS como orígenes de eventos: los orígenes de eventos publican eventos que provocan la invocación de la función de Lambda. Estos pueden ser servicios de AWS, como Amazon S3. Para obtener más información y tutoriales, consulte los siguientes temas:

[Uso de AWS Lambda con Amazon S3 \(p. 214\)](#)

[Uso de AWS Lambda con Kinesis \(p. 232\)](#)

[Uso de AWS Lambda con Amazon DynamoDB \(p. 243\)](#)

[Uso de AWS Lambda con AWS CloudTrail \(p. 254\)](#)

[Uso de AWS Lambda con Amazon SNS desde cuentas distintas \(p. 270\)](#)

- Invocación bajo demanda de la función de Lambda a través de HTTPS (Amazon API Gateway): las funciones de Lambda no solo se pueden invocar mediante orígenes de eventos, sino que también se pueden invocar a través de HTTPS. Puede hacerlo definiendo un punto de enlace y una API de REST personalizados utilizando API Gateway. Para obtener más información y un tutorial, consulte [Uso de AWS Lambda con Amazon API Gateway \(bajo demanda a través de HTTPS\) \(p. 275\)](#).
- Invocación bajo demanda de la función de Lambda (creando sus propios orígenes de eventos utilizando aplicaciones personalizadas): las aplicaciones de los usuarios, como las aplicaciones cliente, móviles o web, pueden publicar eventos e invocar funciones de Lambda mediante los SDK de AWS o los AWS Mobile SDK, como el SDK de AWS Mobile para Android. Para obtener más información y un tutorial, consulte [Introducción \(p. 199\)](#) y [Uso de AWS Lambda como backend de aplicaciones móviles \(origen de eventos personalizados: Android\) \(p. 289\)](#).
- Eventos programados: también puede configurar AWS Lambda para que invoque el código de forma periódica y programada utilizando la consola de AWS Lambda. Puede especificar una frecuencia fija (un número de horas, días o semanas) o una expresión cron. Para obtener más información y un tutorial, consulte [Uso de AWS Lambda con eventos programados \(p. 302\)](#).

Uso de AWS Lambda con Amazon S3

Amazon S3 puede publicar eventos (por ejemplo, cuando se crea un objeto en un bucket) en AWS Lambda e invocar una función de Lambda pasando los datos del evento como parámetro. Esta integración permite escribir funciones de Lambda que procesan eventos de Amazon S3. En Amazon S3, debe añadir una configuración de notificaciones de bucket que identifique el tipo de evento que desea que Amazon S3 publique y la función de Lambda que desea invocar.

Tenga en cuenta lo siguiente sobre cómo funciona la integración entre Amazon S3 y AWS Lambda:

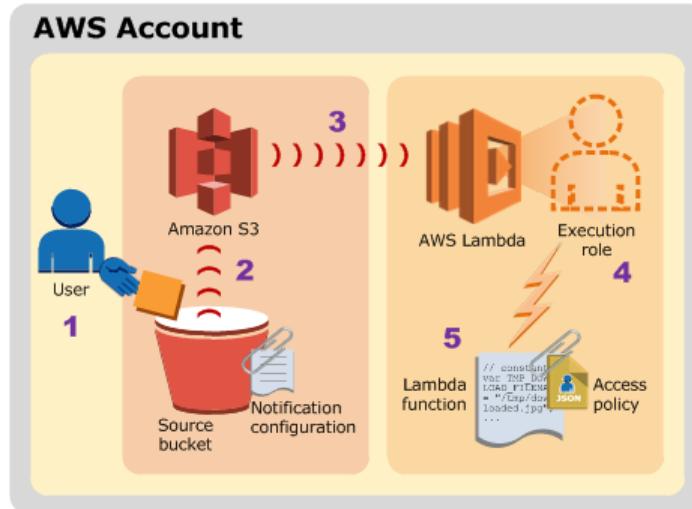
- **Modelo no basado en flujo (asíncrono):** es un modelo (consulte [Mapeo de orígenes de eventos \(p. 134\)](#)) en el que Amazon S3 monitoriza un bucket e invoca la función de Lambda pasando los datos del evento como parámetro. En un modelo de inserción, el mapeo de origen de eventos se mantiene en Amazon S3 mediante la configuración de notificaciones de bucket. En la configuración, debe especificar los tipos de eventos que desea que Amazon S3 monitoree y la función de AWS Lambda que desea que Amazon S3 invoque. Para obtener más información, consulte [Configuración de notificaciones de eventos de Amazon S3](#) en la Guía para desarrolladores de Amazon Simple Storage Service.

- Invocación asíncrona: AWS Lambda invoca una función de Lambda mediante el tipo de invocación Event (invocación asíncrona). Para obtener más información acerca de los tipos de invocación, consulte [Tipos de invocación \(p. 4\)](#).
- Estructura del evento: el evento que recibe la función de Lambda es para un solo objeto y proporciona información, como el nombre del bucket y el nombre de la clave de objeto.

Tenga en cuenta que existen dos tipos de políticas de permisos con las que se trabaja al configurar la experiencia integral:

- Permisos para la función de Lambda: independientemente de qué invoque una función de Lambda, AWS Lambda ejecuta la función asumiendo el rol de IAM (rol de ejecución) que se especifique en el momento de crear la función de Lambda. La política de permisos asociada a este rol se utiliza para conceder a la función de Lambda los permisos que necesita. Por ejemplo, si la función de Lambda necesita leer un objeto, debe conceder permisos para las acciones de Amazon S3 correspondientes en la política de permisos. Para obtener más información, consulte [Administración de permisos mediante un rol de IAM \(rol de ejecución\) \(p. 193\)](#).
- Permisos para que Amazon S3 invoque una función de Lambda: Amazon S3 no puede invocar una función de Lambda sin su permiso. Puede conceder este permiso a través de la política de permisos asociada a la función de Lambda.

En el siguiente diagrama, se resume el flujo:



1. El usuario carga un objeto en un bucket de S3 (evento de creación de objeto).
2. Amazon S3 detecta el evento de creación de objeto.
3. Amazon S3 invoca una función de Lambda especificada en la configuración de notificaciones del bucket.
4. AWS Lambda ejecuta la función de Lambda asumiendo el rol de ejecución que se especificó en el momento de crear la función de Lambda.
5. La función de Lambda se ejecuta.

Para ver un tutorial que muestra una configuración de ejemplo, consulte [Tutorial: Uso de AWS Lambda con Amazon S3 \(p. 215\)](#).

Tutorial: Uso de AWS Lambda con Amazon S3

Supongamos que desea crear una miniatura para cada imagen (objetos .jpg y .png) que se carga en un bucket. Puede crear una función de Lambda (`CreateThumbnail`) que Amazon S3 puede invocar cuando

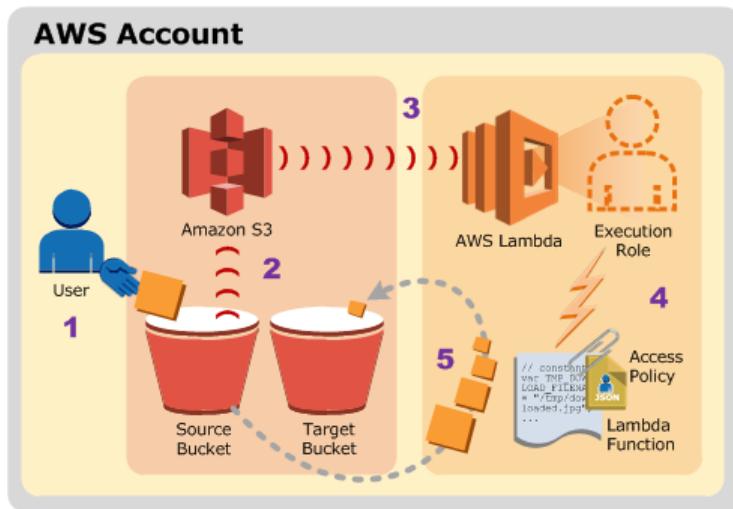
se crean objetos. A continuación, la función de Lambda puede leer el objeto de imagen del bucket de origen (*source*) y crear un bucket de destino para las imágenes en miniatura (en este tutorial, el bucket se denomina *source*resized).

Important

Debe utilizar dos buckets. Si utiliza el mismo bucket como origen y destino, cada miniatura que se carga en el bucket de origen desencadena otro evento de creación de objeto que, a su vez, vuelve a invocar la función de Lambda, con lo que se crea un bucle no deseado.

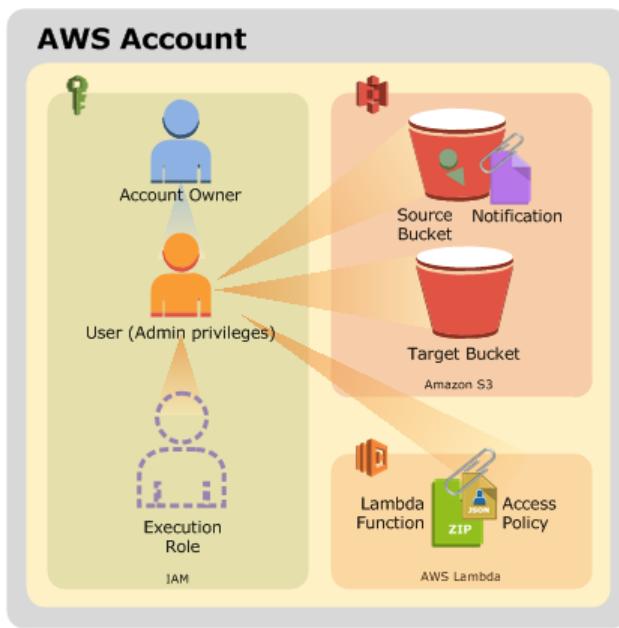
Resumen de implementación

En el diagrama siguiente se ilustra el flujo de la aplicación:



1. El usuario carga un objeto en el bucket de origen de Amazon S3 (evento de creación de objeto).
2. Amazon S3 detecta el evento de creación de objeto.
3. Amazon S3 publica el evento s3:ObjectCreated:* en AWS Lambda invocando la función de Lambda y pasando los datos del evento como parámetro de la función.
4. AWS Lambda ejecuta la función de Lambda asumiendo el rol de ejecución que se especificó en el momento de crear la función de Lambda.
5. A partir de los datos del evento que recibe, la función de Lambda conoce el nombre del bucket de origen y el nombre de la clave de objeto. La función de Lambda lee el objeto y crea una miniatura utilizando las bibliotecas de gráficos, y la guarda en el bucket de destino.

Tenga en cuenta que al finalizar este tutorial, tendrá los siguientes recursos de Amazon S3, Lambda e IAM en su cuenta:



En Lambda:

- Una función de Lambda.
- Una política de permisos de acceso asociada a la función de Lambda. Esta política de permisos se utiliza para conceder permisos a Amazon S3 para invocar la función de Lambda. También restringirá los permisos para que Amazon S3 pueda invocar la función de Lambda únicamente para los eventos de creación de objetos procedentes de un bucket específico que pertenece a una cuenta de AWS concreta.

Note

Es posible que una cuenta de AWS elimine un bucket y que alguna otra cuenta de AWS cree posteriormente un bucket con el mismo nombre. Las condiciones adicionales garantizan que Amazon S3 solo pueda invocar la función de Lambda si detecta eventos de creación de objetos procedentes de un bucket específico que pertenece a una cuenta de AWS determinada.

Para obtener más información, consulte [Funcionamiento \(p. 188\)](#).

En IAM:

- Un usuario administrador, denominado adminuser. No se recomienda utilizar las credenciales raíz de una cuenta de AWS. En su lugar, utilice las credenciales de adminuser para realizar los pasos en este tutorial.
- Un rol de IAM (rol de ejecución). Debe conceder los permisos que necesita la función de Lambda a través del política de permisos asociada a este rol.

En Amazon S3:

- Dos buckets, denominados **source** y **source resized**. Tenga en cuenta que **source** es un marcador de posición para el nombre, y debe reemplazarlo por el nombre del bucket real. Por ejemplo, si dispone de un bucket denominado example como origen, creará el bucket de destino **example resized**.
- Una configuración de notificaciones en el bucket de origen. Debe añadir una configuración de notificaciones al bucket de origen para identificar el tipo de eventos (eventos de creación de objetos) que desea que Amazon S3 publique en AWS Lambda y la función de Lambda que desea invocar. Para obtener más información acerca de la función de notificaciones de Amazon S3, consulte [Configuración](#)

de notificaciones de eventos de Amazon S3 en la Guía para desarrolladores de Amazon Simple Storage Service.

Ahora ya puede comenzar el tutorial. Tenga en cuenta que, después de la preparación inicial, el tutorial se divide en dos secciones principales:

- En primer lugar, debe realizar los pasos de configuración necesarios para crear una función de Lambda e invocarla de forma manual mediante los datos del evento de muestra de Amazon S3. Esta prueba intermedia verifica el funcionamiento de la función.
- En segundo lugar, debe añadir la configuración de notificaciones al bucket de origen para que Amazon S3 pueda invocar la función de Lambda cuando detecte eventos de creación de objetos.

Paso siguiente

[Paso 1: Preparación \(p. 218\)](#)

Paso 1: Preparación

En esta sección, hará lo siguiente:

- Inscribirse en una cuenta de AWS y configurar la AWS CLI.
- Cree dos buckets ([*source*](#) y [*source*resized](#)) con un objeto .jpg de muestra ([HappyFace.jpg](#)) en el bucket de origen. Para obtener instrucciones, consulte el siguiente procedimiento.

Paso 1.1: Inscripción en AWS y configuración de la AWS CLI

Asegúrese de haber completado los pasos siguientes:

- Inscribirse en una cuenta de AWS y crear un usuario administrador en ella (denominado adminuser).
- Instalar y configurar la AWS CLI.

Para obtener instrucciones, consulte [Paso 1: Configuración de una cuenta de AWS y de la AWS CLI \(p. 199\)](#).

Paso 1.2: Creación de los buckets y carga de un objeto de muestra

Siga los pasos para crear buckets y cargar un objeto.

Important

Tanto el bucket de origen como la función de Lambda deben estar en la misma región de AWS. Además, el código de muestra utilizado para la función de Lambda también presupone que los dos buckets se encuentran en la misma región. En este tutorial, utilizamos la región `us-west-2`.

1. Utilizando la dirección URL de inicio de sesión de usuario de IAM, inicie sesión en la consola de Amazon S3 como adminuser.
2. Cree dos buckets. El nombre del bucket de destino debe ser [*source*](#) seguido de [*resized*](#), donde [*source*](#) es el nombre del bucket que desea utilizar para el origen. Por ejemplo, `mybucket` y `mybucketresized`.

Para obtener instrucciones, consulte [Crear un bucket](#) en la Guía de introducción a Amazon Simple Storage Service.

3. En el bucket de origen, cargue un objeto .jpg denominado `HappyFace.jpg`.

Al invocar la función de Lambda manualmente antes de conectarse a Amazon S3, debe pasar los datos del evento de muestra a la función que especifica el bucket de origen, y `HappyFace.jpg` como objeto recién creado, por lo que necesita crear este objeto de muestra primero.

Paso siguiente

[Paso 2: Creación de una función de Lambda e invocación manual \(utilizando datos del evento de muestra\) \(p. 219\)](#)

Paso 2: Creación de una función de Lambda e invocación manual (utilizando datos del evento de muestra)

En esta sección, hará lo siguiente:

- Crear un paquete de implementación de la función de Lambda utilizando el código de muestra proporcionado.

Note

Para ver más ejemplos de uso de otros servicios de AWS dentro de la función, incluyendo llamadas a otras funciones de Lambda, consulte [AWS SDK para JavaScript](#)

- Crear un rol de IAM (rol de ejecución). En el momento de cargar el paquete de implementación, debe especificar un rol de IAM (rol de ejecución) que Lambda puede asumir para ejecutar la función en su nombre.
- Cree la función de Lambda cargando el paquete de implementación y, a continuación, pruébelas invocándolas de forma manual utilizando datos de eventos de Amazon S3.

Temas

- [Paso 2.1: Creación de un paquete de implementación \(p. 219\)](#)
- [Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 226\)](#)
- [Paso 2.3: Creación y prueba manual de la función de Lambda \(p. 227\)](#)

Paso 2.1: Creación de un paquete de implementación

En la lista Filter View, elija el lenguaje que desea utilizar para la función de Lambda. Aparece la sección correspondiente con código e instrucciones específicas para crear un paquete de implementación.

[Node.js](#)

El paquete de implementación es un archivo .zip que contiene el código y las dependencias de la función de Lambda.

1. Cree una carpeta (`examplefolder`) y, a continuación, cree una subcarpeta (`node_modules`).
2. Instale la plataforma Node.js. Para obtener más información, consulte el sitio web de [Node.js](#).
3. Instale las dependencias. Los ejemplos de código utilizan las siguientes bibliotecas:
 - AWS SDK para JavaScript en Node.js
 - gm, GraphicsMagick para node.js
 - Módulo de utilidades asíncronas

El tiempo de ejecución de AWS Lambda ya tiene el AWS SDK para JavaScript en Node.js, por lo que solo tiene que instalar las demás bibliotecas. Abra un símbolo del sistema, vaya a `examplefolder` e instale las bibliotecas utilizando el comando `npm`, que forma parte de Node.js.

```
npm install async gm
```

4. Abrir un editor de texto y, a continuación, copie el siguiente código.

```
// 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(/^.([^.]*\.(jpe?g|png))$/);
    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

El código de muestra es compatible con los tiempos de ejecución v6.10 o v4.3 de Node.js. Para obtener más información, consulte [Modelo de programación \(Node.js\) \(p. 9\)](#).

5. Revise el código anterior y observe lo siguiente:

- La función conoce el nombre del bucket de origen y el nombre de la clave de objeto a partir de los datos del evento que recibe como parámetros. Si el objeto es una imagen .jpg, el código crea una miniatura y la guarda en el bucket de destino.
- El código presupone que el bucket de destino existe y que su nombre es una concatenación del nombre del bucket de origen seguido por la cadena `resized`. Por ejemplo, si el bucket de origen identificado en los datos del evento es `examplebucket`, el código presupone que existe un bucket de destino denominado `examplebucketresized`.

- Para generar el nombre de la clave de la miniatura que crea, el código utiliza la cadena `resized-` seguida del nombre de la clave del objeto de origen. Por ejemplo, si la clave del objeto de origen es `sample.jpg`, el código crea una objeto de miniatura cuya clave es `resized-sample.jpg`.
6. Guarde el archivo como `CreateThumbnail.js` en `examplefolder`. Cuando realice este paso, tendrá la siguiente estructura de carpetas:

```
CreateThumbnail.js
/node_modules/gm
/node_modules/async
```

7. Comprima el archivo `CreateThumbnail.js` y la carpeta `node_modules` como `CreateThumbnail.zip`.

Este es el paquete de implementación de la función de Lambda.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 226\)](#)

Java

A continuación, se muestra un ejemplo de código Java que lee los eventos entrantes de Amazon S3 y crea una miniatura. Observe que implementa la interfaz `RequestHandler` proporcionada en la biblioteca `aws-lambda-java-core`. Por lo tanto, en el momento de crear una función de Lambda debe especificar la clase como controlador (es decir, `example.S3EventProcessorCreateThumbnail`). Para obtener más información acerca de cómo utilizar interfaces para proporcionar un controlador, consulte [Uso de interfaces predefinidas para la creación de un controlador \(Java\) \(p. 32\)](#).

El tipo `S3Event` utilizado por el controlador como tipo de entrada es una de las clases predefinidas en la biblioteca `aws-lambda-java-events` que proporciona métodos para que se pueda leer fácilmente la información del evento entrante de Amazon S3. El controlador devuelve una cadena como salida.

```
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 invoca la función de Lambda mediante el tipo de invocación `Event`, en el que AWS Lambda ejecuta el código de forma asíncrona. No importa lo que se devuelva. Sin embargo, en este caso estamos implementando una interfaz que obliga a especificar un tipo de retorno, por lo que en este ejemplo el controlador utiliza `String` como tipo de retorno.

Utilizando el código anterior (en un archivo denominado `S3EventProcessorCreateThumbnail.java`), cree un paquete de implementación. Asegúrese de que añade las siguientes dependencias:

- `aws-lambda-java-core`
- `aws-lambda-java-events`

Se encuentran en [aws-lambda-java-libs](#).

Para obtener más información, consulte [Modelo de programación para crear funciones de Lambda en Java \(p. 25\)](#).

El paquete de implementación puede ser un archivo `.zip` o `.jar` independiente. Puede utilizar cualquier herramienta de compilación y compresión con la que esté familiarizado para crear un paquete de implementación. Para ver ejemplos de cómo utilizar la herramienta de compilación Maven para crear un archivo `.jar` independiente, consulte [Creación de un paquete de implementación .jar utilizando Maven sin ningún IDE \(Java\) \(p. 70\)](#) y [Creación de un paquete de implementación .jar utilizando Maven y el IDE de Eclipse \(Java\) \(p. 72\)](#). Para ver un ejemplo de cómo utilizar la herramienta de compilación Gradle para crear un archivo `.zip`, consulte [Creación de un paquete de implementación .zip \(Java\) \(p. 75\)](#).

Después de verificar que se ha creado el paquete de implementación, vaya al paso siguiente para crear un rol de IAM (rol de ejecución). Debe especificar este rol en el momento de crear la función de Lambda.

Paso siguiente

Paso 2.2: Creación del rol de ejecución (rol de IAM) (p. 226)

Python

En esta sección, creará una función de ejemplo en Python e instalará las dependencias. El código de muestra es compatible con las versiones 3.6 o 2.7 del tiempo de ejecución de Python. En los pasos se supone que se usa el tiempo de ejecución 3.6, pero puede utilizar cualquiera de los dos.

1. Abra un editor de texto y copie el siguiente código. El código carga la imagen redimensionada en otro bucket con el mismo nombre de imagen, como se muestra a continuación:

```
source-bucket/image.png -> source-bucketresized/image.png
```

Note

La instrucción `from __future__ import print_function` permite escribir código compatible con Python versión 2 o 3. Si utiliza la versión 3.6 del tiempo de ejecución, no es necesario incluirla.

```
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. Guarde el archivo como `CreateThumbnail.py`.
3. Si el código fuente está en un host local y cópielo.

```
scp -i key.pem /path/to/my_code.py ec2-user@public-ip-address:~/CreateThumbnail.py
```

4. Conéctese a una instancia de Amazon Linux de 64 bits mediante SSH.

```
ssh -i key.pem ec2-user@public-ip-address
```

5. Instale Python 3.6 y virtualenv mediante los siguientes pasos:

```
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. Elija el entorno virtual que se instaló a través de pip3.

```
/usr/local/bin/virtualenv ~/shrink_venv
```

```
source ~/shrink_venv/bin/activate
```

7. Instale las bibliotecas en el entorno virtual.

```
pip install Pillow
```

```
pip install boto3
```

Note

AWS Lambda incluye el AWS SDK para Python (Boto 3), por lo que no es necesario que lo incluya en el paquete de implementación, pero puede incluirlo si lo desea para realizar pruebas locales.

8. Añada el contenido de los site-packages `lib` y `lib64` al archivo `.zip`. Tenga en cuenta que los siguientes pasos dan por hecho que utiliza la versión 3.6 del tiempo de ejecución de Python. Si utiliza la versión 2.7, tendrá que actualizarlos según sea necesario.

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

```
zip -r9 ~/CreateThumbnail.zip *
```

Note

Para incluir todos los archivos ocultos, utilice la siguiente opción: `zip -r9 ~/CreateThumbnail.zip .`

9. Añada el código de Python al archivo `.zip`.

```
cd ~
```

```
zip -g CreateThumbnail.zip CreateThumbnail.py
```

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 226\)](#)

Paso 2.2: Creación del rol de ejecución (rol de IAM)

En esta sección, creará un rol de IAM con el tipo de rol y la política de permisos de acceso predefinidos que se indican a continuación:

- Rol de servicio de AWS del tipo AWS Lambda: este rol concede permisos a AWS Lambda para asumir el rol.
- `AWSLambdaExecute`: es la política de permisos de acceso que se asocia al rol.
- Introduzca un nombre de rol en el campo `Role name` y, a continuación, elija `Create role`.

Para obtener más información acerca de los roles de IAM, consulte [IAM Roles](#) en la Guía del usuario de IAM. Utilice el siguiente procedimiento para crear el rol de IAM.

Para crear un rol de IAM (rol de ejecución)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.

2. Siga los pasos de [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM para crear un rol de IAM (rol de ejecución). Cuando siga los pasos para crear un rol, tenga en cuenta lo siguiente:
 - En Role Name, utilice un nombre que sea único dentro de la cuenta de AWS (por ejemplo, lambda-s3-execution-role).
 - En Select Role Type, elija AWS Service Roles, seguido de AWS Lambda. Esto garantiza que los permisos del servicio de AWS Lambda asuman el rol.
 - En Attach Policy, elija AWSLambdaBasicExecuteRole.
3. Anote el ARN del rol. Lo necesitará en el siguiente paso al crear la función de Lambda.

Paso siguiente

[Paso 2.3: Creación y prueba manual de la función de Lambda \(p. 227\)](#)

Paso 2.3: Creación y prueba manual de la función de Lambda

En esta sección, hará lo siguiente:

- Crear una función de Lambda cargando el paquete de implementación.
- Pruebe la función de Lambda invocándola de forma manual y utilizando los datos de un evento de muestra de Amazon S3.

Paso 2.3.1: Creación de la función de Lambda (carga del paquete de implementación)

En este paso, cargará el paquete de implementación utilizando la AWS CLI.

1. En el símbolo del sistema, ejecute el siguiente comando `create-function` de la AWS CLI de Lambda utilizando `adminuser` para `--profile`. Debe actualizar el comando proporcionando la ruta del archivo .zip y el ARN del rol de ejecución. Para el parámetro de tiempo de ejecución, seleccione `nodejs6.10`, `nodejs4.3`, `python3.6`, `python2.7` o `java8`, en función del código de muestra que haya seleccionado al crear el paquete de implementación.

```
$ aws lambda create-function \
--region us-west-2 \
--function-name CreateThumbnail \
--zip-file fileb://file-path/CreateThumbnail.zip \
--role role-arn \
--handler CreateThumbnail.handler \
--runtime runtime \
--profile adminuser \
--timeout 10 \
--memory-size 1024
```

Si lo desea, puede cargar el archivo .zip en un bucket de Amazon S3 en la misma región de AWS y, a continuación, especificar el nombre del objeto y el bucket en el comando anterior. Debe reemplazar el parámetro `--zip-file` por el parámetro `--code`, como se muestra a continuación:

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

2. Anote el ARN de la función. Lo necesitará en la sección siguiente al añadir la configuración de notificaciones al bucket de Amazon S3.
3. (Opcional) El comando anterior especifica un tiempo de espera de 10 segundos como configuración de la función. En función del tamaño de los objetos que cargue, es posible que necesite para aumentar el tiempo de espera utilizando el siguiente comando de la AWS CLI.

```
$ aws lambda update-function-configuration \
--function-name CreateThumbnail \
--region us-west-2 \
--timeout timeout-in-seconds \
--profile adminuser
```

Note

Puede crear una función de Lambda utilizando la consola de AWS Lambda, en cuyo caso debe tomar nota del valor de los parámetros del comando `create-function` de la AWS CLI. Debe proporcionar los mismos valores en la interfaz de usuario de la consola.

Paso 2.3.2: Prueba de la función de Lambda (invocación manual)

En este paso, invocará la función de Lambda de forma manual con los datos de un evento de muestra de Amazon S3. Puede probar la función utilizando la Consola de administración de AWS o la AWS CLI.

Para probar la función de Lambda (consola)

1. Siga los pasos de la introducción para crear e invocar la función de Lambda en [Paso 2.2: Invocación manual de la función de Lambda y verificación de los resultados, los logs y las métricas \(p. 206\)](#). Para las pruebas, elija el evento de muestra S3 Put en Sample event template.
2. Compruebe que se ha creado la miniatura en el bucket de destino y monitorice la actividad de la función de Lambda en la consola de AWS Lambda como se indica a continuación:
 - La consola de AWS Lambda muestra una representación gráfica de algunas de las métricas de CloudWatch en la sección Cloudwatch Metrics at a glance para la función.
 - Para cada gráfico, también puede hacer clic en el enlace logs para ver directamente los CloudWatch Logs.

Para probar la función de Lambda (AWS CLI)

1. Guarde los siguientes datos del evento de muestra de Amazon S3 en un archivo denominado `input.txt`. Deberá actualizar el JSON proporcionando el nombre de `sourcebucket` y una clave de objeto `.jpg`.

```
{
    "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": "096ffKKXTRTtl3on89fv0.nfljtsv6qko"
}
}
```

- Ejecute el siguiente comando `invoke` de la CLI de Lambda para invocar la función. Tenga en cuenta que el comando solicita la ejecución asíncrona. También puede invocarla de forma síncrona especificando `RequestResponse` como valor del parámetro `invocation-type`.

```
$ aws lambda invoke \
--invocation-type Event \
--function-name CreateThumbnail \
--region us-west-2 \
--payload file:///file-path/inputfile.txt \
--profile adminuser \
outputfile.txt
```

Note

Puede invocar esta función, debido a que utiliza sus propias credenciales para invocar su propia función. En la siguiente sección, configurará Amazon S3 para invocar esta función en su nombre, lo que requiere que añada permisos a la política de acceso asociada a la función de Lambda con objeto de conceder permisos a Amazon S3 para invocar la función.

- Compruebe que se ha creado la miniatura en el bucket de destino y monitorice la actividad de la función de Lambda en la consola de AWS Lambda como se indica a continuación:
 - La consola de AWS Lambda muestra una representación gráfica de algunas de las métricas de CloudWatch en la sección Cloudwatch Metrics at a glance para la función.
 - Para cada gráfico, también puede hacer clic en el enlace logs para ver directamente los CloudWatch Logs.

Paso siguiente

[Paso 3: Adición de un origen de eventos \(configuración de Amazon S3 para que publique eventos\) \(p. 229\)](#)

Paso 3: Adición de un origen de eventos (configuración de Amazon S3 para que publique eventos)

En este paso, añadirá la configuración restante, de manera que Amazon S3 pueda publicar eventos de creación de objetos en AWS Lambda e invocar la función de Lambda. En este paso, hará lo siguiente:

- Añadir permisos a la política de acceso de la función de Lambda para permitir que Amazon S3 invoque la función.

- Añadir la configuración de notificaciones al bucket de origen. En la configuración de notificaciones, debe proporcionar lo siguiente:
 - Tipo de evento para el que desea que Amazon S3 publique eventos. Para este tutorial, debe especificar el tipo de evento `s3:ObjectCreated:*` con objeto de que Amazon S3 publique eventos cuando se creen objetos.
 - Función de Lambda que se debe invocar.

Paso 3.1: Adición de permisos a la política de permisos de acceso de la función de Lambda

1. Ejecute el siguiente comando `add-permission` de la CLI de Lambda para conceder a la entidad principal del servicio de Amazon S3 (`s3.amazonaws.com`) permisos para realizar la acción `lambda:InvokeFunction`. Tenga en cuenta únicamente que se concede permiso a Amazon S3 para invocar la función si se cumplen las siguientes condiciones:
 - Se detecta un evento de creación de objeto en un bucket específico.
 - El bucket pertenece a una cuenta determinada de AWS. Si el propietario de un bucket lo elimina, otra cuenta de AWS puede crear un bucket con el mismo nombre. Esta condición garantiza que solo una cuenta determinada de AWS pueda invocar la función de Lambda.

```
$ aws lambda add-permission \
--function-name CreateThumbnail \
--region us-west-2 \
--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
```

2. Verifique la política de acceso de la función ejecutando el comando `get-policy` de la AWS CLI.

```
$ aws lambda get-policy \
--function-name function-name \
--profile adminuser
```

Pasos 3.2: Configuración de las notificaciones en el bucket

Añada la configuración de notificaciones al bucket de origen para solicitar a Amazon S3 que publique los eventos de creación de objetos en Lambda. En la configuración, especifique lo siguiente:

- Tipo de evento: para este tutorial, seleccione el tipo de evento `ObjectCreated (All)` de Amazon S3.
- Función de Lambda: es la función de Lambda que desea que Amazon S3 invoque.

Para obtener instrucciones acerca de cómo añadir una configuración de notificaciones a un bucket, consulte [Habilitación de notificaciones de eventos](#) en la Guía del usuario de la consola de Amazon Simple Storage Service.

Paso 3.3: Prueba de la configuración

¡Ya ha terminado! Ahora, `adminuser` puede probar la configuración como se indica a continuación:

1. Cargue objetos `.jpg` o `.png` en el bucket de origen mediante la consola de Amazon S3.
2. Verifique que se crea la miniatura en el bucket de destino mediante la función `CreateThumbnail`.

3. adminuser también puede verificar los CloudWatch Logs. Puede monitorizar la actividad de la función de Lambda en la consola de AWS Lambda. Por ejemplo, elija el enlace logs en la consola para ver los logs, incluidos los que ha escrito la función en CloudWatch Logs.

Paso 4: Implementación con AWS SAM y AWS CloudFormation

En la sección anterior, ha utilizado la API de AWS Lambda para crear y actualizar una función de Lambda proporcionando un paquete de implementación en un archivo ZIP. Sin embargo, puede que este mecanismo no sea conveniente para automatizar los pasos de implementación de las funciones, o para coordinar implementaciones y actualizaciones con otros elementos de una aplicación sin servidor, como los orígenes de eventos y los recursos posteriores.

Puede utilizar AWS CloudFormation para especificar, implementar y configurar fácilmente aplicaciones sin servidor. AWS CloudFormation es un servicio que le ayuda a modelar y configurar recursos de Amazon Web Services, por lo que podrá dedicar menos tiempo a la administración de dichos recursos y más tiempo a centrarse en las aplicaciones que se ejecutan en AWS. Puede crear una plantilla que describa todos los recursos de AWS que desea (como funciones de Lambda y tablas de DynamoDB), y AWS CloudFormation se encargará del aprovisionamiento y la configuración de dichos recursos.

Además, puede utilizar AWS Serverless Application Model para expresar los recursos que componen la aplicación sin servidor. Estos tipos de recursos, como las API y las funciones de Lambda, son totalmente compatibles con AWS CloudFormation, y facilitan la tarea de definir e implementar aplicaciones sin servidor.

Para obtener más información, consulte [Implementación de aplicaciones basadas en Lambda \(p. 156\)](#).

Especificaciones para la aplicación de miniaturas de Amazon S3

A continuación encontrará la plantilla SAM para esta aplicación. Copie el texto siguiente en un archivo .yaml y guárdelo junto al paquete ZIP que ha creado anteriormente. Tenga en cuenta que los valores de los parámetros Handler y Runtime deben coincidir con los que utilizó cuando creó la función en la sección anterior.

```
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
```

Implementación de la aplicación sin servidor

Para obtener información acerca de cómo empaquetar e implementar la aplicación sin servidor utilizando los comandos package y deploy, consulte [Empaquetado e implementación \(p. 162\)](#).

Uso de AWS Lambda con Kinesis

Puede crear un flujo de Kinesis para capturar de forma continua terabytes de datos por hora provenientes de cientos de miles de orígenes, como secuencias de clics de sitios web, transacciones financieras, fuentes de redes sociales, logs de TI y eventos de seguimiento de ubicación. Para obtener más información, consulte [Kinesis](#).

Puede suscribir funciones de Lambda para que lean de forma automática lotes de registros procedentes de un flujo de Kinesis y los procesen si se detectan registros en el flujo. A continuación, AWS Lambda sondea el flujo de forma periódica (una vez por segundo) buscando registros nuevos.

Tenga en cuenta lo siguiente sobre cómo funciona la integración entre Kinesis y AWS Lambda:

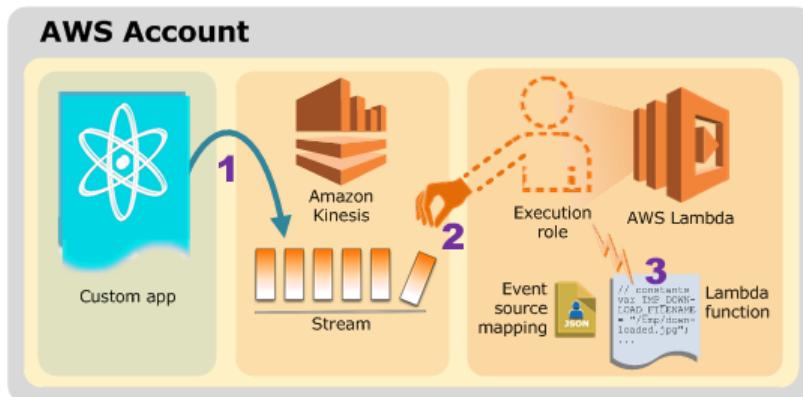
- **Modelo basado en flujos:** este es un modelo (consulte [Mapeo de orígenes de eventos \(p. 134\)](#)) en el que AWS Lambda sondea el flujo y, cuando detecta nuevos registros, invoca la función de Lambda pasando los registros nuevos como parámetro.

En un modelo basado en flujos, el mapeo de origen de eventos se mantiene en AWS Lambda. El mapeo de origen de eventos describe el mapeo entre los flujos y las funciones de Lambda. AWS Lambda proporciona una API ([CreateEventSourceMapping \(p. 380\)](#)) que puede utilizarse para crear el mapeo. También se puede utilizar la consola de AWS Lambda para crear mapeos de orígenes de eventos.

- **Invocación síncrona:** AWS Lambda invoca una función de Lambda mediante el tipo de invocación `RequestResponse` (invocación síncrona) sondeando el flujo de Kinesis. Para obtener más información acerca de los tipos de invocación, consulte [Tipos de invocación \(p. 4\)](#).
- **Estructura del evento:** el evento que recibe la función de Lambda es una colección de registros que AWS Lambda lee del flujo. Al configurar el mapeo de origen de eventos, el tamaño del lote que debe especificar es el número máximo de registros que puede recibir la función de Lambda por invocación.

Independientemente de qué invoque una función de Lambda AWS Lambda siempre ejecuta una función de Lambda en nombre del usuario. Si la función de Lambda necesita obtener acceso a recursos de AWS, debe conceder los permisos pertinentes para permitir dicho acceso. También debe conceder permisos de AWS Lambda para sondear el flujo de Kinesis. Puede conceder todos estos permisos a un rol de IAM (rol de ejecución) que AWS Lambda pueda asumir para sondear el flujo y ejecutar la función de Lambda en su nombre. Primero debe crear este rol y, a continuación, activarlo en el momento de crear la función de Lambda. Para obtener más información, consulte [Administración de permisos mediante un rol de IAM \(rol de ejecución\) \(p. 193\)](#).

En el diagrama siguiente se ilustra el flujo de la aplicación:



1. Una aplicación personalizada escribe los registros en el flujo.
2. AWS Lambda sondea el flujo y, cuando detecta registros nuevos en él, llama a la función de Lambda.

3. AWS Lambda ejecuta la función de Lambda asumiendo el rol de ejecución que se especificó en el momento de crear la función de Lambda.

Para ver un tutorial que muestra una configuración de ejemplo, consulte [Tutorial: Uso de AWS Lambda con Kinesis \(p. 233\)](#).

Tutorial: Uso de AWS Lambda con Kinesis

En este tutorial, se crea una función de Lambda para consumir eventos de un flujo de Kinesis.

El tutorial se divide en dos secciones principales:

- En primer lugar, realice los pasos de configuración necesarios para crear una función de Lambda y, a continuación, pruébela invocándola de forma manual utilizando los datos del evento de muestra (no necesita un flujo de Kinesis).
- En segundo lugar, cree un flujo (origen de eventos) de Kinesis. Debe añadir un mapeo de origen de eventos en AWS Lambda para asociar el flujo a la función de Lambda. AWS Lambda comienza a sondear el flujo. Añada registros de prueba al flujo mediante la API de Kinesis y, a continuación, compruebe que AWS Lambda ha ejecutado la función de Lambda.

Important

Tanto la función de Lambda como el flujo de Kinesis deben estar en la misma región de AWS. En este tutorial se supone que crea estos recursos en la región `us-west-2`.

En este tutorial, utilice la AWS Command Line Interface para realizar operaciones de AWS Lambda, como crear una función de Lambda, crear un flujo o añadir registros al flujo. Utilice la consola de AWS Lambda para invocar la función manualmente antes de crear un flujo de Kinesis. Verifique los valores de retorno y los logs en la interfaz de usuario de la consola.

Paso siguiente

[Paso 1: Preparación \(p. 233\)](#)

Paso 1: Preparación

Asegúrese de haber completado los pasos siguientes:

- Inscribirse en una cuenta de AWS y crear un usuario administrador en ella.
- Instalar y configurar la AWS CLI.

Para obtener instrucciones, consulte [Paso 1: Configuración de una cuenta de AWS y de la AWS CLI \(p. 199\)](#).

Paso siguiente

[Paso 2: Creación de una función de Lambda e invocación manual \(utilizando datos del evento de muestra\) \(p. 233\)](#)

Paso 2: Creación de una función de Lambda e invocación manual (utilizando datos del evento de muestra)

En esta sección, hará lo siguiente:

- Crear un paquete de implementación de la función de Lambda utilizando el código de muestra proporcionado. El código de muestra de la función de Lambda que utilizará para procesar los eventos de Kinesis se proporciona en varios lenguajes. Seleccione uno de los lenguajes y siga las instrucciones correspondientes para crear un paquete de implementación.

Note

Para ver más ejemplos de uso de otros servicios de AWS dentro de la función, incluyendo llamadas a otras funciones de Lambda, consulte [AWS SDK para JavaScript](#)

- Crear un rol de IAM (rol de ejecución). En el momento de cargar el paquete de implementación, debe especificar un rol de IAM (rol de ejecución) que Lambda puede asumir para ejecutar la función en su nombre.
- Cree la función de Lambda cargando el paquete de implementación y, a continuación, pruébelala invocándola de forma manual utilizando los datos del evento de muestra de Kinesis.

Temas

- [Paso 2.1: Creación de un paquete de implementación \(p. 234\)](#)
- [Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 237\)](#)
- [Paso 2.3: Creación y prueba manual de la función de Lambda \(p. 238\)](#)

Paso 2.1: Creación de un paquete de implementación

En la lista Filter View, elija el lenguaje que desea utilizar para la función de Lambda. Aparece la sección correspondiente con código e instrucciones específicas para crear un paquete de implementación.

[Node.js](#)

A continuación se muestra código de ejemplo de Node.js que recibe registros de eventos de Kinesis como entrada y los procesa. Con fines ilustrativos, el código escribe parte de los datos de los eventos de entrada en CloudWatch Logs.

Siga las instrucciones para crear un paquete de implementación para la función de AWS Lambda.

1. Abrir un editor de texto y, a continuación, copie el siguiente código.

```
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

El código de muestra es compatible con los tiempos de ejecución v6.10 o v4.3 de Node.js. Para obtener más información, consulte [Modelo de programación \(Node.js\) \(p. 9\)](#).

2. Guarde el archivo como `ProcessKinesisRecords.js`.
3. Comprima el archivo `ProcessKinesisRecords.js` como `ProcessKinesisRecords.zip`.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 237\)](#)

Java

A continuación se muestra código de ejemplo de Java que recibe datos de registros de eventos de Kinesis como entrada y los procesa. Con fines ilustrativos, el código escribe parte de los datos de los eventos de entrada en CloudWatch Logs.

En el código, `recordHandler` es el controlador. Este controlador utiliza la clase `KinesisEvent` predefinida, que se define en la biblioteca `aws-lambda-java-events`.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.KinesisEventRecord;

public class 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;
}
}
```

Si el controlador vuelve normalmente sin excepciones, Lambda considera que el lote de registros de entrada se ha procesado correctamente y empieza a leer nuevos registros en el flujo. Si el controlador genera una excepción, Lambda considera que el lote de registros de entrada no se ha procesado e invoca de nuevo la función con el mismo lote de registros.

Utilizando el código anterior (en un archivo denominado `ProcessKinesisEvents.java`), cree un paquete de implementación. Asegúrese de que añade las siguientes dependencias:

- `aws-lambda-java-core`
- `aws-lambda-java-events`

Para obtener más información, consulte [Modelo de programación para crear funciones de Lambda en Java \(p. 25\)](#).

El paquete de implementación puede ser un archivo .zip o .jar independiente. Puede utilizar cualquier herramienta de compilación y compresión con la que esté familiarizado para crear un paquete de implementación. Para ver ejemplos de cómo utilizar la herramienta de compilación Maven para crear un archivo .jar independiente, consulte [Creación de un paquete de implementación .jar utilizando Maven sin ningún IDE \(Java\) \(p. 70\)](#) y [Creación de un paquete de implementación .jar utilizando Maven y el IDE de Eclipse \(Java\) \(p. 72\)](#). Para ver un ejemplo de cómo utilizar la herramienta de compilación Gradle para crear un archivo .zip, consulte [Creación de un paquete de implementación .zip \(Java\) \(p. 75\)](#).

Después de verificar que se ha creado el paquete de implementación, vaya al paso siguiente para crear un rol de IAM (rol de ejecución). Debe especificar este rol en el momento de crear la función de Lambda.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 237\)](#)

C#

A continuación se muestra código de ejemplo de C# que recibe datos de registros de eventos de Kinesis como entrada y los procesa. Con fines ilustrativos, el código escribe parte de los datos de los eventos de entrada en CloudWatch Logs.

En el código, `HandleKinesisRecord` es el controlador. Este controlador utiliza la clase `KinesisEvent` predefinida, que se define en la biblioteca `Amazon.Lambda.KinesisEvents`.

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

Para crear un paquete de implementación, siga los pasos que se indican en [CLI de .NET Core \(p. 65\)](#). Al hacerlo, tenga en cuenta lo siguiente después de crear el proyecto de .NET:

- Cambie el nombre del archivo `Program.cs` predeterminado por un nombre de su elección, como, por ejemplo, `ProcessingKinesisEvents.cs`.
- Sustituya el contenido predeterminado del archivo `Program.cs` cuyo nombre ha cambiado por el ejemplo de código anterior.
- En el archivo `project.json`, asegúrese de que las referencias siguientes están incluidas en el nodo `dependencies`.
 - `"Amazon.Lambda.Core": "1.0.0-*"`
 - `"Amazon.Lambda.KinesisEvents": "1.0.0-*"`

- "Amazon.Lambda.Serialization.Json": "1.0.0-*"

Después de verificar que se ha creado el paquete de implementación, vaya al paso siguiente para crear un rol de IAM (rol de ejecución). Debe especificar este rol en el momento de crear la función de Lambda.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 237\)](#)

Python

A continuación se muestra código de ejemplo de Python que recibe datos de registros de eventos de Kinesis como entrada y los procesa. Con fines ilustrativos, el código escribe parte de los datos de los eventos de entrada en CloudWatch Logs.

Siga las instrucciones para crear un paquete de implementación para la función de AWS Lambda.

1. Abrir un editor de texto y, a continuación, copie el siguiente código.

Note

La instrucción `from __future__ import print_function` permite escribir código compatible con Python versión 2 o 3. Si utiliza la versión 3.6 del tiempo de ejecución, no es necesario incluirla.

```
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. Guarde el archivo como `ProcessKinesisRecords.py`.
3. Comprima el archivo `ProcessKinesisRecords.py` como `ProcessKinesisRecords.zip`.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 237\)](#)

Paso 2.2: Creación del rol de ejecución (rol de IAM)

En esta sección, creará un rol de IAM con el tipo de rol y la política de acceso predefinidos que se indican a continuación:

- Rol de servicio de AWS del tipo AWS Lambda: este rol concede permisos a AWS Lambda para asumir el rol.
- `AWSLambdaKinesisExecutionRole`: es la política de permisos de acceso que se asocia al rol.

Para obtener más información acerca de los roles de IAM, consulte [IAM Roles](#) en la Guía del usuario de IAM. Utilice el siguiente procedimiento para crear el rol de IAM.

Para crear un rol de IAM (rol de ejecución)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.

2. Siga los pasos de [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM para crear un rol de IAM (rol de ejecución). Cuando siga los pasos para crear un rol, tenga en cuenta lo siguiente:
 - En Role Name, utilice un nombre que sea único dentro de la cuenta de AWS (por ejemplo, lambda-kinesis-execution-role).
 - En Select Role Type, elija AWS Service Roles, seguido de AWS Lambda. Esto garantiza que los permisos del servicio de AWS Lambda asuman el rol.
 - En Attach Policy, seleccione AWSLambdaKinesisExecutionRole. Los permisos de esta política son suficientes para la función de Lambda de este tutorial.
3. Anote el ARN del rol. Lo necesitará en el siguiente paso al crear la función de Lambda.

Paso siguiente

[Paso 2.3: Creación y prueba manual de la función de Lambda \(p. 238\)](#)

Paso 2.3: Creación y prueba manual de la función de Lambda

En esta sección, hará lo siguiente:

- Crear una función de Lambda cargando el paquete de implementación.
- Probar la función de Lambda invocándola manualmente. En lugar de crear un origen de eventos, utilice los datos del evento de muestra de Kinesis.

En la siguiente sección, va a crear un flujo de Kinesis y va a probar la experiencia integral.

[Paso 2.3.1: Creación de una función de Lambda \(carga del paquete de implementación\)](#)

En este paso, cargará el paquete de implementación utilizando la AWS CLI.

En el símbolo del sistema, ejecute el siguiente comando `create-function` de la CLI de Lambda utilizando el perfil adminuser. Para obtener más información acerca de esta configuración, consulte [Configuración de la AWS CLI](#).

Debe actualizar el comando proporcionando la ruta del archivo .zip y el ARN del rol de ejecución. El valor del parámetro `--runtime` puede ser `python3.6`, `python2.7`, `nodejs6.10`, `nodejs4.3` o `java8`, dependiendo del lenguaje que utilice para crear el código.

```
$ aws lambda create-function \
--region us-west-2 \
--function-name ProcessKinesisRecords \
--zip-file fileb://file-path/ProcessKinesisRecords.zip \
--role execution-role-arn \
--handler handler \
--runtime runtime-value \
--profile adminuser
```

El valor del parámetro `--handler` para Java debe ser `example.ProcessKinesisRecords::recordHandler`. Para Node.js, debe ser `ProcessKinesisRecords.handler` y para Python debe ser `ProcessKinesisRecords.lambda_handler`.

Si lo desea, puede cargar el archivo .zip en un bucket de Amazon S3 en la misma región de AWS y, a continuación, especificar el nombre del objeto y el bucket en el comando anterior. Debe reemplazar el parámetro `--zip-file` por el parámetro `--code`, como se muestra a continuación:

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

Note

Puede crear una función de Lambda utilizando la consola de AWS Lambda, en cuyo caso debe tomar nota del valor de los parámetros del comando `create-function` de la AWS CLI. Debe proporcionar los mismos valores en la interfaz de usuario de la consola.

Paso 2.3.2: Prueba de la función de Lambda (invocación manual)

Invoque la función de forma manual utilizando los datos del evento de muestra de Kinesis. Recomendamos que invoque la función utilizando la consola, ya que la interfaz de usuario de la consola proporciona una interfaz fácil de utilizar para examinar los resultados de la ejecución, incluido el resumen de ejecución, los logs escritos por el código y los resultados devueltos por la función (dado que la consola siempre realiza una ejecución síncrona, invoca la función de Lambda utilizando el tipo de invocación `RequestResponse`).

Para probar la función de Lambda (consola)

1. Siga los pasos de la introducción para crear e invocar la función de Lambda en [Paso 2.2: Invocación manual de la función de Lambda y verificación de los resultados, los logs y las métricas \(p. 206\)](#). Para las pruebas, elija el evento de muestra Kinesis en Sample event template.
2. Verifique los resultados en la consola.

Para probar la función de Lambda (AWS CLI)

1. Copie el siguiente JSON en un archivo y guárdearlo como `input.txt`.

```
{  
    "Records": [  
        {  
            "kinesis": {  
                "partitionKey": "partitionKey-3",  
                "kinesisSchemaVersion": "1.0",  
                "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0IDEyMy4=",  
                "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. Ejecute el siguiente comando invoke:

```
$ aws lambda invoke \  
--invocation-type Event \  
--function-name ProcessKinesisRecords \  
--region us-west-2 \  
--payload file://file-path/input.txt \  
--profile adminuser  
outputfile.txt
```

Note

En este tutorial de ejemplo, el mensaje se guarda en el archivo `outputfile.txt`. Si solicita una ejecución síncrona (con el tipo de invocación `RequestResponse`), la función devuelve la cadena de mensaje en el cuerpo de la respuesta.

Para Node.js, podría ser uno de los siguientes (lo que se haya especificado en el código):

```
context.succeed("message")
context.fail("message")
context.done(null, "message")
```

Para Python o Java, es el mensaje de la instrucción `return`:

```
return "message"
```

Paso siguiente

[Paso 3: Adición de un origen de eventos \(creación y asociación de un flujo de Kinesis a la función de Lambda\) \(p. 240\)](#)

Paso 3: Adición de un origen de eventos (creación y asociación de un flujo de Kinesis a la función de Lambda)

En esta sección, creará un flujo de Kinesis y, a continuación, añadirá un origen de eventos a AWS Lambda para asociar el flujo de Kinesis a la función de Lambda.

Una vez creado un origen de eventos, AWS Lambda comienza a sondear el flujo. Seguidamente, deberá probar la configuración añadiendo eventos al flujo y verificando que AWS Lambda ejecuta la función de Lambda en su nombre:

Paso 3.1: Creación de un flujo de Kinesis

Utilice el siguiente comando `create-stream` de la CLI de Kinesis para crear un flujo.

```
$ aws kinesis create-stream \
--stream-name examplestream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

Ejecute el siguiente comando `describe-stream` de la AWS CLI de Kinesis para obtener el ARN del flujo.

```
$ aws kinesis describe-stream \
--stream-name examplestream \
--region us-west-2 \
--profile adminuser
```

Necesitará el ARN del flujo en el siguiente paso para asociar el flujo a la función de Lambda. El flujo tiene el siguiente formato:

```
arn:aws:kinesis:aws-region:account-id:stream/stream-name
```

Paso 3.2: Adición de un origen de eventos en AWS Lambda

Ejecute el siguiente comando de AWS CLI `add-event-source`. Una vez ejecutado el comando, anote el UUID. Necesitará este UUID para hacer referencia al origen de eventos en los comandos (por ejemplo, al eliminar el origen de eventos).

```
$ aws lambda create-event-source-mapping \
--region us-west-2 \
--function-name ProcessKinesisRecords \
--event-source kinesis-stream-arn \
--batch-size 100 \
--starting-position TRIM_HORIZON \
--profile adminuser
```

Para obtener una lista de mapeos de orígenes de eventos, ejecute el siguiente comando.

```
$ aws lambda list-event-source-mappings \
--region us-west-2 \
--function-name ProcessKinesisRecords \
--event-source kinesis-stream-arn \
--profile adminuser \
--debug
```

En la respuesta, puede comprobar que el valor de estado es enabled.

Note

Si desactiva el mapeo de origen de eventos, AWS Lambda deja de sondear el flujo de Kinesis. Si vuelve a activar el mapeo de origen de eventos, reanudará el sondeo desde el número de secuencia donde se detuvo, de modo que cada registro se procese antes de desactivar el mapeo o después de activarlo. Si el número de secuencia es inferior a TRIM_HORIZON, al volver a activarlo, el sondeo comenzará a partir de TRIM_HORIZON. Sin embargo, si crea un nuevo mapeo de origen de eventos, el sondeo siempre comenzará desde TRIM_HORIZON, LATEST o AT_TIMESTAMP, en función de la posición inicial que especifique. Esto se aplica incluso si elimina un mapeo de origen de eventos y crea uno con la misma configuración que el que ha eliminado.

Paso 3.3: Prueba de la configuración

¡Ya ha terminado! Ahora, adminuser puede probar la configuración como se indica a continuación:

1. Añada registros de eventos al flujo de Kinesis con el siguiente comando de la AWS CLI. El valor de --data es un valor codificado en base64 de la cadena "Hello, this is a test.". Puede ejecutar el mismo comando más de una vez para añadir varios registros al flujo.

```
$ aws kinesis put-record \
--stream-name examplestream \
--data "This is a test. final" \
--partition-key shardId-000000000000 \
--region us-west-2 \
--profile adminuser
```

2. AWS Lambda sondea el flujo y, cuando detecta actualizaciones en este, invoca la función de Lambda pasando los datos de eventos procedentes del flujo.

AWS Lambda asume el rol de ejecución para sondear el flujo. Ha concedido al rol permisos para las acciones de Kinesis necesarias para que AWS Lambda pueda sondear el flujo y leer eventos de este.

3. La función se ejecuta y añade logs al grupo de logs correspondiente a la función de Lambda en Amazon CloudWatch.

adminuser también puede verificar los logs mostrados en la consola de Amazon CloudWatch. Asegúrese de que comprueba los logs en la misma región de AWS en la que ha creado la función de Lambda.

Paso 4: Implementación con AWS SAM y AWS CloudFormation

En la sección anterior, ha utilizado la API de AWS Lambda para crear y actualizar una función de Lambda proporcionando un paquete de implementación en un archivo ZIP. Sin embargo, puede que este mecanismo no sea conveniente para automatizar los pasos de implementación de las funciones, o para coordinar implementaciones y actualizaciones con otros elementos de una aplicación sin servidor, como los orígenes de eventos y los recursos posteriores.

Puede utilizar AWS CloudFormation para especificar, implementar y configurar fácilmente aplicaciones sin servidor. AWS CloudFormation es un servicio que le ayuda a modelar y configurar recursos de Amazon Web Services, por lo que podrá dedicar menos tiempo a la administración de dichos recursos y más tiempo a centrarse en las aplicaciones que se ejecutan en AWS. Puede crear una plantilla que describa todos los recursos de AWS que desea (como funciones de Lambda y tablas de DynamoDB), y AWS CloudFormation se encargará del aprovisionamiento y la configuración de dichos recursos.

Además, puede utilizar AWS Serverless Application Model para expresar los recursos que componen la aplicación sin servidor. Estos tipos de recursos, como las API y las funciones de Lambda, son totalmente compatibles con AWS CloudFormation, y facilitan la tarea de definir e implementar aplicaciones sin servidor.

Para obtener más información, consulte [Implementación de aplicaciones basadas en Lambda \(p. 156\)](#).

Especificación para la aplicación de Kinesis

A continuación encontrará la plantilla SAM para esta aplicación. Copie el texto siguiente en un archivo .yaml y guárdelo junto al paquete ZIP que ha creado anteriormente. Tenga en cuenta que los valores de los parámetros Handler y Runtime deben coincidir con los que utilizó cuando creó la función en la sección anterior.

```
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
```

Implementación de la aplicación sin servidor

Para obtener información acerca de cómo empaquetar e implementar la aplicación sin servidor utilizando los comandos package y deploy, consulte [Empaquetado e implementación \(p. 162\)](#).

Uso de AWS Lambda con Amazon DynamoDB

Puede utilizar funciones de Lambda como disparadores para una tabla de Amazon DynamoDB. Los disparadores son acciones personalizadas que se llevan a cabo en respuesta a actualizaciones realizadas en la tabla de DynamoDB. Para crear un disparador, primero debe habilitar Amazon DynamoDB Streams para la tabla. A continuación, debe escribir una función de Lambda para procesar las actualizaciones publicadas en el flujo.

Tenga en cuenta lo siguiente sobre cómo funciona la integración entre Amazon DynamoDB y AWS Lambda:

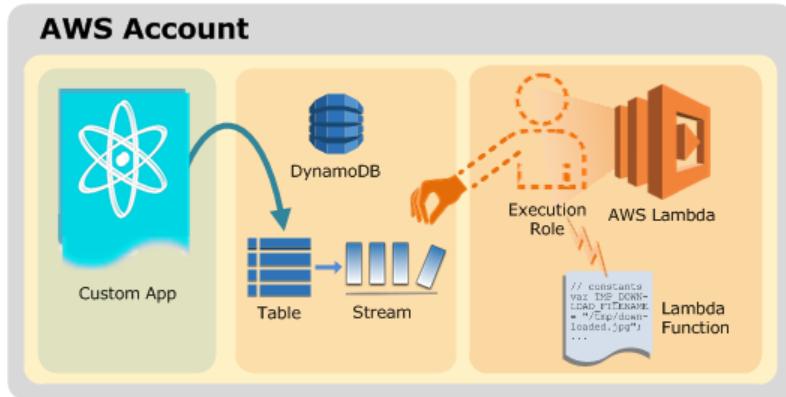
- **Modelo basado en flujos:** este es un modelo (consulte [Mapeo de orígenes de eventos \(p. 134\)](#)) en el que AWS Lambda sondea el flujo 4 veces por segundo y, cuando detecta nuevos registros, invoca la función de Lambda pasando el evento de actualización como parámetro.

En un modelo basado en flujos, el mapeo de origen de eventos se mantiene en AWS Lambda. El mapeo de origen de eventos describe el mapeo entre los flujos y las funciones de Lambda. AWS Lambda proporciona una API ([CreateEventSourceMapping \(p. 380\)](#)) para crear el mapeo. También se puede utilizar la consola de AWS Lambda para crear mapeos de orígenes de eventos.

- **Invocación síncrona:** AWS Lambda invoca una función de Lambda mediante el tipo de invocación `RequestResponse` (invocación síncrona). Para obtener más información acerca de los tipos de invocación, consulte [Tipos de invocación \(p. 4\)](#).
- **Estructura del evento:** el evento que recibe la función de Lambda es la información de actualización de la tabla que AWS Lambda lee del flujo. Al configurar el mapeo de origen de eventos, el tamaño del lote que debe especificar es el número máximo de registros que puede recibir la función de Lambda por invocación.

Independientemente de qué invoque una función de Lambda AWS Lambda siempre ejecuta una función de Lambda en nombre del usuario. Si la función de Lambda necesita obtener acceso a recursos de AWS, debe conceder los permisos pertinentes para permitir dicho acceso. También debe conceder permisos de AWS Lambda para sondear el flujo de DynamoDB. Puede conceder todos estos permisos a un rol de IAM (rol de ejecución) que AWS Lambda pueda asumir para sondear el flujo y ejecutar la función de Lambda en su nombre. Primero debe crear este rol y, a continuación, activarlo en el momento de crear la función de Lambda. Para obtener más información, consulte [Administración de permisos mediante un rol de IAM \(rol de ejecución\) \(p. 193\)](#).

En el diagrama siguiente se ilustra el flujo de la aplicación:



1. La aplicación personalizada actualiza la tabla de DynamoDB.
2. Amazon DynamoDB publica las actualizaciones de elementos en el flujo.
3. AWS Lambda sondea el flujo y llama a la función de Lambda cuando detecta nuevos registros en él.

4. AWS Lambda ejecuta la función de Lambda asumiendo el rol de ejecución que se especificó en el momento de crear la función de Lambda.

Para ver un tutorial que muestra una configuración de ejemplo, consulte [Tutorial: Uso de AWS Lambda con Amazon DynamoDB \(p. 244\)](#).

Tutorial: Uso de AWS Lambda con Amazon DynamoDB

En este tutorial, debe crear una función de Lambda para consumir eventos de un flujo de DynamoDB.

El tutorial se divide en dos secciones principales:

- En primer lugar, realice las configuraciones necesarias para crear una función de Lambda y, a continuación, pruébela invocándola de forma manual utilizando los datos del evento de muestra.
- En segundo lugar, cree una tabla de DynamoDB habilitada para flujos y agregue un mapeo de origen de eventos en AWS Lambda para asociar el flujo a la función de Lambda. AWS Lambda comienza a sondear el flujo. A continuación, se prueba la configuración integral. A medida que se crean, actualizan y eliminan elementos de la tabla, Amazon DynamoDB escribe registros en el flujo. AWS Lambda detecta los nuevos registros al sondear el flujo y ejecuta la función de Lambda en nombre del usuario.

Important

Tanto la función de Lambda como el flujo de DynamoDB deben estar en la misma región de AWS.

En este tutorial se supone que crea estos recursos en la región `us-east-1`.

En este tutorial, utilice la AWS Command Line Interface para realizar operaciones de AWS Lambda, como crear una función de Lambda, crear un flujo o añadir registros al flujo. Utilice la consola de AWS Lambda para invocar la función manualmente antes de crear un flujo de DynamoDB. Verifique los valores de retorno y los logs en la interfaz de usuario de la consola.

Paso siguiente

[Paso 1: Preparación \(p. 244\)](#)

Paso 1: Preparación

Asegúrese de haber completado los pasos siguientes:

- Inscribirse en una cuenta de AWS y crear un usuario administrador en ella.
- Instalar y configurar la AWS CLI.

Para obtener instrucciones, consulte [Paso 1: Configuración de una cuenta de AWS y de la AWS CLI \(p. 199\)](#).

Paso siguiente

[Paso 2: Creación de una función de Lambda e invocación manual \(utilizando datos del evento de muestra\) \(p. 244\)](#)

Paso 2: Creación de una función de Lambda e invocación manual (utilizando datos del evento de muestra)

En esta sección, hará lo siguiente:

- Crear un paquete de implementación de la función de Lambda utilizando el código de muestra proporcionado. El código de muestra de la función de Lambda que utilizará para procesar los eventos de DynamoDB se proporciona en varios lenguajes. Seleccione uno de los lenguajes y siga las instrucciones correspondientes para crear un paquete de implementación.

Note

Para ver más ejemplos de uso de otros servicios de AWS dentro de la función, incluyendo llamadas a otras funciones de Lambda, consulte [AWS SDK para JavaScript](#)

- Crear un rol de IAM (rol de ejecución). En el momento de cargar el paquete de implementación, debe especificar un rol de IAM (rol de ejecución) que Lambda puede asumir para ejecutar la función en su nombre. Por ejemplo, AWS Lambda necesita permisos para las acciones de DynamoDB de forma que pueda sondear el flujo y leer los registros de este. En el modelo de extracción, también debe conceder a AWS Lambda permisos para invocar la función de Lambda. La función de Lambda de ejemplo escribe algunos de los datos de los eventos en CloudWatch, por lo que necesita permisos para realizar las acciones de CloudWatch necesarias.
- Cree la función de Lambda cargando el paquete de implementación y, a continuación, pruébela invocándola de forma manual utilizando datos del evento de muestra de DynamoDB. Proporcione tanto el paquete de implementación como el rol de IAM en el momento de crear una función de Lambda. También puede especificar otra información de configuración, como el nombre de la función, el tamaño de la memoria, el entorno en tiempo de ejecución que desea utilizar y el controlador. Para obtener más información sobre estos parámetros, consulte [CreateFunction \(p. 385\)](#). Después de crear la función de Lambda, puede invocarla mediante los datos del evento de muestra de Amazon DynamoDB.

Temas

- [Paso 2.1: Creación de un paquete de implementación de la función de Lambda \(p. 245\)](#)
- [Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 248\)](#)
- [Paso 2.3: Creación y prueba manual de la función de Lambda \(p. 249\)](#)

Paso 2.1: Creación de un paquete de implementación de la función de Lambda

En la lista Filter View, elija el lenguaje que desea utilizar para la función de Lambda. Aparece la sección correspondiente con código e instrucciones específicas para crear un paquete de implementación.

Node.js

1. Abrir un editor de texto y, a continuación, copie el siguiente código.

```
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

El código de muestra es compatible con los tiempos de ejecución v6.10 o v4.3 de Node.js.
Para obtener más información, consulte [Modelo de programación \(Node.js\) \(p. 9\)](#).

2. Guarde el archivo como `ProcessDynamoDBStream.js`.
3. Comprima el archivo `ProcessDynamoDBStream.js` como `ProcessDynamoDBStream.zip`.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 248\)](#)

Java

En el código siguiente, `handleRequest` es el controlador que AWS Lambda invoca y que proporciona datos de eventos. Este controlador utiliza la clase `DynamodbEvent` predefinida, que se define en la biblioteca `aws-lambda-java-events`.

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent;
import com.amazonaws.services.lambda.runtime.events.DynamodbEvent.DynamodbStreamRecord;

public class DDBEventProcessor implements
    RequestHandler<DynamodbEvent, String> {

    public String handleRequest(DynamodbEvent ddbEvent, Context context) {
        for (DynamodbStreamRecord record : ddbEvent.getRecords()){
            System.out.println(record.getEventID());
            System.out.println(record.geteventName());
            System.out.println(record.getDynamodb().toString());

        }
        return "Successfully processed " + ddbEvent.getRecords().size() + " records.";
    }
}
```

Si el controlador vuelve normalmente sin excepciones, Lambda considera que el lote de registros de entrada se ha procesado correctamente y empieza a leer nuevos registros en el flujo. Si el controlador genera una excepción, Lambda considera que el lote de registros de entrada no se ha procesado e invoca de nuevo la función con el mismo lote de registros.

Utilizando el código anterior (en un archivo denominado `DDBEventProcessor.java`), cree un paquete de implementación. Asegúrese de que añade las siguientes dependencias:

- `aws-lambda-java-core`
- `aws-lambda-java-events`

Para obtener más información, consulte [Modelo de programación para crear funciones de Lambda en Java \(p. 25\)](#).

El paquete de implementación puede ser un archivo `.zip` o `.jar` independiente. Puede utilizar cualquier herramienta de compilación y compresión con la que esté familiarizado para crear un paquete de implementación. Para ver ejemplos de cómo utilizar la herramienta de compilación Maven para crear un archivo `.jar` independiente, consulte [Creación de un paquete de implementación .jar utilizando Maven sin ningún IDE \(Java\) \(p. 70\)](#) y [Creación de un paquete de implementación .jar utilizando Maven y el IDE de Eclipse \(Java\) \(p. 72\)](#). Para ver un ejemplo de cómo utilizar la herramienta de compilación Gradle para crear un archivo `.zip`, consulte [Creación de un paquete de implementación .zip \(Java\) \(p. 75\)](#).

Después de verificar que se ha creado el paquete de implementación, vaya al paso siguiente para crear un rol de IAM (rol de ejecución). Debe especificar este rol en el momento de crear la función de Lambda.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 248\)](#)

C#

En el código siguiente, `ProcessDynamoEvent` es el controlador que AWS Lambda invoca y que proporciona datos de eventos. Este controlador utiliza la clase `DynamoDbEvent` predefinida, que se define en la biblioteca `Amazon.Lambda.DynamoDBEvents`.

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

Para crear un paquete de implementación, siga los pasos que se indican en [CLI de .NET Core \(p. 65\)](#). Al hacerlo, tenga en cuenta lo siguiente después de crear el proyecto de .NET:

- Cambie el nombre del archivo `Program.cs` predeterminado por un nombre de su elección, como, por ejemplo, `ProcessingDynamoDBStreams.cs`.
- Sustituya el contenido predeterminado del archivo `Program.cs` cuyo nombre ha cambiado por el ejemplo de código anterior.
- En el archivo `project.json`, añada las siguientes referencias al nodo `dependencies`.
 - `"Amazon.Lambda.Core": "1.0.0-*"`
 - `"Amazon.Lambda.Serialization.Json": "1.0.0-*"`
 - `"Amazon.Lambda.DynamoDBEvents": "1.0.0-*"`

Después de verificar que se ha creado el paquete de implementación, vaya al paso siguiente para crear un rol de IAM (rol de ejecución). Debe especificar este rol en el momento de crear la función de Lambda.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 248\)](#)

Python

1. Abrir un editor de texto y, a continuación, copie el siguiente código.

Note

La instrucción `from __future__ import print_function` permite escribir código compatible con Python versión 2 o 3. Si utiliza la versión 3.6 del tiempo de ejecución, no es necesario incluirla.

```
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'])))
```

2. Guarde el archivo como `ProcessDynamoDBStream.py`.
3. Comprima el archivo `ProcessDynamoDBStream.py` como `ProcessDynamoDBStream.zip`.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 248\)](#)

Paso 2.2: Creación del rol de ejecución (rol de IAM)

En esta sección, creará un rol de IAM con el tipo de rol y la política de acceso predefinidos que se indican a continuación:

- Rol de servicio de AWS del tipo AWS Lambda: este rol concede permisos a AWS Lambda para asumir el rol.
- AWSLambdaDynamoDBExecutionRole: es la política de permisos de acceso que se asocia al rol.

Para obtener más información acerca de los roles de IAM, consulte [IAM Roles](#) en la Guía del usuario de IAM. Utilice el siguiente procedimiento para crear el rol de IAM.

Para crear un rol de IAM (rol de ejecución)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.
2. Siga los pasos de [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM para crear un rol de IAM (rol de ejecución). Cuando siga los pasos para crear un rol, tenga en cuenta lo siguiente:
 - En Role Name, utilice un nombre que sea único dentro de la cuenta de AWS (por ejemplo, `lambda-dynamodb-execution-role`).
 - En Select Role Type, elija AWS Service Roles, seguido de AWS Lambda. Esto garantiza que los permisos del servicio de AWS Lambda asuman el rol.
 - En Attach Policy, elija AWSLambdaDynamoDBExecutionRole. Los permisos de esta política son suficientes para la función de Lambda de este tutorial.

3. Anote el ARN del rol. Lo necesitará en el siguiente paso al crear la función de Lambda.

Paso siguiente

[Paso 2.3: Creación y prueba manual de la función de Lambda \(p. 249\)](#)

Paso 2.3: Creación y prueba manual de la función de Lambda

En esta sección, hará lo siguiente:

- Crear una función de Lambda cargando el paquete de implementación.
- Probar la función de Lambda invocándola manualmente. En lugar de crear un origen de eventos, utilizará datos de un evento de muestra de DynamoDB.

En la siguiente sección, va a crear un flujo de DynamoDB y va a probar la experiencia integral.

[Paso 2.3.1: Creación de una función de Lambda \(carga del paquete de implementación\)](#)

En este paso, cargará el paquete de implementación utilizando la AWS CLI.

En el símbolo del sistema, ejecute el siguiente comando `create-function` de la CLI de Lambda utilizando el perfil adminuser.

Debe actualizar el comando proporcionando la ruta del archivo .zip y el ARN del rol de ejecución. El valor del parámetro `--runtime` puede ser `python3.6`, `python2.7`, `nodejs6.10`, `nodejs4.3` o `java8`, dependiendo del lenguaje que utilice para crear el código.

```
$ 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

Si elige Java 8 como tiempo de ejecución, el valor del controlador debe ser `packageName::methodName`.

Para obtener más información, consulte [CreateFunction \(p. 385\)](#). AWS Lambda crea la función y devuelve información de configuración de esta.

Si lo desea, puede cargar el archivo .zip en un bucket de Amazon S3 en la misma región de AWS y, a continuación, especificar el nombre del objeto y el bucket en el comando anterior. Debe reemplazar el parámetro `--zip-file` por el parámetro `--code`, como se muestra a continuación:

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

Paso 2.3.2: Prueba de la función de Lambda (invocación manual)

En este paso, invocará la función de Lambda manualmente mediante el comando `invoke` de la CLI de AWS Lambda y el siguiente evento de muestra de DynamoDB.

1. Copie el siguiente JSON en un archivo y guárdelo como `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"
                }
            },
            "SequenceNumber": "333",
            "SizeBytes": 0,
            "StreamViewType": "NEW_AND_OLD_IMAGES"
        },
        "eventSourceARN": "stream-ARN"
    }
]
```

```
"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"
}
]
```

- Ejecute el siguiente comando invoke.

```
$ aws lambda invoke \
--invocation-type RequestResponse \
--function-name ProcessDynamoDBStream \
--region us-east-1 \
--payload file://file-path/input.txt \
--profile adminuser \
outputfile.txt
```

Observe que el comando invoke especifica RequestResponse como tipo de invocación, lo que solicita la ejecución síncrona. Para obtener más información, consulte [Invoke \(p. 423\)](#). La función devuelve la cadena de mensaje (el mensaje de context.succeed() del código) en el cuerpo de la respuesta.

- Verifique la salida en el archivo outputfile.txt.

Puede monitorizar la actividad de la función de Lambda en la consola de AWS Lambda.

- La consola de AWS Lambda muestra una representación gráfica de algunas de las métricas de CloudWatch; en la sección Cloudwatch Metrics at a glance para la función. Inicie sesión en la Consola de administración de AWS en <https://console.aws.amazon.com/>.
- Para cada gráfico, también puede hacer clic en el enlace logs para ver directamente los logs de CloudWatch.

Paso siguiente

[Paso 3: Adición de un origen de eventos \(creación y asociación de un flujo de DynamoDB a la función de Lambda\) \(p. 251\)](#)

Paso 3: Adición de un origen de eventos (creación y asociación de un flujo de DynamoDB a la función de Lambda)

En esta sección, hará lo siguiente:

- Crear una tabla de Amazon DynamoDB con un flujo habilitado.

- Crear un mapeo de origen de eventos en AWS Lambda. Este mapeo de origen de eventos asocia el flujo de DynamoDB a la función de Lambda. Una vez creado este mapeo de origen de eventos, AWS Lambda comienza a sondear el flujo.
- Probar la experiencia integral. A medida que se actualiza la tabla, DynamoDB escribe los registros de eventos en el flujo. AWS Lambda sondea el flujo, detecta nuevos registros en él y ejecuta la función de Lambda en nombre del usuario pasando eventos a esta.

Paso 3.1: Creación de una tabla de DynamoDB con un flujo habilitado

Siga el procedimiento para crear una tabla con un flujo:

1. Inicie sesión en la Consola de administración de AWS y abra la consola de DynamoDB en <https://console.aws.amazon.com/dynamodb/>.
2. En la consola de DynamoDB, cree una tabla con los flujos habilitados. Para obtener más información acerca de cómo habilitar flujos, consulte [Captura de la actividad de las tablas con flujos de DynamoDB](#).

Important

Debe crear una tabla de DynamoDB en la misma región en la que ha creado la función de Lambda. En este tutorial se supone que se usa la región US East (N. Virginia). Además, tanto la tabla como las funciones de Lambda deben pertenecer a la misma cuenta de AWS.

3. Anote el ARN del flujo. Lo necesitará en el siguiente paso al asociar el flujo a la función de Lambda.

Paso 3.2: Adición de un origen de eventos en AWS Lambda

Ejecute el siguiente comando de AWS CLI `create-event-source-mapping`. Una vez ejecutado el comando, anote el UUID. Necesitará este UUID para hacer referencia al mapeo de origen de eventos en los comandos, por ejemplo, al eliminar el mapeo de origen de eventos.

```
$ 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

Esto crea un mapeo entre el flujo de DynamoDB especificado y la función de Lambda. Puede asociar un flujo de DynamoDB a varias funciones de Lambda, así como asociar la misma función de Lambda a varios flujos. Sin embargo, las funciones de Lambda compartirán el desempeño de lectura para el flujo que comparten.

Para obtener la lista de mapeos de orígenes de eventos, ejecute el siguiente comando.

```
$ aws lambda list-event-source-mappings \
--region us-east-1 \
--function-name ProcessDynamoDBStream \
--event-source DynamoDB-stream-arn \
--profile adminuser
```

La lista devuelve todos los mapeos de orígenes de eventos creados, y para cada uno de ellos muestra el `LastProcessingResult`, entre otras cosas. Este campo se utiliza para proporcionar un mensaje

informativo en caso de que surja algún problema. Los valores como `No records processed` (indica que AWS Lambda no ha comenzado el sondeo o que no hay registros en el flujo) y `OK` (indica que AWS Lambda ha leído correctamente los registros del flujo y ha invocado la función de Lambda) indican que no hay ningún problema. Si hay algún problema, recibirá un mensaje de error.

Paso 3.3: Prueba de la configuración

¡Ya ha terminado! Ahora, adminuser puede probar la configuración como se indica a continuación:

1. En la consola de DynamoDB, añada, actualice y elimine elementos de la tabla. DynamoDB escribe registros de estas acciones en el flujo.
2. AWS Lambda sondea el flujo y, cuando detecta actualizaciones en este, invoca la función de Lambda pasando los datos de eventos que encuentra en el flujo.
3. La función ejecuta y crea logs en Amazon CloudWatch. adminuser también puede verificar los logs mostrados en la consola de Amazon CloudWatch.

Paso 4: Implementación con AWS SAM y AWS CloudFormation

En la sección anterior, ha utilizado la API de AWS Lambda para crear y actualizar una función de Lambda proporcionando un paquete de implementación en un archivo ZIP. Sin embargo, puede que este mecanismo no sea conveniente para automatizar los pasos de implementación de las funciones, o para coordinar implementaciones y actualizaciones con otros elementos de una aplicación sin servidor, como los orígenes de eventos y los recursos posteriores.

Puede utilizar AWS CloudFormation para especificar, implementar y configurar fácilmente aplicaciones sin servidor. AWS CloudFormation es un servicio que le ayuda a modelar y configurar recursos de Amazon Web Services, por lo que podrá dedicar menos tiempo a la administración de dichos recursos y más tiempo a centrarse en las aplicaciones que se ejecutan en AWS. Puede crear una plantilla que describa todos los recursos de AWS que desea (como funciones de Lambda y tablas de DynamoDB), y AWS CloudFormation se encargará del aprovisionamiento y la configuración de dichos recursos.

Además, puede utilizar AWS Serverless Application Model para expresar los recursos que componen la aplicación sin servidor. Estos tipos de recursos, como las API y las funciones de Lambda, son totalmente compatibles con AWS CloudFormation, y facilitan la tarea de definir e implementar aplicaciones sin servidor.

Para obtener más información, consulte [Implementación de aplicaciones basadas en Lambda \(p. 156\)](#).

Especificación para la aplicación de DynamoDB

A continuación encontrará la plantilla SAM para esta aplicación. Copie el texto siguiente en un archivo `.yaml` y guárdelo junto al paquete ZIP que ha creado anteriormente. Tenga en cuenta que los valores de los parámetros `Handler` y `Runtime` deben coincidir con los que utilizó cuando creó la función en la sección anterior.

```
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
```

Implementación de la aplicación sin servidor

Para obtener información acerca de cómo empaquetar e implementar la aplicación sin servidor utilizando los comandos package y deploy, consulte [Empaquetado e implementación \(p. 162\)](#).

Uso de AWS Lambda con AWS CloudTrail

Puede activar CloudTrail en su cuenta de AWS para obtener logs de llamadas a la API y el historial de eventos relacionados de su cuenta. CloudTrail registra todos los eventos de acceso a la API como objetos en el bucket de Amazon S3 que especifique en el momento de activar CloudTrail.

Puede utilizar la característica de notificaciones de los buckets de Amazon S3 y hacer que Amazon S3 publique eventos de creación de objetos para AWS Lambda. Siempre que CloudTrail escriba logs en el bucket de S3, Amazon S3 puede invocar la función de Lambda pasando el evento de creación de objeto de Amazon S3 como parámetro. El evento de S3 proporciona información, incluido el nombre del bucket y el nombre de la clave del objeto de log creado por CloudTrail. El código de la función de Lambda puede leer el objeto de log y procesar los registros de acceso generados por CloudTrail. Por ejemplo, puede escribir el código de una función de Lambda para que le notifique si se ha realizado una llamada a una API específica en su cuenta.

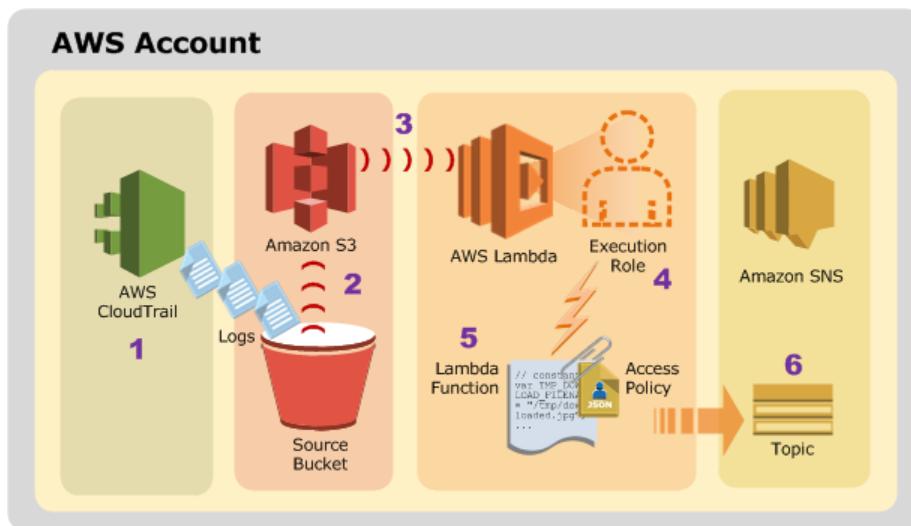
En este caso, activará CloudTrail para que pueda escribir logs de acceso en el bucket de S3. En cuanto a AWS Lambda, Amazon S3 es el origen de eventos, por lo que Amazon S3 publica los eventos para AWS Lambda e invoca la función de Lambda.

Note

Amazon S3 solo puede admitir un destino de eventos.

Para obtener información detallada acerca de cómo configurar Amazon S3 como origen de eventos, consulte [Uso de AWS Lambda con Amazon S3 \(p. 214\)](#).

En el siguiente diagrama, se resume el flujo:



1. AWS CloudTrail guarda los logs en un bucket de S3 (evento de creación de objeto).
2. Amazon S3 detecta el evento de creación de objeto.
3. Amazon S3 publica el evento `s3:ObjectCreated:*` en AWS Lambda invocando la función de Lambda, como se especifica en la configuración de notificaciones del bucket. Debido a que la política de permisos de acceso de la función de Lambda incluye permisos para que Amazon S3 invoque la función, Amazon S3 puede invocarla.
4. AWS Lambda ejecuta la función de Lambda asumiendo el rol de ejecución que se especificó en el momento de crear la función de Lambda.
5. La función de Lambda lee el evento de Amazon S3 que recibe como parámetro, determina dónde se encuentra el objeto de CloudTrail, lee el objeto de CloudTrail y, a continuación, procesa los registros de log en el objeto de CloudTrail.
6. Si el log incluye un registro con valores específicos de `eventType` y `eventSource`, publica el evento en el tema de Amazon SNS. En el [Tutorial: Uso de AWS Lambda con AWS CloudTrail \(p. 255\)](#), puede suscribirse al tema de SNS utilizando el protocolo de correo electrónico, lo que le permite obtener notificaciones por correo electrónico.

Para ver un tutorial que muestra un escenario de ejemplo, consulte [Tutorial: Uso de AWS Lambda con AWS CloudTrail \(p. 255\)](#).

Tutorial: Uso de AWS Lambda con AWS CloudTrail

Suponga que ha activado AWS CloudTrail para que su cuenta de AWS mantenga registros (logs) de las llamadas a la API de AWS que se realicen en su cuenta, y que desea recibir una notificación siempre que se realice una llamada a la API para crear un tema de SNS. A medida que se realizan llamadas a la API en su cuenta, CloudTrail escribe logs en el bucket de Amazon S3 que se haya configurado. En esta situación, desea que Amazon S3 publique los eventos de creación de objetos en AWS Lambda e invoque la función de Lambda a medida que CloudTrail cree objetos de log.

Cuando Amazon S3 invoca la función de Lambda, pasa un evento de S3; que identifica, entre otras cosas, el nombre del bucket y el nombre del clave del objeto creado por CloudTrail. La función de Lambda puede leer el objeto de log y conoce las llamadas a la API que se notificaron en el log.

Cada objeto que CloudTrail crea en el bucket de S3 es un objeto JSON, que contiene uno o varios registros de eventos. Cada registro, entre otras cosas, proporciona `eventSource` y `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",  
        ...  
    },  
    {  
        ...  
    },  
    ...  
}
```

Con fines ilustrativos, la función de Lambda le informará por correo electrónico si se registra en el log una llamada a la API para crear un tema de Amazon SNS. Es decir, cuando la función de Lambda analiza el log, busca los registros que contienen lo siguiente:

- `eventSource = "sns.amazonaws.com"`
- `eventName = "CreateTopic"`

Si los encuentra, publica el evento en el tema de Amazon SNS (configura este tema para que le envíe una notificación por correo electrónico).

Resumen de implementación

Al finalizar este tutorial, tendrá los siguientes recursos de Amazon S3, AWS Lambda, Amazon SNS y AWS Identity and Access Management (IAM) en su cuenta:

Note

En este tutorial se supone que crea estos recursos en la región `us-west-2`.

En Lambda:

- Una función de Lambda.
- Una política de acceso asociada a la función de Lambda. Esta política de permisos se utiliza para conceder permisos a Amazon S3 para invocar la función de Lambda. También restringirá los permisos para que Amazon S3 pueda invocar la función de Lambda únicamente para los eventos de creación de objetos procedentes de un bucket específico que pertenece a una cuenta de AWS concreta.

Note

Es posible que una cuenta de AWS elimine un bucket y que alguna otra cuenta de AWS cree posteriormente un bucket con el mismo nombre. Las condiciones adicionales garantizan que Amazon S3 solo pueda invocar la función de Lambda si detecta eventos de creación de objetos procedentes de un bucket específico que pertenece a una cuenta de AWS determinada.

Para obtener más información, consulte [Funcionamiento \(p. 188\)](#).

En IAM:

- Un rol de IAM (rol de ejecución). Debe conceder los permisos que necesita la función de Lambda a través del política de permisos asociada a este rol.

En Amazon S3:

- Un bucket. En este tutorial, el nombre del bucket será `examplebucket`. Al activar el registro de seguimiento en la consola de CloudTrail, debe especificar este bucket para que CloudTrail guarde los logs en él.
- Configuración de notificaciones en `examplebucket`. En la configuración, indicará a Amazon S3 que publique los eventos de creación de objetos en Lambda, invocando la función de Lambda. Para obtener más información acerca de la función de notificaciones de Amazon S3, consulte [Configuración de notificaciones de eventos de Amazon S3](#) en la Guía para desarrolladores de Amazon Simple Storage Service.
- Objeto de log de muestra de CloudTrail (`ExampleCloudTrailLog.json`) en el bucket `examplebucket`. En la primera mitad de este ejercicio, creará y probará la función de Lambda invocándola manualmente mediante un evento de muestra de S3. Este evento de muestra identifica `examplebucket` como nombre del bucket y este nombre de la clave del objeto de muestra. A continuación, la función de Lambda lee el objeto y le envía notificaciones por correo electrónico utilizando un tema de SNS.

En Amazon SNS:

- Un tema de SNS. Puede suscribirse a este tema especificando el correo electrónico como protocolo.

Ahora ya puede comenzar el tutorial.

Paso siguiente

[Paso 1: Preparación \(p. 257\)](#)

Paso 1: Preparación

En esta sección, hará lo siguiente:

- Inscribirse en una cuenta de AWS y configurar la AWS CLI.
- Activar CloudTrail en la cuenta.
- Crear un tema de SNS y suscribirse a él.

Siga los pasos de las secciones siguientes para realizar el proceso de configuración.

Note

En este tutorial, se presupone que se configuran los recursos en la región `us-west-2`.

Paso 1.1: Inscripción en AWS y configuración de la AWS CLI

Asegúrese de haber completado los pasos siguientes:

- Inscribirse en una cuenta de AWS y crear un usuario administrador en ella (denominado `adminuser`).
- Instalar y configurar la AWS CLI.

Para obtener instrucciones, consulte [Paso 1: Configuración de una cuenta de AWS y de la AWS CLI \(p. 199\)](#).

Paso 1.2: Activación de CloudTrail

En la consola de AWS CloudTrail, active el registro de seguimiento en la cuenta especificando `examplebucket` en la región `us-west-2` para que CloudTrail guarde los logs. Al configurar el registro de seguimiento, no active la notificación de SNS.

Para obtener instrucciones consulte [Introducción a la creación de registros de seguimiento](#) en la AWS CloudTrail User Guide.

Note

Aunque active CloudTrail ahora, no realizará ninguna configuración adicional para que la función de Lambda procese los logs reales de CloudTrail en la primera mitad de este ejercicio, sino que utilizará objetos de log de muestra de CloudTrail (que cargará) y eventos de muestra de S3 para invocar y probar manualmente la función de Lambda. En la segunda mitad de este tutorial, realizará pasos de configuración adicionales que permiten que la función de Lambda procese los logs de CloudTrail.

Paso 1.3: Creación de un tema de SNS y suscripción al tema

Siga el procedimiento para crear un tema de SNS en la región `us-west-2` y suscribirse a él proporcionando una dirección de correo electrónico como punto de enlace.

Para crear un tema y suscribirse a él

1. Crear un tema de SNS.

Para obtener instrucciones, consulte [Creación de un tema](#) en la Guía para desarrolladores de Amazon Simple Notification Service.

2. Suscríbase al tema proporcionando una dirección de correo electrónico como punto de enlace.

Para obtener instrucciones, consulte [Suscripción a un tema](#) en la Guía para desarrolladores de Amazon Simple Notification Service.

3. Anote el ARN del tema. Necesitará el valor en las secciones siguientes.

Paso siguiente

[Paso 2: Creación de una función de Lambda e invocación manual \(utilizando datos del evento de muestra\) \(p. 258\)](#)

Paso 2: Creación de una función de Lambda e invocación manual (utilizando datos del evento de muestra)

En esta sección, hará lo siguiente:

- Crear un paquete de implementación de la función de Lambda utilizando el código de muestra proporcionado. El código de muestra de la función de Lambda que utilizará para procesar los eventos de Amazon S3 se proporciona en varios lenguajes. Seleccione uno de los lenguajes y siga las instrucciones correspondientes para crear un paquete de implementación.

Note

La función de Lambda utiliza un evento de S3 que proporciona el nombre del bucket y el nombre de la clave del objeto creado por CloudTrail. A continuación, la función de Lambda lee ese objeto para procesar los registros de CloudTrail.

- Crear un rol de IAM (rol de ejecución). En el momento de cargar el paquete de implementación, debe especificar un rol de IAM (rol de ejecución) que Lambda puede asumir para ejecutar la función en su nombre.

- Cree la función de Lambda cargando el paquete de implementación y, a continuación, pruébelo invocándola de forma manual utilizando los datos del evento de muestra de CloudTrail.

Temas

- [Paso 2.1: Creación de un paquete de implementación](#) (p. 259)
- [Paso 2.2: Creación del rol de ejecución \(rol de IAM\)](#) (p. 260)
- [Paso 2.3: Creación y prueba manual de la función de Lambda](#) (p. 262)

Paso 2.1: Creación de un paquete de implementación

El paquete de implementación es un archivo .zip que contiene el código de la función de Lambda. En este tutorial, tendrá que instalar la biblioteca `async`. Para ello, abra una ventana de comandos y vaya al directorio donde desea guardar el archivo de código que copiará y guardará a continuación. Utilice npm para instalar la biblioteca `async` como se muestra a continuación:

```
npm install async
```

Node.js

1. Abrir un editor de texto y, a continuación, copie el siguiente código.

```
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) {
                console.error('Error parsing CloudTrail JSON');
                return next(err);
            }
            if (!records || !records.length) {
                console.log('No CloudTrail events found');
                return next();
            }
            var snsMessage = {
                TopicArn: SNS_TOPIC_ARN,
                Subject: 'CloudTrail Log',
                Message: json
            };
            sns.publish(snsMessage, next);
        }
    ],
    function(err, result) {
        if (err) {
            return callback(err);
        }
        else {
            return callback(null, result);
        }
    }
});
```

```
        } catch (err) {
            next('Unable to parse CloudTrail JSON: ' + err);
            return;
        }
    var matchingRecords = records
        .Records
        .filter(function(record) {
            return record.eventSource.match(EVENT_SOURCE_TO_TRACK)
                && record.eventName.match(EVENT_NAME_TO_TRACK);
        });

    console.log('Publishing ' + matchingRecords.length + ' notification(s) in
parallel...');
    async.each(
        matchingRecords,
        function(record, publishComplete) {
            console.log('Publishing notification: ', record);
            sns.publish({
                Message:
                    'Alert... SNS topic created: \n TopicARN=' +
                record.responseElements.topicArn + '\n\n' +
                    JSON.stringify(record),
                TopicArn: SNS_TOPIC_ARN
            }, publishComplete);
        },
        next
    );
},
], function (err) {
    if (err) {
        console.error('Failed to publish notifications: ', err);
    } else {
        console.log('Successfully published all notifications.');
    }
    callback(null,"message");
});
};

};
```

Note

El código de muestra es compatible con los tiempos de ejecución v6.10 o v4.3 de Node.js. Para obtener más información, consulte [Modelo de programación \(Node.js\) \(p. 9\)](#).

2. Guarde el archivo como `CloudTrailEventProcessing.js`.
3. Comprima el archivo `CloudTrailEventProcessing.js` como `CloudTrailEventProcessing.zip`.

Note

Estamos utilizando Node.js en este tutorial de ejemplo, pero también puede crear funciones de Lambda en Java o en Python.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 260\)](#)

Paso 2.2: Creación del rol de ejecución (rol de IAM)

Ahora debe crear un rol de IAM (rol de ejecución) que especificará al crear la función de Lambda. Este rol tiene una política de permisos que concede los permisos que necesita la función de Lambda, como permisos para escribir logs de CloudWatch, permisos para leer objetos de log de CloudTrail desde

un bucket de S3 y permisos para publicar eventos en un tema de SNS cuando la función de Lambda encuentra determinadas llamadas a la API en los logs de CloudTrail.

Para obtener más información acerca del rol de ejecución, consulte [Administración de permisos mediante un rol de IAM \(rol de ejecución\) \(p. 193\)](#).

Para crear un rol de IAM (rol de ejecución)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.
2. Cree una política administrada y asóciela al rol de IAM. En este paso, modificará una política administrada de AWS existente, la guardará con otro nombre y, a continuación, asociará la política de permisos a un rol de IAM que va a crear.
 - a. En el panel de navegación de la consola de IAM, elija Policies, seguido de Create policy.
 - b. Junto a Copy an AWS Managed Policy, elija Select.
 - c. Junto a AWSLambdaExecute, elija Select.
 - d. Copie la política siguiente en el campo Policy Document, sustituyendo la política existente y, a continuación, actualice la política con el ARN del tema de Amazon SNS que creó.

```
{  
    "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. Anote el nombre de la política de permisos porque lo utilizará en el siguiente paso.
4. Siga los pasos de [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM para crear un rol de IAM y, a continuación, asocie al rol la política de permisos que acaba de crear. Cuando siga los pasos para crear un rol, tenga en cuenta lo siguiente:
 - En Role Name, utilice un nombre que sea único dentro de la cuenta de AWS (por ejemplo, lambda-cloudtrail-execution-role).
 - En Select Role Type, elija AWS Service Roles, seguido de AWS Lambda.
 - En Attach Policy, seleccione la política que ha creado en el paso anterior.

Paso siguiente

Paso 2.3: Creación y prueba manual de la función de Lambda (p. 262)

Paso 2.3: Creación y prueba manual de la función de Lambda

En esta sección, hará lo siguiente:

- Crear una función de Lambda cargando el paquete de implementación.
- Probar la función de Lambda invocándola manualmente.

En este paso, se utiliza un evento de muestra de S3 que identifica el nombre del bucket y el objeto de muestra (es decir, un log de CloudTrail de ejemplo). En la siguiente sección, configurará la notificación en el bucket de S3 para publicar eventos de creación de objetos y probar la experiencia integral.

Paso 2.3.1: Creación de la función de Lambda (carga del paquete de implementación)

En este paso, cargará el paquete de implementación utilizando la AWS CLI y proporcionará información de configuración al crear la función de Lambda. En el símbolo del sistema, ejecute el siguiente comando `create-function` de la CLI de Lambda utilizando `adminuser` para `profile`.

Note

Debe actualizar el comando proporcionando la ruta del archivo .zip (`//file-path/CloudTrailEventProcessing.zip`) y el ARN del rol de ejecución (`execution-role-arn`). Si ha utilizado el código de ejemplo proporcionado anteriormente en este tutorial, establezca el valor del parámetro `--runtime` en `nodejs6.10` o en `nodejs4.3`.

También puede crear funciones de Lambda en Java o en Python. Si utiliza otro lenguaje, cambie el valor del parámetro `--runtime` a `java8`, `python3.6` o `python2.7`, según sea necesario.

```
$ aws lambda create-function \
--region us-west-2 \
--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
```

Si lo desea, puede cargar el archivo .zip en un bucket de Amazon S3 en la misma región de AWS y, a continuación, especificar el nombre del objeto y el bucket en el comando anterior. Debe reemplazar el parámetro `--zip-file` por el parámetro `--code`, como se muestra a continuación:

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

Note

Puede crear una función de Lambda utilizando la consola de AWS Lambda, en cuyo caso debe tomar nota del valor de los parámetros del comando `create-function` de la AWS CLI. Debe proporcionar los mismos valores en la consola.

Paso 2.3.2: Prueba de la función de Lambda (invocación manual)

En esta sección invocará la función de Lambda de forma manual con los datos de un evento de muestra de Amazon S3. Cuando se ejecuta la función de Lambda, lee el objeto de S3 (un log de muestra de CloudTrail) desde el bucket identificado en los datos del evento de S3 y, a continuación, publica un evento en el tema de SNS si los informes del log de muestra de CloudTrail utilizan una API específica. Para este tutorial, la API es la API de SNS que se utilizó para crear un tema. Es decir, el log de CloudTrail informa

sobre un registro que identifica sns.amazonaws.com como eventSource y CreateTopic como eventName.

1. Guarde el siguiente log de muestra de CloudTrail en un archivo (`ExampleCloudTrailLog.json`).

Note

Observe que uno de los eventos de este log tiene sns.amazonaws.com como eventSource y CreateTopic como eventName. La función de Lambda lee los logs y, si encuentra un evento de este tipo, lo publica en el tema de Amazon SNS que se ha creado y, a continuación, el usuario recibirá un mensaje de correo electrónico cuando invoque la función de Lambda manualmente.

```
{  
    "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"
    }
]
```

- Ejecute el comando `gzip` para crear un archivo `.gz` a partir del archivo de origen anterior.

```
$ gzip ExampleCloudTrailLog.json
```

Esto crea el archivo `ExampleCloudTrailLog.json.gz`.

- Cargue el archivo `ExampleCloudTrailLog.json.gz` en el `examplebucket` que especificó en la configuración de CloudTrail.

Este objeto se especifica en los datos del evento de muestra de Amazon S3 que utilizamos para invocar manualmente la función de Lambda.

- Guarde el siguiente JSON (un ejemplo de evento de S3) en un archivo, `input.txt`. Anote los valores del nombre del bucket y del nombre de la clave del objeto.

Debe proporcionar esta evento de muestra cuando invoque la función de Lambda. Para obtener más información sobre la estructura del evento de S3, consulte [Estructura de mensaje de evento](#) en la Guía para desarrolladores de Amazon Simple Storage Service.

```
{
    "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. En la Consola de administración de AWS, invoque la función manualmente utilizando los datos del evento de muestra de Amazon S3. Para obtener instrucciones, consulte el ejercicio [Paso 2.2: Invocación manual de la función de Lambda y verificación de los resultados, los logs y las métricas \(p. 206\)](#) de la Introducción. En la consola, utilice los siguientes datos de un evento de muestra de Amazon S3.

Note

Recomendamos que invoque la función utilizando la consola, ya que la interfaz de usuario de la consola proporciona una interfaz fácil de utilizar para examinar los resultados de la ejecución, incluido el resumen de ejecución, los logs escritos por el código y los resultados devueltos por la función (dado que la consola siempre realiza una ejecución síncrona, invoca la función de Lambda utilizando el tipo de invocación `RequestResponse`).

```
{  
    "Records": [  
        {  
            "eventVersion": "2.0",  
            "eventSource": "aws:s3",  
            "awsRegion": "us-west-2",  
            "eventTime": "1970-01-01T00:00:00.000Z",  
            "eventName": "ObjectCreated:Put",  
            "userIdentity": {  
                "principalId": "AIDAJDPLRKLG7UEEXAMPLE"  
            },  
            "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"  
                }  
            }  
        }  
    ]
```

```
}
```

6. Ejecute el siguiente comando de la AWS CLI para invocar la función manualmente utilizando `adminuser` para `profile`.

```
$ aws lambda invoke-async \
--function-name CloudTrailEventProcessing \
--region us-west-2 \
--invoke-args /filepath/input.txt \
--debug \
--profile adminuser
```

Debido a que el objeto de log de ejemplo tiene un registro de evento que muestra la API de SNS a la que se debe llamar para crear un tema, la función de Lambda publica ese evento en el tema de SNS, y el usuario debería recibir una notificación por correo electrónico.

Puede monitorizar la actividad de la función de Lambda mediante el uso de las métricas y los logs de CloudWatch. Para obtener más información sobre la monitorización de CloudWatch, consulte [Solución de problemas y monitorización de funciones de AWS Lambda con Amazon CloudWatch \(p. 120\)](#).

7. (Opcional) Invoque la función de Lambda manualmente mediante la AWS CLI, como se indica a continuación:
 - a. Guarde el JSON del paso 2 anterior de este procedimiento en un archivo denominado `input.txt`.
 - b. Ejecute el siguiente comando `invoke`:

```
$ aws lambda invoke \
--invocation-type Event \
--function-name CloudTrailEventProcessing \
--region us-west-2 \
--payload file:///filepath/input.txt \
--profile adminuser
outputfile.txt
```

Note

En este tutorial de ejemplo, el mensaje se guarda en el archivo `outputfile.txt`. Si solicita una ejecución síncrona (con el tipo de invocación `RequestResponse`), la función devuelve la cadena de mensaje en el cuerpo de la respuesta.

Para Node.js, podría ser uno de los siguientes (lo que se haya especificado en el código):

```
context.succeed("message")
context.fail("message")
context.done(null, "message")
```

Para Python o Java, es el mensaje de la instrucción `return`:

```
return "message"
```

Paso siguiente

[Paso 3: Adición de un origen de eventos \(configuración de Amazon S3 para que publique eventos\) \(p. 267\)](#)

Paso 3: Adición de un origen de eventos (configuración de Amazon S3 para que publique eventos)

En esta sección, añadirá la configuración restante, de manera que Amazon S3 pueda publicar eventos de creación de objetos en AWS Lambda e invocar la función de Lambda. Hará lo siguiente:

- Añadir permisos a la política de acceso de la función de Lambda para permitir que Amazon S3 invoque la función.
- Añadir la configuración de notificaciones al bucket de origen. En la configuración de notificaciones, debe proporcionar lo siguiente:
 - Tipo de evento para el que desea que Amazon S3 publique eventos. Para este tutorial, debe especificar el tipo de evento `s3:ObjectCreated:*`.
 - Función de Lambda que se debe invocar.

Paso 3.1: Adición de permisos a la política de acceso de la función de Lambda

1. Ejecute el siguiente comando `add-permission` de la CLI de Lambda para conceder a la entidad principal del servicio de Amazon S3 (`s3.amazonaws.com`) permisos para realizar la acción `lambda:InvokeFunction`. Tenga en cuenta únicamente que se concede permiso a Amazon S3 para invocar la función si se cumplen las siguientes condiciones:
 - Se detecta un evento de creación de objeto en un bucket específico.
 - El bucket pertenece a una cuenta determinada de AWS. Si el propietario de un bucket lo elimina, otra cuenta de AWS puede crear un bucket con el mismo nombre. Esta condición garantiza que solo una cuenta determinada de AWS pueda invocar la función de Lambda.

```
$ aws lambda add-permission \
--function-name CloudTrailEventProcessing \
--region us-west-2 \
--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. Verifique la política de acceso de la función ejecutando el comando `get-policy` de la AWS CLI.

```
$ aws lambda get-policy \
--function-name function-name \
--profile adminuser
```

Pasos 3.2: Configuración de las notificaciones en el bucket

Añada la configuración de notificaciones al bucket `examplebucket` para solicitar a Amazon S3 que publique los eventos de creación de objetos en Lambda. En la configuración, especifique lo siguiente:

- Tipo de evento: para este tutorial, puede tratarse de cualquier tipo de evento que cree objetos.
- ARN de la función de Lambda: es la función de Lambda que desea que Amazon S3 invoque. El ARN tiene el formato siguiente:

`arn:aws:lambda:aws-region:account-id:function:function-name`

Por ejemplo, la función CloudTrailEventProcessing creada en la región us-west-2 tiene en siguiente ARN:

`arn:aws:lambda:us-west-2:account-id:function:CloudTrailEventProcessing`

Para obtener instrucciones acerca de cómo añadir una configuración de notificaciones a un bucket, consulte [Habilitación de notificaciones de eventos](#) en la Guía del usuario de la consola de Amazon Simple Storage Service.

Paso 3.3: Prueba de la configuración

¡Ya ha terminado! Ahora puede probar la configuración como se indica a continuación:

1. Realice alguna acción en su cuenta de AWS. Por ejemplo, añada otro tema en la consola de Amazon SNS.
2. Recibirá una notificación por correo electrónico acerca de este evento.
3. AWS CloudTrail crea un log de objeto en el bucket.
4. Si abre el objeto de log (el archivo .gz), el log muestra el evento CreateTopic de SNS.
5. Para cada objeto que crea AWS CloudTrail, Amazon S3 invoca la función de Lambda pasando el objeto de log como datos del evento.
6. Lambda ejecuta la función. La función analiza el log, busca un evento CreateTopic de SNS y, a continuación, el usuario recibe una notificación por correo electrónico.

Puede monitorizar la actividad de la función de Lambda mediante el uso de las métricas y los logs de CloudWatch. Para obtener más información sobre la monitorización de CloudWatch, consulte [Solución de problemas y monitorización de funciones de AWS Lambda con Amazon CloudWatch \(p. 120\)](#).

The screenshot shows the AWS CloudWatch Logs interface. On the left, there's a navigation sidebar with links like Dashboard, Alarms, Logs (which is selected and highlighted in orange), Metrics, DynamoDB, EBS, EC2, and ELB. The main area has a title bar that says "Log Groups > Streams for /aws/lambda/CloudTrailEvent...". Below the title bar are two buttons: "Create Log Stream" and "Delete Log Stream". The main content area is titled "Log Streams" and contains a table with the following data:

| Log Streams | Last Ingestion Time |
|----------------------------------|------------------------|
| 00478cb7889e4986b8eaccc248ae5b17 | 2014-12-18 06:49 UTC-8 |
| 03faa169843045c4b0228b2cca9688ab | 2014-12-17 21:04 UTC-8 |
| 0a769e0aeb314968a5a97f3bea96c932 | 2014-12-18 12:34 UTC-8 |
| 163d530f1d7e431abd2eb0cf26276dd3 | 2014-12-18 17:39 UTC-8 |
| 17f694642e704654bf054ddca9334383 | 2014-12-17 19:34 UTC-8 |
| 1b667a95eae84b7da4357009ab544040 | 2014-12-18 01:34 UTC-8 |
| 1b84b98a43ea42f78d59c735a8277905 | 2014-12-18 18:34 UTC-8 |

Paso 4: Implementación con AWS SAM y AWS CloudFormation

En la sección anterior, ha utilizado la API de AWS Lambda para crear y actualizar una función de Lambda proporcionando un paquete de implementación en un archivo ZIP. Sin embargo, puede que este mecanismo no sea conveniente para automatizar los pasos de implementación de las funciones, o para coordinar implementaciones y actualizaciones con otros elementos de una aplicación sin servidor, como los orígenes de eventos y los recursos posteriores.

Puede utilizar AWS CloudFormation para especificar, implementar y configurar fácilmente aplicaciones sin servidor. AWS CloudFormation es un servicio que le ayuda a modelar y configurar recursos de Amazon

Web Services, por lo que podrá dedicar menos tiempo a la administración de dichos recursos y más tiempo a centrarse en las aplicaciones que se ejecutan en AWS. Puede crear una plantilla que describa todos los recursos de AWS que desea (como funciones de Lambda y tablas de DynamoDB), y AWS CloudFormation se encargará del aprovisionamiento y la configuración de dichos recursos.

Además, puede utilizar AWS Serverless Application Model para expresar los recursos que componen la aplicación sin servidor. Estos tipos de recursos, como las API y las funciones de Lambda, son totalmente compatibles con AWS CloudFormation, y facilitan la tarea de definir e implementar aplicaciones sin servidor.

Para obtener más información, consulte [Implementación de aplicaciones basadas en Lambda \(p. 156\)](#).

Especificaciones para la aplicación de Amazon API Gateway

A continuación encontrará la plantilla SAM para esta aplicación. Copie el texto siguiente en un archivo .yaml y guárdelo junto al paquete ZIP que ha creado en la sección anterior. Tenga en cuenta que los valores de los parámetros Handler y Runtime deben coincidir con los que utilizó cuando creó la función en la sección anterior.

```
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
```

Implementación de la aplicación sin servidor

Para obtener información acerca de cómo empaquetar e implementar la aplicación sin servidor utilizando los comandos package y deploy, consulte [Empaquetado e implementación \(p. 162\)](#).

Uso de AWS Lambda con Amazon SNS desde cuentas distintas

Para poder realizar entregas de Amazon SNS a Lambda entre distintas cuentas, debe autorizar que se pueda invocar la función de Lambda desde Amazon SNS. A su vez, Amazon SNS debe permitir que la cuenta de Lambda se suscriba al tema de Amazon SNS. Por ejemplo, si el tema de Amazon SNS está en la cuenta A y la función de Lambda está en la cuenta B, cada cuenta debe conceder permisos a la otra para que acceda a sus recursos. Dado que en la consola de AWS no están disponibles todas las opciones para configurar permisos entre cuentas, se utiliza la CLI de AWS para configurar todo el proceso.

Para ver un tutorial que muestra una configuración de ejemplo, consulte [Tutorial: Uso de AWS Lambda con Amazon SNS \(p. 270\)](#).

Tutorial: Uso de AWS Lambda con Amazon SNS

En este tutorial, se crea una función de Lambda en una cuenta de AWS para suscribirse a un tema de Amazon SNS de otra cuenta de AWS.

El tutorial se divide en tres secciones principales:

- En primer lugar, se realiza la configuración necesaria para crear una función de Lambda.
- En segundo lugar, se crea un tema de Amazon SNS en otra cuenta de AWS.
- En tercer lugar, se conceden permisos desde cada cuenta para que la función de Lambda se suscriba al tema de Amazon SNS. A continuación, se prueba la configuración integral.

Important

En este tutorial se supone que crea estos recursos en la región `us-east-1`.

En este tutorial, se utiliza la AWS Command Line Interface para realizar operaciones de AWS Lambda, como la creación de una función de Lambda, la creación de un tema de Amazon SNS y la concesión de permisos permitir que estos dos recursos puedan acceder el uno al otro.

Paso siguiente

[Paso 1: Preparación \(p. 270\)](#)

Paso 1: Preparación

- Inscríbase en una cuenta de AWS y cree un usuario administrador en ella (denominado adminuser).
- Instale y configure la AWS CLI.

Para obtener instrucciones, consulte [Paso 1: Configuración de una cuenta de AWS y de la AWS CLI \(p. 199\)](#).

Paso siguiente

[Paso 2: Creación de una función de Lambda \(p. 271\)](#)

Paso 2: Creación de una función de Lambda

En esta sección, hará lo siguiente:

- Crear un paquete de implementación de la función de Lambda utilizando el código de muestra proporcionado. El código de muestra de la función de Lambda que utilizará para suscribirse a un tema de Amazon SNS se proporciona en varios lenguajes. Seleccione uno de los lenguajes y siga las instrucciones correspondientes para crear un paquete de implementación.
- Crear un rol de IAM (rol de ejecución). En el momento de cargar el paquete de implementación, debe especificar un rol de IAM (rol de ejecución) que Lambda puede asumir para ejecutar la función en su nombre.

Temas

- [Paso 2.1: Creación de un paquete de implementación de la función de Lambda \(p. 271\)](#)
- [Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 273\)](#)

Paso 2.1: Creación de un paquete de implementación de la función de Lambda

En la lista Filter View, elija el lenguaje que desea utilizar para la función de Lambda. Aparece la sección correspondiente con código e instrucciones específicas para crear un paquete de implementación.

Node.js

1. Abrir un editor de texto y, a continuación, copie el siguiente código.

```
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

El código de muestra es compatible con los tiempos de ejecución v6.10 o v4.3 de Node.js.
Para obtener más información, consulte [Modelo de programación \(Node.js\) \(p. 9\)](#).

2. Guarde el archivo como `index.js`.
3. Comprima el archivo `index.js` como `LambdaWithSNS.zip`.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 273\)](#)

Java

Abrir un editor de texto y, a continuación, copie el siguiente código.

```
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;
    }
}
```

Utilizando el código anterior (en un archivo denominado `LambdaWithSNS.java`), cree un paquete de implementación. Asegúrese de que añade las siguientes dependencias:

- `aws-lambda-java-core`
- `aws-lambda-java-events`

Para obtener más información, consulte [Modelo de programación para crear funciones de Lambda en Java \(p. 25\)](#).

El paquete de implementación puede ser un archivo `.zip` o `.jar` independiente. Puede utilizar cualquier herramienta de compilación y compresión con la que esté familiarizado para crear un paquete de implementación. Para ver ejemplos de cómo utilizar la herramienta de compilación Maven para crear un archivo `.jar` independiente, consulte [Creación de un paquete de implementación .jar utilizando Maven sin ningún IDE \(Java\) \(p. 70\)](#) y [Creación de un paquete de implementación .jar utilizando Maven y el IDE de Eclipse \(Java\) \(p. 72\)](#). Para ver un ejemplo de cómo utilizar la herramienta de compilación Gradle para crear un archivo `.zip`, consulte [Creación de un paquete de implementación .zip \(Java\) \(p. 75\)](#).

Después de verificar que se ha creado el paquete de implementación, vaya al paso siguiente para crear un rol de IAM (rol de ejecución). Debe especificar este rol en el momento de crear la función de Lambda.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 273\)](#)

Python

1. Abrir un editor de texto y, a continuación, copie el siguiente código.

Note

La instrucción `from __future__ import print_function` permite escribir código compatible con Python versión 2 o 3. Si utiliza la versión 3.6 del tiempo de ejecución, no es necesario incluirla.

```
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. Guarde el archivo como `lambda_handler.py`.
3. Comprima el archivo `lambda_handler.py` como `LambdaWithSNS.zip`.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 273\)](#)

Paso 2.2: Creación del rol de ejecución (rol de IAM)

En esta sección, creará un rol de IAM con el tipo de rol y la política de acceso predefinidos que se indican a continuación:

- Rol de servicio de AWS del tipo AWS Lambda: este rol concede permisos a AWS Lambda para asumir el rol.
- AWSLambdaBasicExecutionRole: esta es la política de permisos de acceso que se asocia al rol.

Para obtener más información acerca de los roles de IAM, consulte [IAM Roles](#) en la Guía del usuario de IAM. Utilice el siguiente procedimiento para crear el rol de IAM.

Para crear un rol de IAM (rol de ejecución)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.
2. Siga los pasos de [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM para crear un rol de IAM (rol de ejecución). Cuando siga los pasos para crear un rol, tenga en cuenta lo siguiente:
 - En Role Name, utilice un nombre que sea único dentro de la cuenta de AWS (por ejemplo, lambda-sns-execution-role).
 - En Select Role Type, elija AWS Service Roles, seguido de AWS Lambda. Esto garantiza que los permisos del servicio de AWS Lambda asuman el rol.
 - En Attach Policy, elija AWSLambdaBasicExecutionRole. Los permisos de esta política son suficientes para la función de Lambda de este tutorial.
3. Anote el ARN del rol. Lo necesitará en el siguiente paso al crear la función de Lambda.

Paso 3: Configuración de los permisos entre las cuentas

En esta sección, utilizará comandos de la CLI para establecer los permisos entre la cuenta de la función de Lambda y la cuenta del tema de Amazon SNS cuenta y, a continuación, probará la suscripción.

1. Desde la cuenta A, cree el tema de Amazon SNS:

```
aws sns create-topic \
--name lambda-x-account
```

Anote el ARN del tema que devuelve el comando. Lo necesitará cuando añada permisos a la función de Lambda para suscribirse al tema.

2. Desde la cuenta B, cree la función de Lambda. Para el parámetro de tiempo de ejecución, seleccione `nodejs6.10`, `nodejs4.3`, `python3.6`, `python2.7` o `java8`, en función del código de muestra que haya seleccionado al crear el paquete de implementación.

```
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
```

Anote el ARN de la función que devuelve el comando. Lo necesitará cuando añada permisos para permitir a Amazon SNS que invoque la función.

3. Desde la cuenta A, añada permiso para que la cuenta B se suscriba al tema:

```
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. Desde la cuenta B añada el permiso de Lambda para permitir la invocación desde Amazon SNS:

```
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
```

Como respuesta, Lambda devuelve el siguiente código JSON. El valor de Statement es una versión de cadena JSON de la instrucción que se añade a la política de la función de Lambda:

```
{
  "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

No utilice el parámetro --source-account para añadir una cuenta de origen a la política de Lambda al añadir la política. La cuenta de origen no es compatible con los orígenes de eventos de Amazon SNS y dará lugar a la denegación de acceso. Esto no tiene ningún impacto en la seguridad, ya que la cuenta de origen se incluye en el ARN del origen.

5. Desde la cuenta B, suscriba la función de Lambda al tema:

```
aws sns subscribe \
--topic-arn Amazon SNS topic arn \
--protocol lambda \
--notification-endpoint arn:aws:lambda:us-east-1:B:function:SNS-X-Account
```

Debería ver una salida JSON similar a esta:

```
{
```

```
    "SubscriptionArn": "arn:aws:sns:us-east-1:A:lambda-x-account:5d906xxxx-7c8x-45dx-a9dx-0484e31c98xx"  
}
```

6. Ahora puede probar la suscripción desde cuenta A. Escriba "Hello World" en un archivo de texto y guárdelo como message.txt. A continuación, ejecute el siguiente comando:

```
aws sns publish \  
  --topic-arn arn:aws:sns:us-east-1:A:lambda-x-account \  
  --message file://message.txt \  
  --subject Test
```

Esto devolverá un ID de mensaje con un identificador único, indicando que el mensaje ha sido aceptado por el servicio de Amazon SNS. Posteriormente, Amazon SNS intentará entregárselo a los suscriptores del tema.

Note

También puede proporcionar una cadena JSON directamente al parámetro `message`, pero si utiliza un archivo de texto, puede incluir saltos de línea en el mensaje.

Para obtener más información sobre Amazon SNS, consulte [¿Qué es Amazon Simple Notification Service?](#)

Uso de AWS Lambda con Amazon API Gateway (bajo demanda a través de HTTPS)

Puede invocar funciones de AWS Lambda a través de HTTPS. Para ello, puede definir un punto de enlace y una API de REST personalizados utilizando [Amazon API Gateway](#) y, a continuación, mapear métodos individuales, como `GET` y `PUT`, a funciones de Lambda específicas. También puede añadir un método especial denominado ANY para mapear todos los métodos admitidos (`GET`, `POST`, `PATCH` y `DELETE`) a la función de Lambda. Cuando se envía una solicitud HTTPS al punto de enlace de la API, el servicio de Amazon API Gateway invoca la función de Lambda correspondiente. Para obtener más información acerca del método ANY, consulte [Paso 3: Creación de un microservicio sencillo utilizando Lambda y API Gateway \(p. 211\)](#).

Amazon API Gateway también añade una capa entre los usuarios de la aplicación y la lógica de la aplicación que permite hacer lo siguiente:

- Aplicar una limitación a determinados usuarios o solicitudes.
- Protegerse contra ataques de denegación de servicio distribuido.
- Proporcionar una capa de caché para almacenar en caché la respuesta de la función de Lambda.

Tenga en cuenta lo siguiente sobre cómo funciona la integración entre Amazon API Gateway y AWS Lambda:

- **Modelo push-event:** es un modelo (consulte [Mapeo de orígenes de eventos \(p. 134\)](#)) en el que Amazon API Gateway invoca la función de Lambda pasando los datos en el cuerpo de la solicitud como parámetro de la función de Lambda.
- **Invocación síncrona:** Amazon API Gateway puede invocar la función de Lambda y obtener una respuesta en tiempo real especificando `RequestResponse` como tipo de invocación. Para obtener información acerca de los tipos de invocación, consulte [Tipos de invocación \(p. 4\)](#).

- Estructura del evento: el evento que recibe la función de Lambda es el cuerpo de la solicitud HTTPS que recibe Amazon API Gateway, y la función de Lambda es el código personalizado escrito para procesar ese tipo de evento específico.

Tenga en cuenta que existen dos tipos de políticas de permisos con las que se trabaja al configurar la experiencia integral:

- Permisos para la función de Lambda: independientemente de qué invoque una función de Lambda, AWS Lambda ejecuta la función asumiendo el rol de IAM (rol de ejecución) que se especifique en el momento de crear la función de Lambda. La política de permisos asociada a este rol se utiliza para conceder a la función de Lambda los permisos que necesita. Por ejemplo, si la función de Lambda necesita leer un objeto, debe conceder permisos para las acciones de Amazon S3 correspondientes en la política de permisos. Para obtener más información, consulte [Administración de permisos mediante un rol de IAM \(rol de ejecución\) \(p. 193\)](#).
- Permiso para que Amazon API Gateway invoque la función de Lambda: Amazon API Gateway no puede invocar la función de Lambda sin su permiso. Puede conceder este permiso a través de la política de permisos asociada a la función de Lambda.

Para ver un tutorial que muestra una configuración de ejemplo, consulte [Uso de AWS Lambda con Amazon API Gateway \(bajo demanda a través de HTTPS\) \(p. 276\)](#).

Uso de AWS Lambda con Amazon API Gateway (bajo demanda a través de HTTPS)

En este ejemplo se crea una API sencilla (`DynamoDBOperations`) utilizando Amazon API Gateway. Una instancia de Amazon API Gateway es un conjunto de recursos y métodos. En este tutorial, se crea un recurso (`DynamoDBManager`) y se define un método (`POST`) en él. El método está respaldado por una función de Lambda (`LambdaFunctionForAPIGateway`). Es decir, cuando se invoca el método a través de un punto de enlace HTTPS, Amazon API Gateway invoca la función de Lambda.

El método `POST` del recurso `DynamoDBManager` admite las siguientes operaciones de DynamoDB:

- Crear, actualizar y eliminar un elemento.
- Leer un elemento.
- Examinar un elemento.
- Otras operaciones (echo, ping), no relacionadas con DynamoDB, que puede utilizar para las pruebas.

La carga de solicitud que se envía en la solicitud `POST` identifica la operación de DynamoDB y proporciona los datos necesarios. Por ejemplo:

- A continuación se muestra una carga de solicitud de ejemplo para una operación put item de DynamoDB:

```
{  
    "operation": "create",  
    "tableName": "LambdaTable",  
    "payload": {  
        "Item": {  
            "Id": "1",  
            "name": "Bob"  
        }  
    }  
}
```

- A continuación se muestra una carga de solicitud de ejemplo para una operación read item de DynamoDB:

```
{  
    "operation": "read",  
    "tableName": "LambdaTable",  
    "payload": {  
        "Key": {  
            "Id": "1"  
        }  
    }  
}
```

- A continuación se muestra una carga de solicitud de ejemplo para la operación echo. A continuación, se debe enviar una solicitud PUT de HTTPS al punto de enlace, utilizando los siguientes datos en el cuerpo de la solicitud.

```
{  
    "operation": "echo",  
    "payload": {  
        "somekey1": "somevalue1",  
        "somekey2": "somevalue2"  
    }  
}
```

También puede crear y administrar puntos de enlace de la API desde la consola de AWS Lambda. Por ejemplo, busque el microservicio (microservice) en los proyectos. Este tutorial no utiliza la consola, sino que utiliza la AWS CLI para mostrarle información más detallada sobre el funcionamiento de la API.

Note

API Gateway ofrece capacidades avanzadas, como por ejemplo:

- Transferencia de la solicitud completa: una función de Lambda puede recibir la solicitud HTTP completa (en lugar de solo el cuerpo de la solicitud) y establecer la respuesta HTTP (en lugar de solo el cuerpo de la respuesta) utilizando el tipo de integración `AWS_PROXY`.
- Métodos catch-all: mapea todos los métodos de un recurso de la API a una sola función con un único mapeo, utilizando el método catch-all `ANY`.
- Recursos catch-all: mapea todas las subrutas de un recurso a una función de Lambda sin necesidad de realizar ninguna configuración adicional, mediante el nuevo parámetro de ruta `({proxy+})`.

Para obtener más información acerca de estas características de API Gateway, consulte [Configuración de la integración del proxy para un recurso de proxy](#).

Paso siguiente

[Paso 1: Preparación \(p. 277\)](#)

Paso 1: Preparación

Asegúrese de haber completado los pasos siguientes:

- Inscribirse en una cuenta de AWS y crear un usuario administrador en ella.
- Instalar y configurar la AWS CLI.

Para obtener instrucciones, consulte [Paso 1: Configuración de una cuenta de AWS y de la AWS CLI \(p. 199\)](#).

Important

En este ejemplo se utiliza la región *region* para crear una API mediante Amazon API Gateway y una función de Lambda.

Paso siguiente

[Paso 2: Creación y prueba manual de una función de Lambda \(p. 278\)](#)

Paso 2: Creación y prueba manual de una función de Lambda

En esta sección, hará lo siguiente:

- Crear un paquete de implementación de la función de Lambda utilizando el código de muestra proporcionado.
- Crear un rol de IAM (rol de ejecución). En el momento de cargar el paquete de implementación, debe especificar un rol de IAM (rol de ejecución) que Lambda puede asumir para ejecutar la función en su nombre.
- Crear la función de Lambda y probarla manualmente.

Temas

- [Paso 2.1: Creación de un paquete de implementación \(p. 278\)](#)
- [Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 280\)](#)
- [Paso 2.3: Creación y prueba manual de la función de Lambda \(p. 281\)](#)

Paso 2.1: Creación de un paquete de implementación

En la lista Filter View, elija el lenguaje que desea utilizar para la función de Lambda. Aparece la sección correspondiente con código e instrucciones específicas para crear un paquete de implementación.

[Node.js](#)

Siga las instrucciones para crear un paquete de implementación para la función de AWS Lambda.

1. Abrir un editor de texto y, a continuación, copie el siguiente código.

```
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

El código de muestra es compatible con los tiempos de ejecución v6.10 o v4.3 de Node.js.
Para obtener más información, consulte [Modelo de programación \(Node.js\) \(p. 9\)](#).

2. Guarde el archivo como `LambdaFunctionOverHttps.js`.
3. Comprima el archivo `LambdaFunctionOverHttps.js` como `LambdaFunctionOverHttps.zip`.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 280\)](#)

Python

Siga las instrucciones para crear un paquete de implementación para la función de AWS Lambda.

1. Abrir un editor de texto y, a continuación, copie el siguiente código.

Note

La instrucción `from __future__ import print_function` permite escribir código compatible con Python versión 2 o 3. Si utiliza la versión 3.6 del tiempo de ejecución, no es necesario incluirla.

```
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. Guarde el archivo como `LambdaFunctionOverHttps.py`.
3. Comprima el archivo `LambdaFunctionOverHttps.py` como `LambdaFunctionOverHttps.zip`.

Paso siguiente

Paso 2.2: Creación del rol de ejecución (rol de IAM) (p. 280)

Paso 2.2: Creación del rol de ejecución (rol de IAM)

En esta sección, creará un rol de IAM con el tipo de rol predefinido que se indica a continuación:

- Rol de servicio de AWS del tipo AWS Lambda: este rol concede permisos a AWS Lambda para asumir el rol.

Para obtener más información acerca de los roles de IAM, consulte [IAM Roles](#) en la Guía del usuario de IAM. Utilice el siguiente procedimiento para crear el rol de IAM.

Para crear un rol de IAM (rol de ejecución)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.
2. Siga los pasos de [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM para crear un rol de IAM (rol de ejecución). Cuando siga los pasos para crear un rol, tenga en cuenta lo siguiente:
 - En Role Name, utilice un nombre que sea único dentro de la cuenta de AWS (por ejemplo, `lambda-gateway-execution-role`).
 - En Select Role Type, elija AWS Service Roles, seguido de AWS Lambda. Esto garantiza que los permisos del servicio de AWS Lambda asuman el rol.
 - Puede crear un rol de IAM sin asociar una política de permisos en la consola. Después de crear el rol, actualícelo y, a continuación, asócielle la siguiente política de permisos.

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

3. Anote el nombre de recurso de Amazon (ARN) del rol. Lo necesitará en el siguiente paso al crear la función de Lambda.

Paso siguiente

[Paso 2.3: Creación y prueba manual de la función de Lambda \(p. 281\)](#)

Paso 2.3: Creación y prueba manual de la función de Lambda

En esta sección, hará lo siguiente:

- Crear una función de Lambda cargando el paquete de implementación.
- Pruebe la función de Lambda invocándola de forma manual y utilizando los datos del evento de muestra.

[Paso 2.3.1: Creación de una función de Lambda \(carga del paquete de implementación\)](#)

En este paso, cargará el paquete de implementación utilizando la AWS CLI.

En el símbolo del sistema, ejecute el siguiente comando `create-function` de la CLI de Lambda utilizando el perfil `adminuser`.

Debe actualizar el comando proporcionando la ruta del archivo .zip y el ARN del rol de ejecución. El valor del parámetro `--runtime` puede ser `python3.6`, `python2.7`, `nodejs6.10`, `nodejs4.3` o `java8`, dependiendo del lenguaje que utilice para crear el código.

```
$ aws lambda create-function \  
--region region \  
--function-name LambdaFunctionOverHttps \  
--zip-file fileb://file-path/LambdaFunctionOverHttps.zip \  
--role arn:aws:iam::account-id:role/LambdaRole
```

```
--role execution-role-arn \
--handler LambdaFunctionOverHttps.handler \
--runtime runtime-value \
--profile adminuser
```

Si lo desea, puede cargar el archivo .zip en un bucket de Amazon S3 en la misma región de AWS y, a continuación, especificar el nombre del objeto y el bucket en el comando anterior. Debe reemplazar el parámetro --zip-file por el parámetro --code, como se muestra a continuación:

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

Note

Puede crear una función de Lambda utilizando la consola de AWS Lambda, en cuyo caso debe tomar nota del valor de los parámetros del comando `create-function` de la AWS CLI. Debe proporcionar los mismos valores en la interfaz de usuario de la consola.

Paso 2.3.2: Prueba de la función de Lambda (invocación manual)

Invoque la función de forma manual utilizando los datos del evento de muestra. Recomendamos que invoque la función utilizando la consola, ya que la interfaz de usuario de la consola proporciona una interfaz fácil de utilizar para examinar los resultados de la ejecución, incluido el resumen de ejecución, los logs escritos por el código y los resultados devueltos por la función (dado que la consola siempre realiza una ejecución síncrona, invoca la función de Lambda utilizando el tipo de invocación `RequestResponse`).

Para probar la función de Lambda (Consola de administración de AWS)

1. Siga los pasos del ejercicio de la introducción para crear e invocar la función de Lambda en [Paso 2.2: Invocación manual de la función de Lambda y verificación de los resultados, los logs y las métricas \(p. 206\)](#). Para el evento de ejemplo para la prueba, elija Hello World en Sample event template y, a continuación, sustituya los datos utilizando lo siguiente:

```
{
    "operation": "echo",
    "payload": {
        "somekey1": "somevalue1",
        "somekey2": "somevalue2"
    }
}
```

2. Para probar una de las operaciones dynamo, como, por ejemplo, `read`, cambie los datos de entrada por los siguientes:

```
{
    "operation": "read",
    "tableName": "the name of your stream table",
    "payload": {
        "Key": {
            "the primary key of the table": "the value of the key"
        }
    }
}
```

3. Verifique los resultados en la consola.

Para probar la función de Lambda (AWS CLI)

1. Copie el siguiente JSON en un archivo y guárdelo como `input.txt`.

```
{  
    "operation": "echo",  
    "payload": {  
        "somekey1": "somevalue1",  
        "somekey2": "somevalue2"  
    }  
}
```

- Ejecute el siguiente comando `invoke`:

```
$ aws lambda invoke \  
--invocation-type Event \  
--function-name LambdaFunctionOverHttps \  
--region region \  
--payload file://file-path/input.txt \  
--profile adminuser  
outputfile.txt
```

Note

En este tutorial de ejemplo, el mensaje se guarda en el archivo `outputfile.txt` si solicita la ejecución síncrona (`RequestResponse` como tipo de invocación). La función devuelve la cadena de mensaje de en el cuerpo de la respuesta. Si utiliza el tipo de invocación `Event`, no se devuelve ningún mensaje en el archivo de salida. En cualquier caso, el parámetro `outputfile.txt` es necesario.

Paso siguiente

[Paso 3: Creación y prueba de una API utilizando Amazon API Gateway \(p. 283\)](#)

Paso 3: Creación y prueba de una API utilizando Amazon API Gateway

En este paso, asociará la función de Lambda a un método de la API que ha creado utilizando Amazon API Gateway y probará la experiencia integral. Es decir, cuando se envíe una solicitud HTTPS a un método de la API, Amazon API Gateway invocará la función de Lambda.

En primer lugar, debe crear una API (`DynamoDBOperations`) utilizando Amazon API Gateway con un recurso (`DynamoDBManager`) y un método (`POST`). A continuación, asociará el método `POST` a la función de Lambda. Y, por último, probará la experiencia integral.

Paso 3.1: Creación de la API

Ejecute el siguiente comando `create-rest-api` para crear la API `DynamoDBOperations` para este tutorial.

```
$ aws apigateway create-rest-api \  
--name DynamoDBOperations \  
--region region \  
--profile profile
```

A continuación se muestra un ejemplo de respuesta:

```
{
```

```
{  
    "name": "DynamoDBOperations",  
    "id": "api-id",  
    "createdDate": 1447724091  
}
```

Anote el ID de la API.

También necesita el ID del recurso raíz de la API. Para obtener el ID, ejecute el comando `get-resources`.

```
$ aws apigateway get-resources \  
--rest-api-id api-id
```

A continuación se muestra un ejemplo de respuesta (en este momento, solo tiene el recurso raíz, pero añadirá más recursos en el siguiente paso):

```
{  
    "items": [  
        {  
            "path": "/",  
            "id": "root-id"  
        }  
    ]  
}
```

Paso 3.2: Creación de un recurso (DynamoDBManager) en la API

Ejecute el siguiente comando `create-resource` para crear un recurso (DynamoDBManager) en la API que ha creado en la sección anterior.

```
$ aws apigateway create-resource \  
--rest-api-id api-id \  
--parent-id root-id \  
--path-part DynamoDBManager
```

A continuación se muestra un ejemplo de respuesta:

```
{  
    "path": "/DynamoDBManager",  
    "pathPart": "DynamoDBManager",  
    "id": "resource-id",  
    "parentId": "root-id"  
}
```

Anote el ID de la respuesta. Es el ID del recurso (DynamoDBManager) que ha creado.

Paso 3.3: Creación del método (POST) en el recurso

Ejecute el siguiente comando `put-method` para crear un método (POST) en el recurso (DynamoDBManager) de la API (DynamoDBOperations).

```
$ aws apigateway put-method \  
--rest-api-id api-id \  
--resource-id resource-id \  
--http-method POST \  
--authorization-type NONE
```

Especificamos `NONE` para el parámetro `--authorization-type`, lo que significa que se admiten solicitudes sin autenticar para este método. Esto está bien para las pruebas, pero en producción debe utilizar la autenticación basada en claves o en roles.

A continuación se muestra un ejemplo de respuesta:

```
{  
    "apiKeyRequired": false,  
    "httpMethod": "POST",  
    "authorizationType": "NONE"  
}
```

Paso 3.4: Definición de la función de Lambda como destino para el método POST

Ejecute el siguiente comando para establecer la función de Lambda como punto de integración para el método `POST` (este es el método que Amazon API Gateway invoca cuando se realiza una solicitud HTTPS para el punto de enlace del método `POST`).

```
$ 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` es el ID de la API (`DynamoDBOperations`) que ha creado en Amazon API Gateway.
- `--resource-id` es el ID de recurso del recurso (`DynamoDBManager`) que ha creado en la API.
- `--http-method` es el método de API Gateway e `--integration-http-method` es el método que utiliza API Gateway para comunicarse con AWS Lambda.
- `--uri` es un identificador único para el punto de enlace al que Amazon API Gateway pueden enviar la solicitud.

A continuación se muestra un ejemplo de respuesta:

```
{  
    "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"  
}
```

Establezca el `content-type` de la respuesta de integración y de la respuesta del método `POST` en JSON como se indica a continuación:

- Ejecute el siguiente comando para establecer la respuesta del método `POST` en JSON. Este es el tipo de respuesta que devuelve el método de la API.

```
$ aws apigateway put-method-response \  
--rest-api-id api-id \  
--method POST
```

```
--resource-id resource-id \
--http-method POST \
--status-code 200 \
--response-models "{\"application/json\": \"Empty\"}"
```

- Ejecute el siguiente comando para establecer la respuesta de integración del método POST en JSON. Este es el tipo de respuesta que devuelve la función de Lambda.

```
$ aws apigateway put-integration-response \
--rest-api-id api-id \
--resource-id resource-id \
--http-method POST \
--status-code 200 \
--response-templates "{\"application/json\": \"\"}"
```

Paso 3.5: Implementación de la API

En este paso, implementará la API que ha creado en una etapa denominada prod.

```
$ aws apigateway create-deployment \
--rest-api-id api-id \
--stage-name prod
```

A continuación se muestra un ejemplo de respuesta:

```
{
  "id": "deployment-id",
  "createdDate": 1447726017
}
```

Paso 3.6: Concesión de permisos a Amazon API Gateway para invocar la función de Lambda

Ahora que ha creado una API con Amazon API Gateway y la ha implementado, puede probarla. En primer lugar, debe agregar permisos para que Amazon API Gateway pueda invocar la función de Lambda cuando se envía la solicitud HTTPS al método POST.

Para ello, debe añadir permisos a la política de permisos asociada a la función de Lambda. Ejecute el siguiente comando add-permission de AWS Lambda para conceder a la entidad principal del servicio de Amazon API Gateway (apigateway.amazonaws.com) permisos para invocar la función de Lambda (`LambdaFunctionForAPIGateway`).

```
$ aws lambda add-permission \
--function-name LambdaFunctionOverHttps \
--statement-id apigateway-test-2 \
--action lambda:InvokeFunction \
--principal apigateway.amazonaws.com \
--source-arn "arn:aws:execute-api:region:aws-acct-id:api-id/*/POST/DynamoDBManager"
```

Debe conceder este permiso para poder realizar pruebas (si va a Amazon API Gateway y elige Test para probar el método de la API, necesitará este permiso). Observe que --source-arn especifica un carácter comodín (*) como valor para la etapa (indica que solo es para pruebas). Esto le permite hacer pruebas sin implementar la API.

Ahora, ejecute el mismo comando de nuevo, pero esta vez conceda a la API implementada permisos para invocar la función de Lambda.

```
$ aws lambda add-permission \
--function-name LambdaFunctionOverHttps \
--statement-id apigateway-prod-2 \
--action lambda:InvokeFunction \
--principal apigateway.amazonaws.com \
--source-arn "arn:aws:execute-api:region:aws-acct-id:api-id/prod/POST/DynamoDBManager"
```

Puede conceder este permiso para que la API implementada tenga permisos para invocar la función de Lambda. Observe que --source-arn especifica prod, que es el nombre de la etapa que se utilizó al implementar la API.

Paso 3.7: Prueba de envío de una solicitud HTTPS

En este paso, puede enviar una solicitud HTTPS al punto de enlace del método POST. Puede utilizar Curl o un método (test-invocation-method) proporcionado por Amazon API Gateway.

Si desea probar las operaciones que admite la función de Lambda en una tabla de DynamoDB, primero debe crear una tabla en Amazon DynamoDB, LambdaTable (Id), donde Id es la clave hash de tipo cadena.

Si va a probar las operaciones echo y ping que admite la función de Lambda, no es necesario que cree la tabla de DynamoDB.

Puede utilizar comandos de la CLI de Amazon API Gateway para enviar una solicitud POST de HTTPS al punto de enlace del recurso (DynamoDBManager). Dado que ha implementado su Amazon API Gateway, puede utilizar Curl para invocar los métodos para realizar la misma operación.

La función de Lambda permite utilizar la operación create para crear un elemento en la tabla de DynamoDB. Para solicitar esta operación, utilice el siguiente JSON:

```
{
  "operation": "create",
  "tableName": "LambdaTable",
  "payload": {
    "Item": {
      "Id": "foo",
      "number": 5
    }
  }
}
```

Ejecute el comando test-invocation-method de Amazon API Gateway para enviar una solicitud del método POST de HTTPS al punto de enlace del recurso (DynamoDBManager) con el JSON anterior en el cuerpo de la solicitud.

```
$ aws apigateway test-invocation-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\"}}}"
```

También puede utilizar el siguiente comando Curl:

```
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
```

Para enviar la solicitud para la operación echo que admite la función de Lambda, puede utilizar la siguiente carga de solicitud:

```
{  
  "operation": "echo",  
  "payload": {  
    "somekey1": "somevalue1",  
    "somekey2": "somevalue2"  
  }  
}
```

Ejecute el comando `test-invoke-method` de la CLI de Amazon API Gateway para enviar una solicitud del método POST de HTTPS al punto de enlace del recurso (`DynamoDBManager`) utilizando el JSON anterior en el cuerpo de la solicitud.

```
$ 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\"}}"
```

También puede utilizar el siguiente comando Curl:

```
curl -X POST -d "{\"operation\":\"echo\", \"payload\":{\"somekey1\":\"somevalue1\", \"somekey2\":\"somevalue2\"}}" https://api-id.execute-api.region.amazonaws.com/prod/  
DynamoDBManager
```

Paso 4: Implementación con AWS SAM y AWS CloudFormation

En la sección anterior, ha utilizado la API de AWS Lambda para crear y actualizar una función de Lambda proporcionando un paquete de implementación en un archivo ZIP. Sin embargo, puede que este mecanismo no sea conveniente para automatizar los pasos de implementación de las funciones, o para coordinar implementaciones y actualizaciones con otros elementos de una aplicación sin servidor, como los orígenes de eventos y los recursos posteriores.

Puede utilizar AWS CloudFormation para especificar, implementar y configurar fácilmente aplicaciones sin servidor. AWS CloudFormation es un servicio que le ayuda a modelar y configurar recursos de Amazon Web Services, por lo que podrá dedicar menos tiempo a la administración de dichos recursos y más tiempo a centrarse en las aplicaciones que se ejecutan en AWS. Puede crear una plantilla que describa todos los recursos de AWS que desea (como funciones de Lambda y tablas de DynamoDB), y AWS CloudFormation se encargará del aprovisionamiento y la configuración de dichos recursos.

Además, puede utilizar AWS Serverless Application Model para expresar los recursos que componen la aplicación sin servidor. Estos tipos de recursos, como las API y las funciones de Lambda, son totalmente compatibles con AWS CloudFormation, y facilitan la tarea de definir e implementar aplicaciones sin servidor.

Para obtener más información, consulte [Implementación de aplicaciones basadas en Lambda \(p. 156\)](#).

Especificaciones para la aplicación de Amazon API Gateway

A continuación encontrará la plantilla SAM para esta aplicación. Copie el texto siguiente en un archivo `.yaml` y guárdelo junto al paquete ZIP que ha creado anteriormente. Tenga en cuenta que los valores de los parámetros `Handler` y `Runtime` deben coincidir con los que utilizó cuando creó la función en la sección anterior.

```
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
```

Implementación de la aplicación sin servidor

Para obtener información acerca de cómo empaquetar e implementar la aplicación sin servidor utilizando los comandos package y deploy, consulte [Empaquetado e implementación \(p. 162\)](#).

Uso de AWS Lambda como backend de aplicaciones móviles (origen de eventos personalizados: Android)

Puede utilizar AWS Lambda para alojar la lógica de backend para aplicaciones móviles. Es decir, parte del código de una aplicación móvil se puede ejecutar como funciones de Lambda. Esto le permite utilizar una lógica mínima en la propia aplicación móvil, lo que facilita el escalado y la actualización (por ejemplo, solo es necesario aplicar actualizaciones de código a la función de Lambda en lugar de tener que implementar actualizaciones de código en los clientes de la aplicación).

Después de crear la función de Lambda, puede invocarla desde la aplicación móvil utilizando los SDK para móviles de AWS, como AWS SDK para Android. Para obtener más información, consulte [Herramientas para Amazon Web Services](#).

Note

También puede invocar la función de Lambda a través de HTTP utilizando Amazon API Gateway (en lugar de usar cualquiera de los SDK de AWS). Amazon API Gateway añade una capa adicional entre los usuarios de dispositivos móviles y la lógica de la aplicación que permite hacer lo siguiente:

- Aplicar una limitación a determinados usuarios o solicitudes.
- Protegerse contra ataques de denegación de servicio distribuido.
- Proporcionar una capa de caché para almacenar en caché la respuesta de la función de Lambda.

Tenga en cuenta lo siguiente sobre cómo funciona la integración entre la aplicación móvil y AWS Lambda:

- Modelo push-event: este es un modelo (consulte [Mapeo de orígenes de eventos \(p. 134\)](#)) en el que la aplicación invoca la función de Lambda pasando los datos del evento como parámetro.
- Invocación síncrona o asíncrona: la aplicación puede invocar la función de Lambda y obtener una respuesta en tiempo real especificando RequestResponse como tipo de invocación (o utilizar el tipo

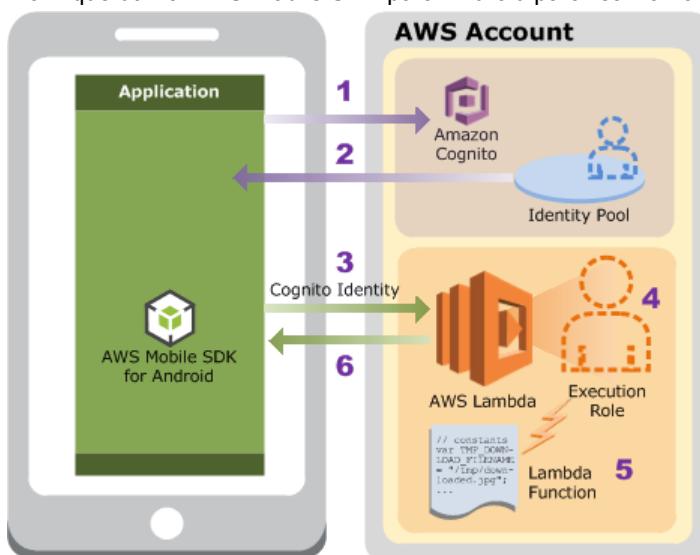
de invocación Event para la invocación asíncrona). Para obtener información acerca de los tipos de invocación, consulte [Administración de permisos mediante una política de función de Lambda \(p. 194\)](#).

- Estructura del evento: el evento que recibe la función de Lambda está definido por la aplicación, y la función de Lambda es el código personalizado escrito para procesar ese tipo de evento específico.

Tenga en cuenta que existen dos tipos de políticas de permisos con las que se trabaja al configurar la experiencia integral:

- Permisos para la función de Lambda: independientemente de qué invoque una función de Lambda, AWS Lambda ejecuta la función asumiendo el rol de IAM (rol de ejecución) que se especifique en el momento de crear la función de Lambda. La política de permisos asociada a este rol se utiliza para conceder a la función de Lambda los permisos que necesita. Por ejemplo, si la función de Lambda necesita leer un objeto, debe conceder permisos para las acciones de Amazon S3 correspondientes en la política de permisos. Para obtener más información, consulte [Administración de permisos mediante un rol de IAM \(rol de ejecución\) \(p. 193\)](#).
- Permisos para que la aplicación móvil invoque la función de Lambda: la aplicación debe tener credenciales y permisos de seguridad válidos para invocar una función de Lambda. Para las aplicaciones móviles, puede utilizar el servicio Amazon Cognito para administrar permisos, autenticaciones e identidades de usuario.

El siguiente diagrama ilustra el flujo de la aplicación (la ilustración supone que se trata de una aplicación móvil que utiliza AWS Mobile SDK para Android para realizar las llamadas a la API):



1. La aplicación móvil envía una solicitud a Amazon Cognito con un ID del grupo de identidades en la solicitud (el grupo de identidades se crea como parte de la configuración).
2. Amazon Cognito devuelve unas credenciales de seguridad temporales a la aplicación.

Amazon Cognito asume el rol asociado al grupo de identidades para generar las credenciales temporales. Lo que la aplicación puede hacer utilizando las credenciales depende de los permisos definidos en la política de permisos asociada al rol que utilizó Amazon Cognito para obtener las credenciales temporales.

Note

El SDK de AWS puede almacenar en caché las credenciales temporales para que la aplicación no envíe una solicitud a Amazon Cognito cada vez que necesite invocar una función de Lambda.

3. La aplicación móvil invoca la función de Lambda utilizando las credenciales temporales (identidad de Cognito).
4. AWS Lambda asume el rol de ejecución para ejecutar la función de Lambda en su nombre.
5. La función de Lambda se ejecuta.
6. AWS Lambda devuelve resultados a la aplicación móvil, suponiendo que la aplicación invocó la función de Lambda utilizando el tipo de invocación RequestResponse (invocación síncrona).

Para ver un tutorial que muestra una configuración de ejemplo, consulte [Tutorial: Uso de AWS Lambda como backend de aplicaciones móviles \(p. 291\)](#).

Tutorial: Uso de AWS Lambda como backend de aplicaciones móviles

En este tutorial, se crea una aplicación móvil sencilla para Android. El objetivo principal de este tutorial es mostrar cómo conectar varios componentes para permitir que una aplicación móvil para Android invoque una función de Lambda y procese la respuesta. La aplicación es sencilla, y supondremos lo siguiente:

- La aplicación móvil de ejemplo generará datos de eventos formados por un nombre (nombre y apellidos) con este formato:

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

- Puede utilizar una función de Lambda para procesar el evento. Es decir, la aplicación (que utiliza AWS Mobile SDK para Android) invoca una función de Lambda (`ExampleAndroidEventProcessor`) pasando los datos del evento. La función de Lambda de este tutorial hace lo siguiente:
 - Registra datos de eventos entrantes en Amazon CloudWatch Logs.
 - Tras una ejecución correcta, devuelve una cadena sencilla en el cuerpo de la respuesta. La aplicación móvil muestra el mensaje utilizando la clase `Toast` de Android.

Note

La forma en que la aplicación móvil invoca una función de Lambda como se muestra en este tutorial es un ejemplo del modelo solicitud-respuesta de AWS Lambda en el que una aplicación invoca una función de Lambda y, a continuación, recibe una respuesta en tiempo real. Para obtener más información, consulte [Modelo de programación \(p. 8\)](#).

Resumen de implementación

El tutorial se divide en dos secciones principales:

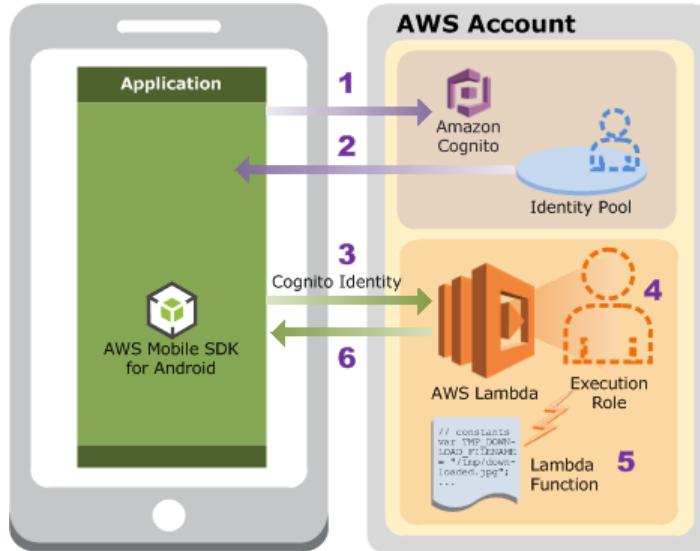
- En primer lugar, realice las configuraciones necesarias para crear una función de Lambda y pruébela invocándola de forma manual utilizando los datos del evento de muestra (no necesita la aplicación móvil para probar la función de Lambda).
- En segundo lugar, cree un grupo de identidades de Amazon Cognito para administrar la autenticación y los permisos, y cree la aplicación de ejemplo para Android. A continuación, ejecute la aplicación y esta invocará la función de Lambda. En este punto, puede verificar la experiencia integral. En este tutorial de ejemplo:
 - Puede utilizar el servicio Amazon Cognito para administrar permisos, autenticaciones e identidades de usuario. La aplicación móvil debe tener credenciales de seguridad y permisos válidos para invocar una función de Lambda. Como parte de la configuración de la aplicación, debe crear un grupo de identidades de Amazon Cognito para almacenar las identidades de usuario y definir permisos. Para obtener más información, consulte [Amazon Cognito](#)

- Esta aplicación móvil no requiere que los usuarios inicien sesión. Una aplicación móvil puede requerir a los usuarios que inicien sesión utilizando proveedores de identidad públicos, como Amazon y Facebook. El ámbito de este tutorial es limitado y supone que los usuarios de la aplicación móvil no están autenticados. Por lo tanto, al configurar el grupo de identidades de Amazon Cognito hará lo siguiente:
 - Permitir el acceso a identidades sin autenticar.

Amazon Cognito proporciona un identificador único y credenciales temporales de AWS para que estos usuarios invoquen la función de Lambda.

- En la política de permisos de acceso asociada al rol de IAM para usuarios sin autenticar, añada permisos para invocar la función de Lambda. Un grupo de identidades tiene dos roles de IAM asociados, uno para usuarios de la aplicación autenticados y otro para usuarios sin autenticar. En este ejemplo, Amazon Cognito asume el rol de los usuarios sin autenticar para obtener unas credenciales temporales. Cuando la aplicación utiliza estas credenciales temporales para invocar la función de Lambda, puede hacerlo únicamente si tiene los permisos necesarios (es decir, las credenciales pueden ser válidas, pero también se necesitan permisos). Para ello, actualice la política de permisos que utiliza Amazon Cognito para obtener las credenciales temporales.

En el diagrama siguiente se ilustra el flujo de la aplicación:



Ahora ya puede comenzar el tutorial.

Paso siguiente

[Paso 1: Preparación \(p. 292\)](#)

Paso 1: Preparación

Asegúrese de haber completado los pasos siguientes:

- Inscribirse en una cuenta de AWS y crear un usuario administrador en ella.
- Instalar y configurar la AWS CLI.

Para obtener instrucciones, consulte [Paso 1: Configuración de una cuenta de AWS y de la AWS CLI \(p. 199\)](#).

Note

El tutorial crea una función de Lambda y un grupo de identidades de Amazon Cognito en la región `us-east-1`. Si desea utilizar una región de AWS diferente, debe crear estos recursos en la misma región. También debe actualizar el código de ejemplo de la aplicación móvil y proporcionar la región específica que desea utilizar.

Paso siguiente

[Paso 2: Creación de la función de Lambda e invocación manual \(utilizando los datos del evento de muestra\) \(p. 293\)](#)

Paso 2: Creación de la función de Lambda e invocación manual (utilizando los datos del evento de muestra)

En esta sección, hará lo siguiente:

- Crear un paquete de implementación de la función de Lambda utilizando el código de muestra proporcionado. El código de la función de Lambda de muestra para procesar los eventos de la aplicación móvil se proporciona en varios lenguajes. Seleccione uno de los lenguajes y siga las instrucciones correspondientes para crear un paquete de implementación.

Note

Para ver más ejemplos de uso de otros servicios de AWS dentro de la función, incluyendo llamadas a otras funciones de Lambda, consulte [AWS SDK para JavaScript](#)

- Crear un rol de IAM (rol de ejecución). En el momento de cargar el paquete de implementación, debe especificar un rol de IAM (rol de ejecución). Este es el rol que asume AWS Lambda para invocar la función de Lambda en su nombre.
- Cree la función de Lambda cargando el paquete de implementación y, a continuación, pruébelo invocándola de forma manual utilizando los datos del evento de muestra.

Temas

- [Paso 2.1: Creación de un paquete de implementación \(p. 293\)](#)
- [Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 295\)](#)
- [Paso 2.3: Creación de la función de Lambda e invocación manual \(utilizando los datos del evento de muestra\) \(p. 296\)](#)

Paso 2.1: Creación de un paquete de implementación

En la lista Filter View, elija el lenguaje que desea utilizar para la función de Lambda. Aparece la sección correspondiente con código e instrucciones específicas para crear un paquete de implementación.

Node.js

Siga las instrucciones para crear un paquete de implementación para la función de AWS Lambda.

1. Abrir un editor de texto y, a continuación, copie el siguiente código.

```
exports.handler = function(event, context, callback) {
    console.log("Received event: ", event);
    var data = {
        "greetings": "Hello, " + event.firstName + " " + event.lastName + "."
    };
    callback(null, data);
}
```

Note

El código de muestra es compatible con los tiempos de ejecución v6.10 o v4.3 de Node.js.
Para obtener más información, consulte [Modelo de programación \(Node.js\) \(p. 9\)](#).

2. Guarde el archivo como `AndroidBackendLambdaFunction.js`.
3. Comprima el archivo `AndroidBackendLambdaFunction.js` como `AndroidBackendLambdaFunction.zip`.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 295\)](#)

Java

Utilice el siguiente código Java para crear la función de Lambda (`AndroidBackendLambdaFunction`). El código recibe datos de eventos de la aplicación de Android como primer parámetro del controlador. A continuación, el código procesa datos de eventos (a modo de ejemplo, este código escribe algunos de los datos de eventos en CloudWatch Logs y devuelve una cadena como respuesta).

En el código, el `handler (myHandler)` utiliza los tipos `RequestClass` y `ResponseClass` para entrada y salida. El código proporciona implementación para estos tipos.

Important

Puede utilizar las mismas clases (POJO) para administrar los datos de entrada y de salida al crear la aplicación móvil de ejemplo de la siguiente sección.

```
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);
}
}
```

Guarde el código anterior en un archivo (`HelloPojo.java`). Ahora puede crear un paquete de implementación. Debe incluir la siguiente dependencia:

- `aws-lambda-java-core`

El paquete de implementación puede ser un archivo `.zip` o `.jar` independiente. Puede utilizar cualquier herramienta de compilación y compresión con la que esté familiarizado para crear un paquete de implementación. Para ver ejemplos de cómo utilizar la herramienta de compilación Maven para crear un archivo `.jar` independiente, consulte [Creación de un paquete de implementación .jar utilizando Maven sin ningún IDE \(Java\) \(p. 70\)](#) y [Creación de un paquete de implementación .jar utilizando Maven y el IDE de Eclipse \(Java\) \(p. 72\)](#). Para ver un ejemplo de cómo utilizar la herramienta de compilación Gradle para crear un archivo `.zip`, consulte [Creación de un paquete de implementación .zip \(Java\) \(p. 75\)](#).

Después de verificar que se ha creado el paquete de implementación (`lambda-java-example-1.0-SNAPSHOT.jar`), consulte la siguiente sección para crear un rol de IAM (rol de ejecución). Especificará el rol al crear la función de Lambda.

Paso siguiente

[Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 295\)](#)

Paso 2.2: Creación del rol de ejecución (rol de IAM)

En esta sección, creará un rol de IAM con el tipo de rol y la política de acceso predefinidos que se indican a continuación:

- Rol de servicio de AWS del tipo AWS Lambda: este rol concede permisos a AWS Lambda para asumir el rol.
- `AWSLambdaBasicExecute`: es la política de permisos de acceso que se asocia al rol. Esta función de Lambda solo escribe logs en CloudWatch Logs. Por lo tanto, solo necesita permiso para acciones específicas de CloudWatch. Esta política proporciona estos permisos.

Para obtener más información acerca de los roles de IAM, consulte [IAM Roles](#) en la Guía del usuario de IAM. Utilice el siguiente procedimiento para crear el rol de IAM.

Para crear un rol de IAM (rol de ejecución)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.
2. Siga los pasos de [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM para crear un rol de IAM (rol de ejecución). Cuando siga los pasos para crear un rol, tenga en cuenta lo siguiente:
 - En Role Name, utilice un nombre que sea único dentro de la cuenta de AWS (por ejemplo, lambda-android-execution-role).
 - En Select Role Type, elija AWS Service Roles, seguido de AWS Lambda. Esto garantiza que los permisos del servicio de AWS Lambda asuman el rol.
 - En Attach Policy, seleccione AWSLambdaBasicExecute. Los permisos de esta política son suficientes para la función de Lambda de este tutorial.
3. Anote el ARN del rol. Lo necesitará en el siguiente paso al crear la función de Lambda.

Paso siguiente

[Paso 2.3: Creación de la función de Lambda e invocación manual \(utilizando los datos del evento de muestra\) \(p. 296\)](#)

Paso 2.3: Creación de la función de Lambda e invocación manual (utilizando los datos del evento de muestra)

En esta sección, hará lo siguiente:

- Crear una función de Lambda cargando el paquete de implementación.
- Probar la función de Lambda invocándola manualmente. En lugar de crear un origen de eventos, utilizará los datos del evento de muestra. En la siguiente sección, va a crear una aplicación móvil para Android y va a probar la experiencia integral.

[Paso 2.3.1: Creación de una función de Lambda \(carga del paquete de implementación\)](#)

En este paso, cargará el paquete de implementación utilizando la AWS CLI.

En el símbolo del sistema, ejecute el comando `create-function` de la CLI de Lambda utilizando `adminuser` para `profile`.

Debe actualizar el comando proporcionando la ruta del archivo .zip y el ARN del rol de ejecución. El valor del parámetro `--runtime` puede ser `nodejs6.10`, `nodejs4.3` o `java8`, dependiendo del lenguaje que elija para crear el código.

```
$ 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
```

Si lo desea, puede cargar el archivo .zip en un bucket de Amazon S3 en la misma región de AWS y, a continuación, especificar el nombre del objeto y el bucket en el comando anterior. Debe reemplazar el parámetro `--zip-file` por el parámetro `--code`, como se muestra a continuación:

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

Note

Puede crear una función de Lambda utilizando la consola de AWS Lambda, en cuyo caso debe tomar nota del valor de los parámetros del comando `create-function` de la AWS CLI. Debe proporcionar los mismos valores en la interfaz de usuario de la consola.

Paso 2.3.2: Prueba de la función de Lambda (invocación manual)

Invoca la función de forma manual utilizando los datos del evento de muestra. Recomendamos que invoque la función utilizando la consola, ya que la interfaz de usuario de la consola proporciona una interfaz fácil de utilizar para examinar los resultados de la ejecución, incluido el resumen de ejecución, los logs escritos por el código y los resultados devueltos por la función (dado que la consola siempre realiza una ejecución síncrona, invoca la función de Lambda utilizando el tipo de invocación `RequestResponse`).

Para probar la función de Lambda (Consola de administración de AWS)

1. Siga los pasos del ejercicio de la introducción para crear e invocar la función de Lambda en [Paso 2.2: Invocación manual de la función de Lambda y verificación de los resultados, los logs y las métricas \(p. 206\)](#). Después de elegir la función de Lambda, elija Configure test event en el menú Actions para especificar los siguientes datos del evento de muestra:

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

2. Verifique los resultados en la consola.
 - Execution result debe figurar como Succeeded con el siguiente valor de retorno:

```
{ "greetings": "Hello first-name, last-name." }
```
 - Revise las secciones Summary y Log output.

Para probar la función de Lambda (AWS CLI)

1. Guarde el siguiente evento de muestra JSON en un archivo, `input.txt`.

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

2. Ejecute el siguiente comando `invoke`:

```
$ aws lambda invoke \
--invocation-type Event \
--function-name AndroidBackendLambdaFunction \
--region us-east-1 \
--payload file://file-path/input.txt \
--profile adminuser
outputfile.txt
```

Note

En este tutorial de ejemplo, el mensaje se guarda en el archivo `outputfile.txt`. Si solicita una ejecución síncrona (con el tipo de invocación `RequestResponse`), la función devuelve la cadena de mensaje en el cuerpo de la respuesta.

Para Node.js, podría ser uno de los siguientes (lo que se haya especificado en el código):
`context.succeed("message")`

```
context.fail("message")
context.done(null, "message")
Para Java, es el mensaje de la instrucción return:
return "message"
```

Paso siguiente

[Paso 3: Creación de un grupo de identidades de Amazon Cognito \(p. 298\)](#)

Paso 3: Creación de un grupo de identidades de Amazon Cognito

En esta sección, va a crear un grupo de identidades de Amazon Cognito. El grupo de identidades tiene dos roles de IAM. Puede actualizar el rol de IAM para usuarios sin autenticar y otorgar permisos para ejecutar la función `AndroidBackendLambdaFunction` de Lambda.

Para obtener más información sobre las funciones de IAM, consulte [Funciones de IAM](#) en la Guía del usuario de IAM. Para obtener más información acerca de los servicios de Amazon Cognito, consulte la página de detalles del producto de [Amazon Cognito](#).

Para crear un grupo de identidades

1. Utilizando la URL de inicio de sesión de usuario de IAM, inicie sesión en la consola de Amazon Cognito como **adminuser**.
2. Cree un nuevo grupo de identidades denominado `JavaFunctionAndroidEventHandlerPool`. Antes de seguir el procedimiento para crear un grupo de identidades, tenga en cuenta lo siguiente:
 - El grupo de identidades que está creando debe permitir el acceso a identidades sin autenticar porque la aplicación móvil de ejemplo no requiere que el usuario inicie sesión (los usuarios de la aplicación no están autenticados). Por lo tanto, asegúrese de seleccionar la opción **Enable access to unauthenticated identities**.
 - Los usuarios sin autenticar que accedan a la aplicación necesitan permiso para invocar la función de Lambda. Para permitirlo, añada la siguiente instrucción a la política de permisos asociada a las identidades sin autenticar (da permiso a la acción `lambda:InvokeFunction` de la función de Lambda específica). Debe actualizar el ARN del recurso proporcionando el ID de su cuenta.

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

La política resultante será la siguiente:

```
{
    "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

Puede actualizar la política en el momento de la creación del grupo de identidades. También puede actualizar la política después de crear el grupo de identidades, en cuyo caso, asegúrese de anotar el nombre del rol de IAM para los usuarios sin autenticar en la consola de Amazon Cognito. A continuación, vaya a la consola de IAM, busque el rol específico y edite la política de permisos de acceso.

Para obtener instrucciones acerca de cómo crear un grupo de identidades, inicie sesión en la [consola de Amazon Cognito](#) y siga el asistente New Identity Pool.

3. Anote el ID del grupo de identidades. Debe especificar este ID en la aplicación móvil que creará en la siguiente sección. La aplicación utiliza este ID cuando envía una solicitud a Amazon Cognito para solicitar credenciales de seguridad temporales.

Paso siguiente

[Paso 4: Creación de una aplicación móvil para Android \(p. 299\)](#)

Paso 4: Creación de una aplicación móvil para Android

Ahora puede crear una aplicación móvil sencilla para Android que genera eventos e invoca funciones de Lambda pasando los datos de eventos como parámetros.

Las siguientes instrucciones se han verificado utilizando Android studio.

1. Cree un nuevo proyecto de Android denominado `AndroidEventGenerator` utilizando la siguiente configuración:
 - Seleccione la plataforma Phone and Tablet.
 - Elija Blank Activity.
2. En el archivo `build.gradle` (`Module:app`), añada lo siguiente en la sección `dependencies`:

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

3. Compile el proyecto para que las dependencias necesarias se descarguen según sea necesario.
4. En el manifiesto de la aplicación para Android (`AndroidManifest.xml`), añada los siguientes permisos para que la aplicación se pueda conectar a Internet. Puede añadirlos justo antes de la etiqueta final `</manifest>`.

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

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

5. En `MainActivity`, añada las siguientes importaciones:

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

6. En la sección `package`, añada las dos clases siguientes (`RequestClass` y `ResponseClass`). Tenga en cuenta que el POJO es el mismo que el POJO que creó en la función de Lambda de la sección anterior.

- `RequestClass`. Las instancias de esta clase actúan como POJO (Plain Old Java Object) para datos de eventos que constan de nombre y apellidos. Si está utilizando el ejemplo de Java para la función de Lambda que creó en la sección anterior, este POJO es igual que el POJO que creó en el código de la función de Lambda.

```
package com.example....lambdaeventgenerator;
public class RequestClass {
    String firstName;
    String lastName;

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

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

    public RequestClass() {
    }
}
```

- `ResponseClass`

```
package com.example....lambdaeventgenerator;
public class ResponseClass {
    String greetings;

    public String getGreetings() {
        return greetings;
    }

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

    public ResponseClass(String greetings) {
        this.greetings = greetings;
    }
}
```

```
    }

    public ResponseClass() {
    }
}
```

7. En el mismo paquete, cree una interfaz denominada `MyInterface` para invocar la función de Lambda `AndroidBackendLambdaFunction`.

Note

La anotación `@LambdaFunction` en el código mapea el método de cliente específico a la función de Lambda del mismo nombre. Para obtener más información acerca de esta anotación, consulte [AWS Lambda](#) en la guía SDK de AWS Mobile para Android Developer Guide.

```
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. Para no complicar la aplicación, vamos a añadir código para invocar la función de Lambda en el controlador de eventos `onCreate()`. En `MainActivity`, añada el siguiente código hacia el final del código `onCreate()`.

```
// Create an instance of CognitoCachingCredentialsProvider
CognitoCachingCredentialsProvider cognitoProvider = new
    CognitoCachingCredentialsProvider(
        this.getApplicationContext(), "identity-pool-id", Regions.US_WEST_2);

// Create LambdaInvokerFactory, to be used to instantiate the Lambda proxy.
LambdaInvokerFactory factory = new LambdaInvokerFactory(this.getApplicationContext(),
    Regions.US_WEST_2, cognitoProvider);

// Create the Lambda proxy object with a default Json data binder.
// You can provide your own data binder by implementing
// LambdaDataBinder.
final MyInterface myInterface = factory.build(MyInterface.class);

RequestClass request = new RequestClass("John", "Doe");
// The Lambda function invocation results in a network call.
// Make sure it is not called from the main thread.
new AsyncTask<RequestClass, Void, ResponseClass>() {
    @Override
    protected ResponseClass doInBackground(RequestClass... params) {
        // invoke "echo" method. In case it fails, it will throw a
        // LambdaFunctionException.
        try {
            return myInterface.AndroidBackendLambdaFunction(params[0]);
        } catch (LambdaFunctionException lfe) {
            Log.e("Tag", "Failed to invoke echo", lfe);
            return null;
        }
    }
}
```

```
@Override
protected void onPostExecute(ResponseClass result) {
    if (result == null) {
        return;
    }

    // Do a toast
    Toast.makeText(MainActivity.this, result.getGreetings(),
    Toast.LENGTH_LONG).show();
}
}.execute(request);
```

- Ejecute el código y verifíquelo como se indica a continuación:

- `Toast.makeText()` muestra la respuesta devuelta.
- Verifique que CloudWatch Logs muestra el log creado por la función de Lambda. Debería mostrar los datos del evento (nombre y apellidos). También puede verificar esto en la consola de AWS Lambda.

Uso de AWS Lambda con eventos programados

Puede crear una función de Lambda y configurar AWS Lambda para que la ejecute de manera periódica. Puede especificar una frecuencia fija (por ejemplo, ejecutar una función de Lambda cada hora o cada 15 minutos), o puede especificar una expresión Cron. Para obtener más información sobre las expresiones de programación, consulte [Programación de expresiones con rate o cron \(p. 306\)](#).

Esta funcionalidad está disponible cuando se crea una función de Lambda mediante la consola de AWS Lambda o la AWS CLI. Para configurarla con la AWS CLI, consulte [Programación de la ejecución de una función de AWS Lambda mediante de la CLI de AWS](#). La consola ofrece CloudWatch Events - Schedule como origen de eventos. En el momento de crear una función de Lambda, puede seleccionar este origen de eventos y especificar un intervalo de tiempo.

Si ha realizado cambios manuales en los permisos de la función, es posible que necesite volver a aplicar el acceso de eventos programados a la función. Puede hacerlo con el siguiente comando de la CLI.

```
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:MyFunction
--region region
```

Note

Cada cuenta de AWS puede tener un máximo de 100 orígenes de eventos exclusivos de tipo CloudWatch Events - Schedule. Cada uno de ellos puede ser el origen de eventos de hasta cinco funciones de Lambda. Es decir, puede tener hasta 500 funciones de Lambda que se ejecuten de forma programada en su cuenta de AWS.

En la consola también hay un proyecto (lambda-canary) que utiliza el tipo de origen CloudWatch Events - Schedule. Con este proyecto, puede crear una función de Lambda de muestra y probar esta característica. El código de ejemplo del proyecto comprueba la existencia de una página web específica y de una cadena de texto determinada en ella. Si no se encuentra la página web o la cadena de texto, la función de Lambda genera un error.

Para ver un tutorial que muestra una configuración de ejemplo, consulte [Tutorial: Uso de AWS Lambda con eventos programados \(p. 303\)](#).

Tutorial: Uso de AWS Lambda con eventos programados

En este tutorial, aprenderá a hacer lo siguiente:

- Crear una función de Lambda utilizando el proyecto lambda-canary. También configurará la función de Lambda para que se ejecute cada minuto. Tenga en cuenta que si la función devuelve un error, AWS Lambda registra las métricas de error en CloudWatch.
- Configurar una alarma de CloudWatch en la métrica `Errors` de la función de Lambda para que publique un mensaje en el tema de Amazon SNS cuando AWS Lambda envíe las métricas de error a CloudWatch. Debe suscribirse a los temas de Amazon SNS para obtener notificaciones por correo electrónico. En este tutorial, hará lo siguiente para configurar esto:
 - Crear un tema de Amazon SNS.
 - Suscribirse al tema para poder recibir notificaciones por correo electrónico cuando se publique un mensaje nuevo en el tema.
 - En Amazon CloudWatch, configure una alarma en la métrica `Errors` de la función de Lambda para publicar un mensaje en el tema de SNS cuando se produzcan errores.

Paso siguiente

[Paso 1: Creación de una función de Lambda \(p. 303\)](#)

Paso 1: Creación de una función de Lambda

1. Inicie sesión en la Consola de administración de AWS y abra la consola de AWS Lambda en <https://console.aws.amazon.com/lambda/>.
2. Seleccione Create a Lambda function.
3. En Select blueprint, elija el proyecto lambda-canary.
4. En Configure triggers:
 - Elija CloudWatch Events - Schedule.
 - En Rule name, escriba un nombre (por ejemplo, **CheckWebsiteScheduledEvent**).
 - En Rule descripción, escriba una descripción (por ejemplo, **Disparador CheckWebsiteScheduledEvent**).
 - En Schedule expression, especifique **rate(1 minute)**. Tenga en cuenta que puede especificar el valor con el formato de expresión rate o cron. Todas las programaciones utilizan la zona horaria UTC, y la precisión mínima es de un minuto.

Note

Cuando se configura una expresión rate, la primera ejecución es inmediata, y las siguientes se realizan según la frecuencia programada. En el ejemplo anterior, las ejecuciones posteriores se realizarían cada minuto.

Para obtener más información sobre las expresiones de programación, consulte [Programación de expresiones con rate o cron \(p. 306\)](#).

- En Enable trigger, recomendamos que deje el disparador desactivado hasta que lo haya probado.
 - Seleccione Next.
5. En Configure function, haga lo siguiente:

- Especifique el nombre de la función de Lambda (por ejemplo, **CheckWebsitePeriodically**).
 - En Runtime, especifique Python 3.6, Python 2.7, Node.js 6.10 o Node.js 4.3, dependiendo del lenguaje que prefiera.
 - Revise el código proporcionado por la plantilla. Más adelante en este tutorial, actualizará el código de la función para que esta devuelva un error. Puede especificar una URL inexistente o sustituir un texto de búsqueda por una cadena que no esté en la página.
 - En Role*, elija Create new role from template(s).
 - En Role name, escriba un nombre para el rol.
 - En Policy templates, Lambda proporciona una lista de plantillas adicionales opcionales que amplían los permisos básicos de Lambda. Para este tutorial, puede dejar este campo en blanco, ya que la función de Lambda ya tiene el permiso de ejecución básico que necesita.
 - En Advanced settings, deje la configuración predeterminada y elija Next.
6. En Review, revise la configuración y elija Create Function.

Paso siguiente

[Paso 2: Prueba de la función de Lambda \(utilizando un evento de prueba de muestra\) \(p. 304\)](#)

Paso 2: Prueba de la función de Lambda (utilizando un evento de prueba de muestra)

1. Seleccione la función que creó en el paso anterior y, a continuación, elija Test.
2. En la página Input sample event, elija Scheduled Event en la lista Sample event.

Anote la hora del evento de muestra. Este valor cambiará a medida que AWS Lambda invoque la función a intervalos programados. El código de la función de Lambda de muestra registra esta hora en CloudWatch Logs.

3. Elija Save and test y compruebe que en la sección Execution result se indica que la ejecución ha sido correcta.

Paso siguiente

[Paso 3: Creación de un tema de Amazon SNS y suscripción al tema \(p. 304\)](#)

Paso 3: Creación de un tema de Amazon SNS y suscripción al tema

1. Cree un tema de SNS mediante la consola de Amazon SNS. Para obtener instrucciones, consulte [Creación de un tema](#) en la Guía para desarrolladores de Amazon Simple Notification Service.
2. Suscríbase al tema. Para este ejercicio, utilice el correo electrónico como protocolo de comunicación. Para obtener instrucciones, consulte [Suscripción a un tema](#) en la Guía para desarrolladores de Amazon Simple Notification Service.

Utilizará este tema de Amazon SNS en el siguiente paso al configurar una alarma de CloudWatch para que cuando AWS Lambda emita un error, la alarma publique una notificación en este tema.

Paso siguiente

[Paso 4: Configuración de una alarma de CloudWatch \(p. 305\)](#)

Paso 4: Configuración de una alarma de CloudWatch

Para configurar una alarma de CloudWatch, siga las instrucciones de [Crear o editar una alarma](#) en la Guía del usuario de Amazon CloudWatch. Cuando siga los pasos, tenga en cuenta lo siguiente:

- En Create Alarm (paso 1. Select Metric), elija Lambda Metrics, seguido de Errors (el campo Metric Name contiene Errors) para la función de Lambda que ha creado. Además, en la lista desplegable de estadísticas, cambie la configuración de Average a Sum.
- En Create Alarm (paso 2. Define Metric), defina el umbral de alarma Whenever: Errors is ≥ 1 y seleccione el tema Amazon SNS de la lista Send notification to.

Paso siguiente

[Paso 5: Nueva comprobación de la función de Lambda \(p. 305\)](#)

Paso 5: Nueva comprobación de la función de Lambda

Vuelva a probar la función de Lambda. Esta vez, actualice el código especificando la URL de una página web inexistente o una cadena de texto que no aparezca en la página. Esto hace que la función devuelva un error que AWS Lambda envía a las métricas de error de CloudWatch. CloudWatch publica este mensaje en el tema de Amazon SNS y el usuario recibe una notificación por correo electrónico.

Paso 6: Implementación con AWS SAM y AWS CloudFormation

En la sección anterior, ha utilizado la API de AWS Lambda para crear y actualizar una función de Lambda proporcionando un paquete de implementación en un archivo ZIP. Sin embargo, puede que este mecanismo no sea conveniente para automatizar los pasos de implementación de las funciones, o para coordinar implementaciones y actualizaciones con otros elementos de una aplicación sin servidor, como los orígenes de eventos y los recursos posteriores.

Puede utilizar AWS CloudFormation para especificar, implementar y configurar fácilmente aplicaciones sin servidor. AWS CloudFormation es un servicio que le ayuda a modelar y configurar recursos de Amazon Web Services, por lo que podrá dedicar menos tiempo a la administración de dichos recursos y más tiempo a centrarse en las aplicaciones que se ejecutan en AWS. Puede crear una plantilla que describa todos los recursos de AWS que desea (como funciones de Lambda y tablas de DynamoDB), y AWS CloudFormation se encargará del aprovisionamiento y la configuración de dichos recursos.

Además, puede utilizar AWS Serverless Application Model para expresar los recursos que componen la aplicación sin servidor. Estos tipos de recursos, como las API y las funciones de Lambda, son totalmente compatibles con AWS CloudFormation, y facilitan la tarea de definir e implementar aplicaciones sin servidor.

Para obtener más información, consulte [Implementación de aplicaciones basadas en Lambda \(p. 156\)](#).

Especificación para la aplicación de eventos programada

A continuación encontrará la plantilla SAM para esta aplicación. Copie el texto siguiente en un archivo .yaml y guárdelo junto al paquete ZIP que ha creado anteriormente. Asegúrese de que el valor del parámetro Runtime: coincida con el que eligió en la sección anterior.

```
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'
```

Implementación de la aplicación sin servidor

Para obtener información acerca de cómo empaquetar e implementar la aplicación sin servidor utilizando los comandos package y deploy, consulte [Empaquetado e implementación \(p. 162\)](#).

Programación de expresiones con rate o cron

Expresiones rate

```
rate(Value Unit)
```

Donde:

Value puede ser un número entero positivo.

Unit puede ser minute(s), hour(s), o day(s).

Por ejemplo:

| Ejemplo | Expresión rate |
|---|-----------------|
| Invocar la función de Lambda cada 5 minutos | rate(5 minutes) |
| Invocar la función de Lambda cada hora | rate(1 hour) |

| Ejemplo | Expresión rate |
|--|---------------------------|
| Invocar la función de Lambda cada siete días | <code>rate(7 days)</code> |

Tenga en cuenta lo siguiente:

- No se admiten frecuencias de menos de un minuto para rate.
- Si el valor es 1, la unidad debe ir en singular (por ejemplo, `rate(1 day)`); para otros valores, en plural (por ejemplo, `rate(5 days)`).

Expresiones cron

`cron(Minutes Hours Day-of-month Month Day-of-week Year)`

Todos los campos son obligatorios y la zona horaria siempre es UTC. En la tabla siguiente se describen estos campos.

| Campo | Valores | Comodines |
|--------------|----------------|---------------|
| Minutes | 0-59 | , - * / |
| Hours | 0-23 | , - * / |
| Day-of-month | 1-31 | , - * ? / L W |
| Month | 1-12 o JAN-DEC | , - * / |
| Day-of-week | 1-7 o SUN-SAT | , - * ? / L # |
| Year | 1970-2199 | , - * / |

En la siguiente tabla se describen los caracteres comodín.

| Carácter | Definición | Ejemplo |
|----------|--|--|
| / | Especifica incrementos | 0/15 en el campo Minutes hace que la ejecución se produzca cada 15 minutos. |
| L | Especifica "Último" | Si se utiliza en el campo Day-of-month, especifica el último día del mes. Si se utiliza en el campo Day-of-week, especifica el último día de la semana (sábado). |
| W | Especifica Weekday (día laborable) | Cuando se utiliza con una fecha, por ejemplo 5/w, especifica el día laborable más cercano al día 5 del mes. Si el quinto día cae en sábado, la ejecución se realiza el viernes. Si el quinto día cae en domingo, la ejecución se realiza el lunes. |
| # | Especifica el primero, segundo, tercero, cuarto o quinto día del mes | Si se especifica 3#2, significa: el segundo martes del mes (el martes es el tercer día de la semana de 7 días). |

| Carácter | Definición | Ejemplo |
|----------|--------------------------------|--|
| * | Especifica todos los valores | Si se utiliza en el campo Day-of-month, significa: todos los días del mes. |
| ? | Valor no especificado | Se utiliza junto con otro valor especificado. Por ejemplo, si se especifica una fecha determinada, pero no importa el día de la semana en que caiga. |
| - | Especifica intervalos | 10-12 significaría 10, 11 y 12. |
| , | Especifica valores adicionales | SUN, MON, TUE significa domingo, lunes y martes. |
| / | Especifica incrementos | 5/10 significa 5, 15, 25, 35, etc. |

En la siguiente tabla se muestran ejemplos comunes de expresiones cron.

| Ejemplo | Expresión cron |
|--|---|
| Invocar una función de Lambda a las 10:00 h (UTC) cada día | <code>cron(0 10 * * ? *)</code> |
| Invocar una función de Lambda a las 12:15 h (UTC) cada día | <code>cron(15 12 * * ? *)</code> |
| Invocar una función de Lambda a las 18:00 h (UTC) de lunes a viernes | <code>cron(0 18 ? * MON-FRI *)</code> |
| Invocar una función de Lambda a las 8:00 h (UTC) cada primer día de mes | <code>cron(0 8 1 * ? *)</code> |
| Invocar una función de Lambda cada 10 minutos de lunes a viernes | <code>cron(0/10 * ? * MON-FRI *)</code> |
| Invocar una función de Lambda cada 5 minutos de lunes a viernes entre las 8:00 h y las 17:55 h (UTC) | <code>cron(0/5 8-17 ? * MON-FRI *)</code> |
| Invocar una función de Lambda a las 9:00 h (UTC) el primer lunes de cada mes | <code>cron(0 9 ? * 2#1 *)</code> |

Tenga en cuenta lo siguiente:

- No se admiten expresiones cron que produzcan frecuencias superiores a un minuto.
- Uno de los valores de day-of-month o day-of-week debe ser un signo de interrogación (?).

Uso de AWS Lambda con aplicaciones de usuario personalizadas

Uno de los casos de uso para AWS Lambda consiste en procesar eventos generados por una aplicación del usuario. Para fines de demostración, no es necesario que escriba una aplicación del usuario que invoque la función de Lambda. En lugar de ello, el tutorial de esta sección proporciona datos del evento de muestra que puede utilizar para invocar la función de Lambda manualmente.

Cuando una aplicación del usuario invoca una función de Lambda, es un ejemplo del modelo solicitud-respuesta de AWS Lambda en el que una aplicación invoca una función de Lambda y recibe una respuesta en tiempo real. Para obtener más información, consulte [Funcionamiento \(p. 188\)](#).

Para ver un tutorial que muestra una configuración de ejemplo, consulte [Tutorial: Uso de AWS Lambda con aplicaciones de usuario personalizadas \(p. 309\)](#).

Tutorial: Uso de AWS Lambda con aplicaciones de usuario personalizadas

En este tutorial, utilizará la AWS CLI para crear e invocar una función de Lambda y examinar otras API de AWS Lambda.

Hará lo siguiente:

- Crear una función de Lambda para procesar un evento que reciba como parámetro. Puede utilizar el siguiente código de ejemplo de Node.js para crear la función de Lambda.

```
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

El código de muestra es compatible con el tiempo de ejecución v4.3 de Node.js. Para obtener más información, consulte [Modelo de programación \(Node.js\) \(p. 9\)](#).

La función es sencilla. Procesa datos de eventos entrantes registrándolos (estos logs están disponibles en Amazon CloudWatch) y, en el modelo de solicitud-respuesta, puede solicitar que los datos de log se devuelvan en la respuesta.

- Simular una aplicación del usuario que envíe un evento a la función de Lambda invocando la función de Lambda manualmente con los siguientes datos del evento de muestra.

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

Note

Este ejemplo es similar al ejercicio de introducción (consulte [Introducción \(p. 199\)](#)). Se diferencia en que el ejercicio de introducción proporciona una experiencia basada en la consola. La consola hace muchas cosas automáticamente, lo que simplifica la experiencia. Cuando se utiliza la AWS CLI, se practica realizando llamadas a la API, lo que puede ayudarle a comprender mejor las operaciones de AWS Lambda. Además de crear e invocar una función de Lambda, puede examinar otras API de Lambda.

Paso siguiente

[Paso 1: Preparación \(p. 310\)](#)

Paso 1: Preparación

Asegúrese de haber completado los pasos siguientes:

- Inscribirse en una cuenta de AWS y crear un usuario administrador en ella.
- Instalar y configurar la AWS CLI.

Para obtener instrucciones, consulte [Paso 1: Configuración de una cuenta de AWS y de la AWS CLI \(p. 199\)](#).

Paso siguiente

[Paso 2: Creación de una función de Lambda e invocación manual \(p. 310\)](#)

Paso 2: Creación de una función de Lambda e invocación manual

En esta sección, hará lo siguiente:

- Crear un paquete de implementación. Un paquete de implementación es un archivo .zip que contiene el código y las dependencias. Para este tutorial no hay dependencias, únicamente un ejemplo de código sencillo.
- Crear un rol de IAM (rol de ejecución). En el momento de cargar el paquete de implementación, debe especificar un rol de IAM (rol de ejecución) que Lambda puede asumir para ejecutar la función en su nombre.

También debe conceder a este rol los permisos que necesita la función de Lambda. El código de este tutorial escribe logs en Amazon CloudWatch Logs. Por lo tanto, debe conceder permisos para las acciones de CloudWatch. Para obtener más información, consulte [AWS Lambda Watch Logs](#).

- Cree una función de Lambda (`HelloWorld`) utilizando el comando `create-function` de la CLI. Para obtener más información sobre la API subyacente y los parámetros relacionados, consulte [CreateFunction \(p. 385\)](#).

Temas

- [Paso 2.1: Creación de un paquete de implementación de la función de Lambda \(p. 310\)](#)
- [Paso 2.2: Creación del rol de ejecución \(rol de IAM\) \(p. 311\)](#)
- [Paso 2.3: Creación de una función de Lambda \(p. 312\)](#)
- [Paso siguiente \(p. 312\)](#)

Paso 2.1: Creación de un paquete de implementación de la función de Lambda

Siga las instrucciones para crear un paquete de implementación para la función de AWS Lambda.

1. Abrir un editor de texto y, a continuación, copie el siguiente código.

```
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

El código de muestra es compatible con los tiempos de ejecución v6.10 o v4.3 de Node.js.
Para obtener más información, consulte [Modelo de programación \(Node.js\) \(p. 9\)](#).

2. Guarde el archivo como `helloworld.js`.
3. Comprima el archivo `helloworld.js` como `helloworld.zip`.

Note

Para ver más ejemplos de uso de otros servicios de AWS dentro de la función, incluyendo llamadas a otras funciones de Lambda, consulte [AWS SDK para JavaScript](#)

Paso 2.2: Creación del rol de ejecución (rol de IAM)

Cuando se ejecuta la función de Lambda de este tutorial, necesita permisos para escribir logs en Amazon CloudWatch. Puede conceder estos permisos mediante la creación de un rol de IAM (rol de ejecución). AWS Lambda asume este rol cuando ejecuta la función de Lambda en su nombre. En esta sección, creará un rol de IAM con el tipo de rol y la política de acceso predefinidos que se indican a continuación:

- Rol de servicio de AWS de tipo “AWS Lambda”. Este rol concede a AWS Lambda permiso para asumir el rol.
- Política de acceso “AWSLambdaBasicExecutionRole” que se asocia al rol. Esta política existente concede permisos que incluyen los de las acciones de Amazon CloudWatch que necesita la función de Lambda.

Para obtener más información sobre los roles de IAM, consulte [Roles de IAM](#) en la Guía del usuario de IAM.

En esta sección, creará un rol de IAM con el tipo de rol y la política de permisos de acceso predefinidos que se indican a continuación:

- Rol de servicio de AWS del tipo AWS Lambda: este rol concede permisos a AWS Lambda para asumir el rol.
- AWSLambdaBasicExecutionRole: es la política de permisos de acceso que se asocia al rol.

Para obtener más información acerca de los roles de IAM, consulte [IAM Roles](#) en la Guía del usuario de IAM. Utilice el siguiente procedimiento para crear el rol de IAM.

Para crear un rol de IAM (rol de ejecución)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de IAM en <https://console.aws.amazon.com/iam/>.

2. Siga los pasos de [Creación de un rol para delegar permisos a un servicio de AWS](#) en la Guía del usuario de IAM para crear un rol de IAM (rol de ejecución). Cuando siga los pasos para crear un rol, tenga en cuenta lo siguiente:
 - En Role Name, utilice un nombre que sea único dentro de la cuenta de AWS (por ejemplo, lambda-custom-app-execution-role).
 - En Select Role Type, elija AWS Service Roles, seguido de AWS Lambda. Esto garantiza que los permisos del servicio de AWS Lambda asuman el rol.
 - En Attach Policy, elija AWSLambdaBasicExecutionRole.
3. Anote el ARN del rol. Lo necesitará en el siguiente paso al crear la función de Lambda.

Paso 2.3: Creación de una función de Lambda

Ejecute el siguiente comando `create-function` de la CLI de Lambda para crear una función de Lambda. Debe proporcionar el paquete de implementación y el ARN del rol de IAM como parámetros. Observe que el parámetro `Runtime` utiliza `nodejs6.10`, pero también puede especificar `nodejs4.3`.

```
$ aws lambda create-function \
--region us-west-2 \
--function-name helloworld \
--zip-file fileb://file-path/helloworld.zip \
--role role-arn \
--handler helloworld.handler \
--runtime nodejs6.10 \
--profile adminuser
```

Si lo desea, puede cargar el archivo .zip en un bucket de Amazon S3 en la misma región de AWS y, a continuación, especificar el nombre del objeto y el bucket en el comando anterior. Debe reemplazar el parámetro `--zip-file` por el parámetro `--code`, como se muestra a continuación:

```
--code S3Bucket=bucket-name,S3Key=zip-file-object-key
```

Para obtener más información, consulte [CreateFunction \(p. 385\)](#). AWS Lambda crea la función y devuelve información de configuración de la función, como se muestra en el ejemplo siguiente:

```
{
  "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": ""
}
```

Paso siguiente

[Paso 3: Invocación de la función de Lambda \(AWS CLI\) \(p. 312\)](#)

Paso 3: Invocación de la función de Lambda (AWS CLI)

En esta sección, invocará la función de Lambda de forma manual utilizando el comando `invoke` de la AWS CLI.

```
$ aws lambda invoke \
--invocation-type RequestResponse \
--function-name helloworld \
--region us-west-2 \
--log-type Tail \
--payload '{"key1":"value1", "key2":"value2", "key3":"value3"}' \
--profile adminuser \
outputfile.txt
```

Si lo desea, puede guardar la carga en un archivo (por ejemplo, `input.txt`) y proporcionar el nombre de archivo como parámetro.

```
--payload file://input.txt \
```

El comando `invoke` anterior especifica `RequestResponse` como tipo de invocación, lo que devuelve una respuesta inmediatamente como respuesta a la ejecución de la función. También puede especificar `Event` como tipo de invocación para invocar la función de forma asíncrona.

Al especificar el parámetro `--log-type`, el comando también solicita la parte final del log creado por la función. Los datos de log de la respuesta están codificados en base64, como se muestra en la siguiente respuesta de ejemplo:

```
{
    "LogResult": "base64-encoded-log",
    "StatusCode": 200
}
```

En Linux y Mac, puede utilizar el comando `base64` para descodificar el log.

```
$ echo base64-encoded-log | base64 --decode
```

A continuación se muestra una versión descodificada de un ejemplo de log.

```
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
```

Para obtener más información, consulte [Invoke \(p. 423\)](#).

Dado que ha invocado la función utilizando el tipo de invocación `RequestResponse`, la función se ejecuta y devuelve el objeto que se ha pasado a `context.succeed()` en tiempo real cuando se hace la llamada. En este tutorial, verá el siguiente texto escrito en el archivo `outputfile.txt` que ha especificado en el comando de la CLI:

```
"value1"
```

Note

Puede ejecutar esta función porque está utilizando la misma cuenta de AWS para crear e invocar la función de Lambda. Sin embargo, si desea conceder permisos entre cuentas a otra cuenta

de AWS o conceder permisos a otro servicio de AWS para ejecutar la función, debe añadirlos a la política de permisos de acceso asociada a la función. El tutorial de Amazon S3, que utiliza Amazon S3 como origen de eventos (consulte [Tutorial: Uso de AWS Lambda con Amazon S3 \(p. 215\)](#)), concede a Amazon S3 esos permisos para invocar la función.

Puede monitorizar la actividad de la función de Lambda en la consola de AWS Lambda.

- Inicie sesión en la Consola de administración de AWS y abra la consola de AWS Lambda en <https://console.aws.amazon.com/lambda/>.

La consola de AWS Lambda muestra una representación gráfica de algunas de las métricas de CloudWatch en la sección Cloudwatch Metrics at a glance para la función.

- Para cada gráfico, también puede hacer clic en el enlace logs para ver directamente los logs de CloudWatch.

Paso siguiente

[Paso 4: Prueba de otros comandos de la CLI \(AWS CLI\) \(p. 314\)](#)

Paso 4: Prueba de otros comandos de la CLI (AWS CLI)

Paso 4.1: Listado de las funciones de Lambda de una cuenta

En esta sección, va a probar las operaciones de listado de funciones de AWS Lambda. Ejecute el siguiente comando `list-functions` de la AWS CLI para obtener una lista de las funciones que ha cargado.

```
$ aws lambda list-functions \
--max-items 10 \
--profile adminuser
```

Para ilustrar el uso de la paginación, el comando especifica el parámetro opcional `--max-items` para limitar el número de funciones que se devuelven en la respuesta. Para obtener más información, consulte [ListFunctions \(p. 437\)](#). A continuación se muestra un ejemplo de respuesta.

```
{
    "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",
            "Description": ""
        }
    ]
}
```

```
        "Runtime": "nodejs6.10",
        "Description": ""
    },
    ...
],
"NextMarker": null
}
```

Como respuesta, Lambda devuelve una lista de hasta 10 funciones. Si hay más funciones, `NextMarker` proporciona un marcador que puede utilizar en la siguiente solicitud `list-functions`; de lo contrario, el valor es null. El siguiente comando `list-functions` de la AWS CLI es un ejemplo de uso del parámetro `--next-marker`.

```
$ aws lambda list-functions \
--max-items 10 \
--marker value-of-NextMarker-from-previous-response \
--profile adminuser
```

Paso 4.2: Obtención de los metadatos y de una URL para descargar paquetes de implementación de funciones de Lambda cargados anteriormente

El comando `get-function` de la CLI de Lambda devuelve los metadatos de la función de Lambda y una URL prefijada que puede utilizar para descargar el archivo .zip (paquete de implementación) de la función que cargó para crearla. Para obtener más información, consulte [GetFunction \(p. 411\)](#).

```
$ aws lambda get-function \
--function-name helloworld \
--region us-west-2 \
--profile adminuser
```

A continuación se muestra un ejemplo de respuesta.

```
{
    "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": ""
    }
}
```

Si únicamente desea la información de configuración de la función (y no la URL prefijada), puede utilizar el comando `get-function-configuration` de la CLI de Lambda.

```
$ aws lambda get-function-configuration \
--function-name helloworld \
--region us-west-2 \
```

```
--profile adminuser
```

Paso siguiente

Paso 5: Eliminación de la función de Lambda y del rol de IAM (AWS CLI) (p. 316)

Paso 5: Eliminación de la función de Lambda y del rol de IAM (AWS CLI)

Ejecute el siguiente comando `delete-function` para eliminar la función `helloworld`.

```
$ aws lambda delete-function \
--function-name helloworld \
--region us-west-2 \
--profile adminuser
```

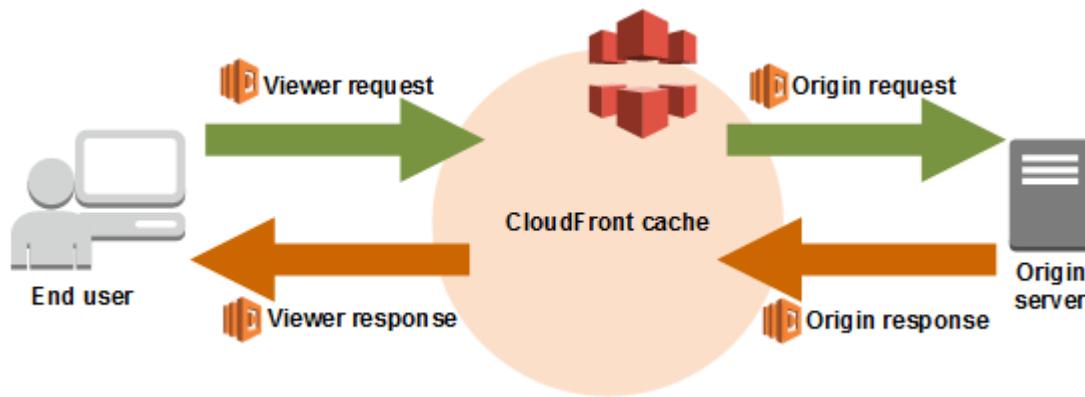
Eliminación del rol de IAM

Después de eliminar la función de Lambda, también puede eliminar el rol de IAM que ha creado en la consola de IAM. Para obtener información acerca de cómo eliminar un rol, consulte [Eliminación de roles o perfiles de instancia](#) en la Guía del usuario de IAM.

AWS Lambda@Edge

Lambda@Edge le permite ejecutar funciones de Lambda en regiones de AWS y ubicaciones de borde de Amazon CloudFront en respuesta a eventos de CloudFront, sin necesidad de aprovisionar ni administrar servidores. Puede utilizar funciones de Lambda para cambiar las solicitudes y las respuestas de CloudFront en los siguientes puntos:

- Despues de que CloudFront reciba una solicitud de un espectador (solicitud del espectador)
- Antes de que CloudFront reenvíe la solicitud al origen (solicitud al origen)
- Despues de que CloudFront reciba la respuesta del origen (respuesta del origen)
- Antes de que CloudFront reenvíe la respuesta al espectador (respuesta al espectador)



También puede generar respuestas a los espectadores sin necesidad de enviar la solicitud al origen.

Las funciones de Lambda para CloudFront se escriben en Node.js 6.10. Con Lambda@Edge, puede crear soluciones para llevar a cabo una gran variedad de operaciones, por ejemplo:

- Inspeccionar las cookies para rescribir URL en diferentes versiones de un sitio y poder realizar pruebas A/B.
- Enviar diferentes objetos a los usuarios en función del encabezado `User-Agent`, que contiene información sobre el dispositivo que envió la solicitud. Por ejemplo, puede enviar imágenes en diferentes resoluciones a los usuarios en función de sus dispositivos.
- Revisar los encabezados o los tokens autorizados, insertar el encabezado correspondiente y permitir el control de acceso antes de reenviar una solicitud al origen.
- Añadir, eliminar y modificar encabezados, así como reescribir la ruta de la URL para dirigir a los usuarios a distintos objetos de la caché.
- Generar nuevas respuestas HTTP para realizar tareas como redirigir a los usuarios sin autenticar a páginas de inicio de sesión, o crear y entregar páginas web estáticas desde el borde. Para obtener más información, consulte [Generación de respuestas HTTP](#) en la Guía para desarrolladores de Amazon CloudFront.

Para obtener más información acerca de cómo configurar CloudFront con Lambda@Edge, incluido el código de muestra, consulte [Usar CloudFront con Lambda@Edge](#) en la Guía para desarrolladores de Amazon CloudFront.

Temas

- [Cómo crear funciones de Lambda para Lambda@Edge \(p. 318\)](#)
- [Configuración de permisos y roles de IAM para Lambda@Edge \(p. 319\)](#)
- [Creación de una función de Lambda@Edge y de un disparador para un evento de CloudFront \(p. 321\)](#)
- [Adición de disparadores para una función de Lambda@Edge \(consola de AWS Lambda\) \(p. 322\)](#)
- [Creación de funciones para Lambda@Edge \(p. 323\)](#)
- [Edición de una función de Lambda para Lambda@Edge \(p. 325\)](#)
- [Comprobación y depuración \(p. 326\)](#)
- [Límites de Lambda@Edge \(p. 327\)](#)

Cómo crear funciones de Lambda para Lambda@Edge

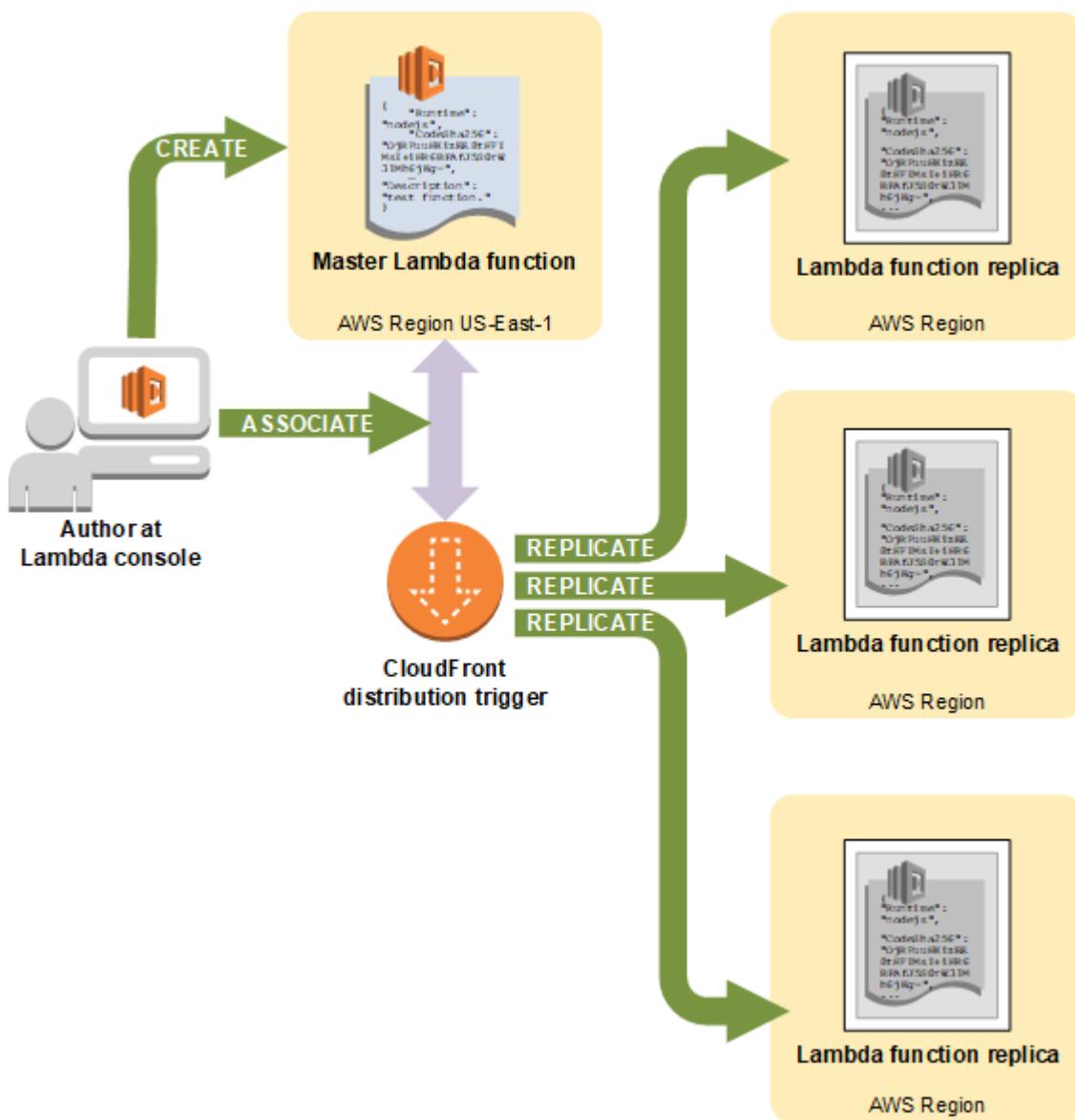
A continuación se muestra una descripción general de cómo crear funciones de Lambda para Lambda@Edge:

1. Utilice Node.js 6.10 para escribir el código para la función de Lambda.
2. Utilice la consola de AWS Lambda para crear una función de Lambda en la Región EE.UU. Este (Norte de Virginia). (También puede crear la función mediante programación, por ejemplo, utilizando uno de los SDK de AWS). Al crear la función, puede especificar los siguientes valores:
 - La distribución de CloudFront a la que desea que se aplique la función
 - Un comportamiento de la caché para la distribución
3. Publique una versión numerada de la función.

Si desea editar la función, debe hacerlo en la Región EE.UU. Este (Norte de Virginia). A continuación, publique una nueva versión numerada.

4. Especifique uno o varios eventos de CloudFront, conocidos como disparadores, que provoquen la ejecución de la función. Por ejemplo, puede crear un disparador que provoque la ejecución de la función cuando CloudFront reciba una solicitud de un espectador.

Al crear un disparador, Lambda replica la función en las regiones de AWS y las ubicaciones de borde de CloudFront de todo el mundo. Tenga en cuenta que las réplicas no se pueden editar ni eliminar.



Configuración de permisos y roles de IAM para Lambda@Edge

Para configurar Lambda@Edge, necesita permisos de IAM y un rol de ejecución de IAM:

Permisos de IAM necesarios para asociar funciones de Lambda a distribuciones de CloudFront

Además de los permisos de IAM necesarios para utilizar AWS Lambda, se necesitan los siguientes permisos de IAM para asociar funciones de Lambda a las distribuciones de CloudFront:

- lambda:GetFunction

Para el recurso, especifique el ARN de la versión de la función que desea ejecutar cuando se produzca un evento de CloudFront, tal como se muestra en el ejemplo siguiente:

```
arn:aws:lambda:us-east-1:123456789012:function:TestFunction:2
• lambda:EnableReplication*
```

Para el recurso, especifique el ARN de la versión de la función que desea ejecutar cuando se produzca un evento de CloudFront, tal como se muestra en el ejemplo siguiente:

```
arn:aws:lambda:us-east-1:123456789012:function:TestFunction:2
• iam:CreateServiceLinkedRole
• cloudfront:UpdateDistribution o bien cloudfront>CreateDistribution
```

Elija `cloudfront:UpdateDistribution` para actualizar una distribución o `cloudfront>CreateDistribution` para crear una distribución.

Para obtener más información, consulte la documentación siguiente:

- [Autenticación y control de acceso de AWS Lambda \(p. 337\)](#) en esta guía
- [Autenticación y control de acceso para CloudFront](#) en la Guía para desarrolladores de Amazon CloudFront

Rol de ejecución

Debe crear un rol de IAM que puedan asumir las entidades principales del servicio `lambda.amazonaws.com` y `edgelambda.amazonaws.com`. Este rol lo asumen las entidades principales del servicio cuando ejecutan la función. Para obtener más información, consulte [Creación de las funciones y políticas \(Conexión de la consola\)](#) en el tema "Funciones de trabajos de AWS Managed Policies" de la Guía del usuario de IAM.

A continuación se muestra un ejemplo de política de confianza de rol:

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

Para obtener información acerca de los permisos que debe conceder al rol de ejecución, consulte [Administración de permisos mediante un rol de IAM \(rol de ejecución\)](#) en la AWS Lambda Developer Guide. Tenga en cuenta lo siguiente:

- De forma predeterminada, siempre que un evento de CloudFront activa una función de Lambda, se escriben datos en CloudWatch Logs. Si desea utilizar estos logs, el rol de ejecución necesita permiso para escribir datos en CloudWatch Logs. Puede utilizar la política predefinida `AWSLambdaBasicExecutionRole` para conceder permisos al rol de ejecución.

Para obtener más información sobre CloudWatch Logs, consulte [Métricas de CloudWatch y CloudWatch Logs para funciones Lambda](#) en la Guía para desarrolladores de Amazon CloudFront.

- Si el código de la función de Lambda tiene acceso a otras operaciones de AWS, como, por ejemplo, la lectura de un objeto de un bucket de S3, el rol de ejecución necesita permiso para realizar esa operación.

Rol AWSServiceRoleForLambdaReplicator

Cuando se crea un disparador por primera vez, se crea automáticamente un rol denominado AWSServiceRoleForLambdaReplicator que permite a Lambda replicar funciones de Lambda@Edge en regiones de AWS. Este rol es necesario para utilizar Lambda@Edge. El ARN del rol AWSServiceRoleForLambdaReplicator tiene este aspecto:

```
arn:aws:iam::123456789012:role/aws-service-role/  
replicator.lambda.amazonaws.com/AWSServiceRoleForLambdaReplicator
```

Creación de una función de Lambda@Edge y de un disparador para un evento de CloudFront

Para configurar AWS Lambda de forma que ejecute funciones de Lambda en función de eventos de CloudFront, lleve a cabo el siguiente procedimiento:

Para crear una función de Lambda@Edge y un disparador para un evento de CloudFront

1. Inicie sesión en la Consola de administración de AWS y abra la consola de AWS Lambda en <https://console.aws.amazon.com/lambda/>.
2. Si ya tiene una o varias funciones de Lambda, elija Create a Lambda function.
Si no tiene ninguna función, elija Get Started Now.
3. En la lista de regiones de la parte superior de la página, elija US East (N. Virginia).
4. Elija Author from scratch.

En Basic information, haga lo siguiente:

- En Name*, especifique el nombre de la función de Lambda.
 - En Role*, elija Create new role from template(s).
 - Escriba un nombre para el nuevo rol en el campo Role Name.
 - En Policy Templates, elija Basic Edge Lambda permissions.
 - Elija Create function.
5. En Runtime, no puede elegir más que la opción Node.js 6.10.
 6. En la sección Lambda function code puede utilizar la opción Edit code inline para sustituir el código del controlador de la función de Lambda por código personalizado.
 7. Elija la pestaña Triggers. En la página Triggers, tiene la opción de elegir un servicio que activa automáticamente la función de Lambda; para ello, seleccione la sección Add trigger y, a continuación, elija el cuadro gris con puntos suspensivos (...) para mostrar una lista de los servicios disponibles. Es importante destacar que los eventos de CloudFront no se pueden asociar a versiones \$LATEST de la función ni a un alias. Para poder utilizar CloudFront debe elegir el botón Actions seguido de Publish new version. En la ventana Publish new version from \$LATEST, introduzca una descripción de la versión y, a continuación, haga clic en Publish.

Si decide utilizar uno de los proyectos cloudfront-* en lugar de la opción Author from scratch, de forma predeterminada se proporciona un disparador de CloudFront antes de crear la función. Utilice la opción Remove para eliminarlo antes de crear la función si no desea añadir un disparador inmediatamente.

Important

Si añade un disparador ahora, la función comenzará su ejecución después de terminar la replicación con las regiones de AWS y las ubicaciones de borde, y se implementará

la distribución correspondiente. Le recomendamos que pruebe y depure la función antes de añadir disparadores. Para obtener más información, consulte [Comprobación y depuración \(p. 326\)](#).

8. Para añadir un disparador:

- Seleccione de nuevo la pestaña Triggers, elija el cuadro con puntos junto al icono de Lambda y, por último, elija CloudFront.
- Escriba los siguientes valores.

Distribution ID

Elija el ID de la distribución donde desea añadir el disparador.

Cache behavior

Elija el comportamiento de la caché que especifica los objetos en los que desea ejecutar la función.

CloudFront event

Elija el evento de CloudFront que provoca la ejecución de la función.

Enable trigger and replicate

Seleccione esta casilla para que AWS Lambda replique la función a las regiones de todo el mundo.

9. Si desea añadir más disparadores para la misma función, consulte la documentación siguiente:

- [Adición de disparadores para una función de Lambda@Edge \(consola de AWS Lambda\) \(p. 322\)](#)
- [Añadir disparadores para eventos de CloudFront \(consola CloudFront\) en la Guía para desarrolladores de Amazon CloudFront](#)

10. En la sección Tags, añada etiquetas si procede.

Adición de disparadores para una función de Lambda@Edge (consola de AWS Lambda)

Cuando se crea una función de Lambda, solo se puede especificar un disparador, es decir, una combinación de distribución de CloudFront, comportamiento de la caché y evento que provoca la ejecución de la función. Puede añadir más disparadores a la misma función mediante la consola de Lambda o editando la distribución en la consola de CloudFront:

- Para utilizar la consola de Lambda, lleve a cabo el siguiente procedimiento. Este método funciona bien si desea añadir más disparadores a una función para la misma distribución de CloudFront.
- Para utilizar la consola de CloudFront, consulte [Añadir disparadores para eventos de CloudFront a una función Lambda](#) en la Guía para desarrolladores de Amazon CloudFront. Este método da buen resultado si desea añadir disparadores para varias distribuciones, ya que es más sencillo encontrar la distribución que desea actualizar. También puede actualizar otros ajustes de CloudFront simultáneamente.

Para añadir disparadores a una función de Lambda@Edge (consola de AWS Lambda)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de AWS Lambda en <https://console.aws.amazon.com/lambda/>.
2. En la lista de regiones de la parte superior de la página, elija US East (N. Virginia).
3. En la página Functions, elija el nombre de la función a la que deseé añadir disparadores.

4. Elija Qualifiers y, a continuación, seleccione la pestaña Versions.
5. Elija la versión a la que desea añadir disparadores.

Important

No puede crear disparadores para la versión \$LATEST, debe crearlos para una versión numerada.

Una vez elegida una versión, el nombre del botón cambia a Version: \$LATEST o Version: número de versión.

6. Elija la pestaña Triggers.
7. Elija Add triggers.
8. En el cuadro de diálogo Add trigger, elija el cuadro con puntos y, a continuación, CloudFront.

Note

Si ya ha creado uno o varios disparadores, CloudFront es el servicio predeterminado.

9. Especifique los siguientes valores para indicar cuándo desea que se ejecute la función de Lambda.

Distribution ID

Elija el ID de la distribución donde desea añadir el disparador.

Cache behavior

Elija el comportamiento de la caché que especifica los objetos en los que desea ejecutar la función.

CloudFront event

Elija el evento de CloudFront que provoca la ejecución de la función.

Enable trigger and replicate

Seleccione esta casilla para que AWS Lambda replique la función a las regiones de todo el mundo.

10. Elija Submit.

La función comienza a procesar solicitudes de los eventos de CloudFront especificados cuando la distribución de CloudFront actualizada se implementa. Para determinar si una distribución se ha implementado, seleccione Distributions en el panel de navegación. Cuando una distribución se implementa, el valor de la columna Status de la distribución cambia de In Progress a Deployed.

Creación de funciones para Lambda@Edge

El modelo de programación para utilizar Node.js con Lambda@Edge es el mismo que para utilizar Lambda en una región de AWS. Para obtener más información, consulte [Modelo de programación \(Node.js\) \(p. 9\)](#).

Para obtener más información acerca de cómo escribir funciones para Lambda@Edge, consulte [Requisitos y restricciones en funciones Lambda](#) en la Guía para desarrolladores de Amazon CloudFront.

Le recomendamos que incluya el parámetro `callback` y devuelva el objeto correspondiente:

- Eventos de solicitud: incluya el objeto `cf.request` en la respuesta.

Si está generando una respuesta, incluya el objeto `cf.response` en ella. Para obtener más información, consulte [Generación de respuestas HTTP](#).

- Eventos de respuesta: incluya el objeto `cf.response` en la respuesta.

Ejemplos

- [Ejemplo: Prueba A/B \(p. 324\)](#)
- [Ejemplo: Redirección HTTP \(p. 325\)](#)

Ejemplo: Prueba A/B

En el siguiente ejemplo se muestra cómo utilizar Lambda@Edge para pruebas A/B.

```
'use strict';

exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;
    const headers = request.headers;

    if (request.uri !== '/experiment-pixel.jpg') {
        // do not process if this is not an A-B test request
        callback(null, request);
        return;
    }

    const cookieExperimentA = 'X-Experiment-Name=A';
    const cookieExperimentB = 'X-Experiment-Name=B';
    const pathExperimentA = '/experiment-group/control-pixel.jpg';
    const pathExperimentB = '/experiment-group/treatment-pixel.jpg';

    /*
     * Lambda at the Edge headers are array objects.
     *
     * Client may send multiple Cookie headers, i.e.:
     * > GET /viewerRes/test HTTP/1.1
     * > User-Agent: curl/7.18.1 (x86_64-unknown-linux-gnu) libcurl/7.18.1 OpenSSL/1.0.1u
     * zlib/1.2.3
     * > Cookie: First=1; Second=2
     * > Cookie: ClientCode=abc
     * > Host: example.com
     *
     * You can access the first Cookie header at headers["cookie"][0].value
     * and the second at headers["cookie"][1].value.
     *
     * Header values are not parsed. In the example above,
     * headers["cookie"][0].value is equal to "First=1; Second=2"
     */
    let experimentUri;
    if (headers.cookie) {
        for (let i = 0; i < headers.cookie.length; i++) {
            if (headers.cookie[i].value.indexOf(cookieExperimentA) >= 0) {
                console.log('Experiment A cookie found');
                experimentUri = pathExperimentA;
                break;
            } else if (headers.cookie[i].value.indexOf(cookieExperimentB) >= 0) {
                console.log('Experiment B cookie found');
                experimentUri = pathExperimentB;
                break;
            }
        }
    }

    if (!experimentUri) {
        console.log('Experiment cookie has not been found. Throwing dice...');

        if (Math.random() < 0.75) {
            experimentUri = pathExperimentA;
        } else {
    
```

```
        experimentUri = pathExperimentB;
    }

    request.uri = experimentUri;
    console.log(`Request uri set to "${request.uri}"`);
    callback(null, request);
};
```

Ejemplo: Redirección HTTP

En el siguiente ejemplo se muestra cómo generar respuestas de redirección HTTP mediante funciones de Lambda que están asociadas a eventos de solicitud del espectador y de solicitud al origen de CloudFront. Si asocia la función a solicitudes al origen, la respuesta se almacena en la caché.

Note

Puede generar respuestas HTTP únicamente para los eventos de solicitud del espectador y de solicitud al origen. Para obtener más información, consulte [Generación de respuestas HTTP](#) en la Guía para desarrolladores de Amazon CloudFront.

```
'use strict';

exports.handler = (event, context, callback) => {
    /*
     * Generate HTTP redirect response with 302 status code and Location header.
     */
    const response = {
        status: '302',
        statusDescription: 'Found',
        headers: {
            location: [
                {
                    key: 'Location',
                    value: 'http://docs.aws.amazon.com/lambda/latest/dg/lambda-edge.html',
                },
            ],
        },
    };
    callback(null, response);
};
```

Edición de una función de Lambda para Lambda@Edge

Cuando desee editar una función de Lambda, tenga en cuenta lo siguiente:

- La versión original se denomina \$LATEST.
- Solo puede editar la versión \$LATEST.
- Cada vez que edite la versión \$LATEST, debe publicar una nueva versión numerada.
- No puede crear disparadores para \$LATEST.
- Cuando se publica una nueva versión de una función, Lambda no copia automáticamente los disparadores de la versión anterior a la nueva. Debe reproducir los disparadores para la nueva versión.
- Cuando se añade un disparador para un evento de CloudFront a una función, si ya existe un disparador para la misma distribución, comportamiento de la caché y evento para una versión anterior de la misma función, Lambda lo elimina de esta versión anterior.

Para editar una función de Lambda (consola de AWS Lambda)

1. Inicie sesión en la Consola de administración de AWS y abra la consola de AWS Lambda en <https://console.aws.amazon.com/lambda/>.
2. En la lista de regiones de la parte superior de la página, elija US East (N. Virginia).
3. En la lista de funciones, elija el nombre de la función que desea editar.

De forma predeterminada, la consola muestra la versión \$LATEST. Puede ver las versiones anteriores (elija Qualifiers), pero solo puede editar \$LATEST.

4. En la pestaña Code, en Code entry type, elija si desea editar el código en el navegador, cargar un archivo .zip o cargar un archivo desde Amazon S3.
5. Elija Save o Save and test.
6. Elija Actions y, a continuación, Publish new version.
7. En el cuadro de diálogo Publish new version from \$LATEST, introduzca una descripción de la nueva versión. Esta descripción aparece en la lista de versiones, junto con un número de versión generado de forma automática.
8. Elija Publish.

La nueva versión se convierte automáticamente la versión más reciente. El número de versión aparece en el botón Version de la esquina superior izquierda de la página.

9. Elija la pestaña Triggers.
10. Elija Add trigger.
11. En el cuadro de diálogo Add trigger, elija el cuadro con puntos y, a continuación, CloudFront.

Note

Si ya ha creado uno o varios disparadores para una función, CloudFront es el servicio predeterminado.

12. Especifique los siguientes valores para indicar cuándo desea que se ejecute la función de Lambda.

Distribution ID

Elija el ID de la distribución donde desea añadir el disparador.

Cache behavior

Elija el comportamiento de la caché que especifica los objetos en los que desea ejecutar la función.

CloudFront event

Elija el evento de CloudFront que provoca la ejecución de la función.

Enable trigger and replicate

Seleccione esta casilla para que Lambda replique la función a las regiones de todo el mundo.

13. Elija Submit.
14. Para añadir más disparadores para esta función, repita los pasos del 10 al 13.

Comprobación y depuración

Puede probar las funciones de Lambda@Edge en la consola de Lambda con eventos de prueba modelados en los eventos de CloudFront. Sin embargo, las pruebas en la consola solo validan la lógica, y no aplican límites de servicio específicos de Lambda@Edge.

Puede crear instrucciones de registro para las funciones de Lambda que se ejecutan en Lambda@Edge que escribirán en CloudWatch Logs. Para obtener más información, consulte [Métricas de CloudWatch y CloudWatch Logs para funciones Lambda](#).

Para obtener más información, consulte los siguientes temas en la Guía para desarrolladores de Amazon CloudFront:

- [Estructura de eventos](#)
- [Requisitos y restricciones en funciones Lambda](#)

Límites de Lambda@Edge

Debido al entorno de ejecución limitado, Lambda@Edge tiene ciertas restricciones además de los límites predeterminados de Lambda. Para obtener más información, consulte la documentación siguiente:

- [Límites de AWS Lambda \(p. 335\)](#) en esta guía
- [Requisitos y restricciones en funciones Lambda](#) en la Guía para desarrolladores de Amazon CloudFront

| Elemento | Límite |
|---|------------|
| Memoria | 128 MB |
| Tiempo de espera de los eventos de solicitud al origen y respuesta del origen de CloudFront. (La función puede realizar llamadas de red a recursos como buckets de S3, tablas de DynamoDB o instancias EC2 en regiones de AWS). | 3 segundos |
| Tiempo de espera de los eventos de solicitud del espectador y respuesta al espectador de CloudFront. (La función no puede realizar llamadas de red). | 1 segundo |
| Tamaño comprimido máximo de la función de Lambda y cualquier biblioteca incluida | 1 MB |

Registro de llamadas a la API de AWS Lambda mediante AWS CloudTrail

AWS Lambda está integrado con AWS CloudTrail, un servicio que captura las llamadas a la API realizadas por o en nombre de AWS Lambda en la cuenta de AWS y envía los archivos de log al bucket de Amazon S3 que se especifique. CloudTrail captura las llamadas a la API realizadas desde la consola de AWS Lambda o desde la API de AWS Lambda. Con la información recopilada por CloudTrail, puede identificar la solicitud que se realizó a AWS Lambda, la dirección IP de origen desde la que se realizó la solicitud, quién realizó la solicitud, cuándo se realizó, etc. Para obtener más información sobre CloudTrail, incluido cómo configurarlo y habilitarlo, consulte la [AWS CloudTrail User Guide](#).

Información de AWS Lambda en CloudTrail

Si el registro de CloudTrail está activado en la cuenta de AWS, las llamadas a la API realizadas con acciones de AWS Lambda se registran en archivos de log. Los registros de AWS Lambda se escriben junto con los de otros servicios de AWS en un archivo de log. CloudTrail determina cuándo crear y escribir en un nuevo archivo en función del período de tiempo y del tamaño del archivo.

Se admiten las siguientes acciones:

- [AddPermission \(p. 372\)](#)
- [CreateEventSourceMapping \(p. 380\)](#)
- [CreateFunction \(p. 385\)](#)

(Se omite el parámetro `ZipFile` de los logs de CloudTrail para `CreateFunction`).

- [DeleteEventSourceMapping \(p. 395\)](#)
- [DeleteFunction \(p. 398\)](#)
- [GetEventSourceMapping \(p. 408\)](#)
- [GetFunction \(p. 411\)](#)
- [GetFunctionConfiguration \(p. 415\)](#)
- [GetPolicy \(p. 420\)](#)
- [ListEventSourceMappings \(p. 434\)](#)
- [ListFunctions \(p. 437\)](#)
- [RemovePermission \(p. 452\)](#)
- [UpdateEventSourceMapping \(p. 462\)](#)
- [UpdateFunctionCode \(p. 466\)](#)

(Se omite el parámetro `ZipFile` de los logs de CloudTrail para `UpdateFunctionCode`).

- [UpdateFunctionConfiguration \(p. 472\)](#)

Cada entrada de log contiene información sobre quién generó la solicitud. La información de identidad del usuario que figura en el log le ayudará a determinar si la solicitud se hizo con credenciales de usuario de

IAM o raíz, con credenciales de seguridad temporales para un rol o un usuario federado, o mediante otro servicio de AWS. Para obtener más información, consulte el campo `userIdentity` en [Referencia de eventos de CloudTrail](#).

Puede almacenar los archivos de log en su bucket todo el tiempo que desee, y también puede definir reglas de ciclo de vida de Amazon S3 para archivar o eliminar archivos de log automáticamente. De forma predeterminada, los archivos de log se cifran mediante cifrado en el lado de servidor de Amazon S3 (SSE).

Puede hacer que CloudTrail publique notificaciones de Amazon SNS cuando se entreguen los nuevos archivos de log si desea actuar rápidamente al recibir un archivo de log. Para obtener más información, consulte [Configuración de notificaciones de Amazon SNS para CloudTrail](#).

También puede agregar archivos de log de AWS Lambda de varias regiones de AWS y de varias cuentas de AWS en un solo bucket de S3. Para obtener más información, consulte [Trabajar con archivos de registro de CloudTrail](#).

Información sobre las entradas de archivos de log de AWS Lambda

Los archivos de log de CloudTrail contienen una o varias entradas de log, cada una de las cuales está compuesta por varios eventos con formato JSON. Una entrada de log representa una única solicitud de cualquier origen e incluye información sobre la acción solicitada, los parámetros, la fecha y la hora de la acción, etcétera. No se garantiza que las entradas de log sigan un orden específico; es decir, no son un rastro de stack ordenado de las llamadas públicas a la API.

En el siguiente ejemplo se muestran entradas de log de CloudTrail para las acciones `GetFunction` y `DeleteFunction`.

```
{  
  "Records": [  
    {  
      "eventVersion": "1.03",  
      "userIdentity": {  
        "type": "IAMUser",  
        "principalId": "A1B2C3D4E5F6G7EXAMPLE",  
        "arn": "arn:aws:iam::999999999999:user/myUserName",  
        "accountId": "999999999999",  
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",  
        "userName": "myUserName"  
      },  
      "eventTime": "2015-03-18T19:03:36Z",  
      "eventSource": "lambda.amazonaws.com",  
      "eventName": "GetFunction",  
      "awsRegion": "us-east-1",  
      "sourceIPAddress": "127.0.0.1",  
      "userAgent": "Python-httplib2/0.8 (gzip)",  
      "errorCode": "AccessDenied",  
      "errorMessage": "User: arn:aws:iam::999999999999:user/myUserName" is  
not authorized to perform: lambda:GetFunction on resource: arn:aws:lambda:us-  
west-2:999999999999:function:other-acct-function",  
      "requestParameters": null,  
      "responseElements": null,  
      "requestID": "7aebcd0f-cda1-11e4-aaa2-e356da31e4ff",  
      "eventID": "e92a3e85-8ecd-4d23-8074-843aabfe89bf",  
      "eventType": "AwsApiCall",  
      "recipientAccountId": "999999999999"  
    },  
    {  
    }  
  ]  
}
```

```
"eventVersion": "1.03",
"userIdentity": {
    "type": "IAMUser",
    "principalId": "A1B2C3D4E5F6G7EXAMPLE",
    "arn": "arn:aws:iam::999999999999:user/myUserName",
    "accountId": "999999999999",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "myUserName"
},
"eventTime": "2015-03-18T19:04:42Z",
"eventSource": "lambda.amazonaws.com",
"eventName": "DeleteFunction",
"awsRegion": "us-east-1",
"sourceIPAddress": "127.0.0.1",
"userAgent": "Python-httplib2/0.8 (gzip)",
"requestParameters": {
    "functionName": "basic-node-task"
},
"responseElements": null,
"requestID": "a2198ecc-cda1-11e4-aaa2-e356da31e4ff",
"eventID": "20b84ce5-730f-482e-b2b2-e8fcc87ceb22",
"eventType": "AwsApiCall",
"recipientAccountId": "999999999999"
}
]
```

Note

eventName puede incluir información de fecha y versión, como "GetFunction20150331", pero sigue haciendo referencia a la misma API pública.

Prácticas recomendadas para trabajar con funciones de AWS Lambda

A continuación se indican las prácticas recomendadas para utilizar AWS Lambda:

Temas

- [Código de la función \(p. 331\)](#)
- [Función de configuración \(p. 332\)](#)
- [Alarms y métricas \(p. 332\)](#)
- [Invocaciones con eventos de flujo \(p. 333\)](#)
- [Invocaciones asíncronas \(p. 333\)](#)
- [VPC de Lambda \(p. 333\)](#)

Código de la función

- Separe el controlador de Lambda (punto de entrada) de la lógica básica. Esto le permite probar las distintas unidades de la función con mayor facilidad. En Node.js, podría tener este aspecto:

```
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
}
```

- Aproveche la reutilización de contenedores para mejorar el rendimiento de la función. Asegúrese de que las configuraciones o las dependencias externalizadas recuperadas por el código se almacenan y se hace referencias a ellas localmente después de la ejecución inicial. Limite la reinicialización de los objetos o las variables en cada invocación. En su lugar, utilice inicialización/constructor estáticos, variables globales/estáticas y singlettons. Mantenga activas y reutilice las conexiones (de HTTP, de base de datos, etc.) que se establecieron durante una invocación anterior.
- Utilice [Variables de entorno \(p. 97\)](#) para pasar parámetros operativos a la función. Por ejemplo, si está escribiendo en un bucket de Amazon S3, en lugar de codificar de forma rígida el nombre del bucket, configúrello como una variable de entorno.
- Controle las dependencias del paquete de implementación de la función. El entorno de ejecución de AWS Lambda contiene una serie de bibliotecas, como el SDK de AWS para los tiempos de ejecución de Node.js y Python (puede encontrar una lista completa aquí: [Entorno de ejecución de Lambda y bibliotecas disponibles \(p. 195\)](#)). Para disponer del conjunto más reciente de características y actualizaciones de seguridad, Lambda actualizará periódicamente estas bibliotecas. Estas actualizaciones pueden introducir cambios sutiles en el comportamiento de la función de Lambda. Para disponer de un control total de las dependencias que utiliza la función, recomendamos empaquetar todas las dependencias con el paquete de implementación.

- Minimice el tamaño del paquete de implementación de acuerdo con las necesidades de su tiempo de ejecución. Esto reducirá la cantidad de tiempo que tarda el paquete de implementación en descargarse y desempaquetarse antes de la invocación. En las funciones creadas en Java o .NET Core, evite cargar toda la biblioteca del SDK de AWS como parte del paquete de implementación. En lugar de ello, cree dependencias selectivas de los módulos que seleccionen los componentes del SDK que necesita (por ejemplo, DynamoDB, módulos del SDK de Amazon S3 y [bibliotecas básicas de Lambda](#)).
- Reduzca el tiempo que tarda Lambda en desempaquetar los paquetes de implementación escritos en Java colocando los archivos .jar de dependencias en un directorio /lib independiente. Esto es más rápido que colocar todo el código de la función en un único archivo jar con un gran número de archivos .class.
- Minimice la complejidad de las dependencias. Son preferibles los marcos de trabajo más sencillos, ya que se cargan rápidamente al arrancar el contenedor. Por ejemplo, es preferible utilizar los marcos de trabajo de inserción de dependencias (IoC) de Java más sencillos, como [Dagger](#) o [Guice](#), que los más complejos, como [Spring Framework](#).

Función de configuración

- La prueba de desempeño de la función de Lambda es una parte crucial a la hora de garantizar la elección de la configuración de memoria óptima. Cualquier aumento del tamaño de la memoria activa una aumento equivalente en la disponibilidad de CPU para la función. El uso de la memoria de la función se determina por invocación y puede visualizarse en [AWS CloudWatch Logs](#). En cada invocación se genera una entrada REPORT:, tal como se muestra a continuación:

```
REPORT RequestId: 3604209a-e9a3-11e6-939a-754dd98c7be3 Duration: 12.34 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 18 MB
```

Analice el campo **Max Memory Used**: para determinar si la función necesita más memoria o si ha aprovisionado en exceso el tamaño de la memoria para la función.

- Realice una prueba de carga de la función de Lambda para determinar un valor de tiempo de espera óptimo. Es importante analizar cuánto tiempo se ejecuta la función para determinar más fácilmente los posibles problemas con un servicio de dependencias que puedan aumentar la concurrencia de la función más allá de lo esperado. Esto es especialmente importante cuando la función de Lambda realiza llamadas de red a recursos que pueden no gestionar el escalado de Lambda.
- Utilice los permisos más restrictivos al establecer las políticas de IAM. Conozca los recursos y las operaciones que la función de Lambda necesita, y limite el rol de ejecución a estos permisos. Para obtener más información, consulte [Autenticación y control de acceso de AWS Lambda \(p. 337\)](#).
- Familiarícese con los [Límites de AWS Lambda \(p. 335\)](#). A menudo, el tamaño de carga, los descriptores de archivo y el espacio de /tmp se pasan por alto a la hora de determinar los límites de recursos del tiempo de ejecución.
- Elimine las funciones de Lambda que ya no utilice. Al hacerlo, las funciones no utilizadas no contarán innecesariamente para el límite de tamaño del paquete de implementación.

Alarmas y métricas

- Utilice [Métricas de AWS Lambda \(p. 125\)](#) y las [alarmas de CloudWatch](#) en lugar de crear o actualizar una métrica desde el código de la función de Lambda. Es una forma mucho más eficaz de realizar el seguimiento del estado de las funciones de Lambda, y le permitirá identificar los problemas al comienzo del proceso de desarrollo. Por ejemplo, puede configurar una alarma de acuerdo con la duración esperada del tiempo de ejecución de la función de Lambda para solucionar los cuellos de botella o las latencias atribuibles al código de la función.

- Utilice la biblioteca de registro y las [dimensiones y métricas de AWS Lambda](#) para detectar los errores de las aplicaciones (por ejemplo, ERR, ERROR, WARNING, etc.)

Invocaciones con eventos de flujo

- Pruebe con diferentes tamaños de lote y de registro para que la frecuencia de sondeo de cada origen de eventos se ajuste a la velocidad con la que la función es capaz de completar su tarea. [BatchSize](#) controla el número máximo de registros que se pueden enviar a la función en cada invocación. A menudo, un tamaño de lote mayor puede absorber de forma más eficiente la sobrecarga de invocaciones en un conjunto más grande de registros, aumentando el desempeño.

Note

Cuando no hay suficientes registros para procesar, en lugar de esperar, se invocará la función de procesamiento de flujos con un menor número de registros.

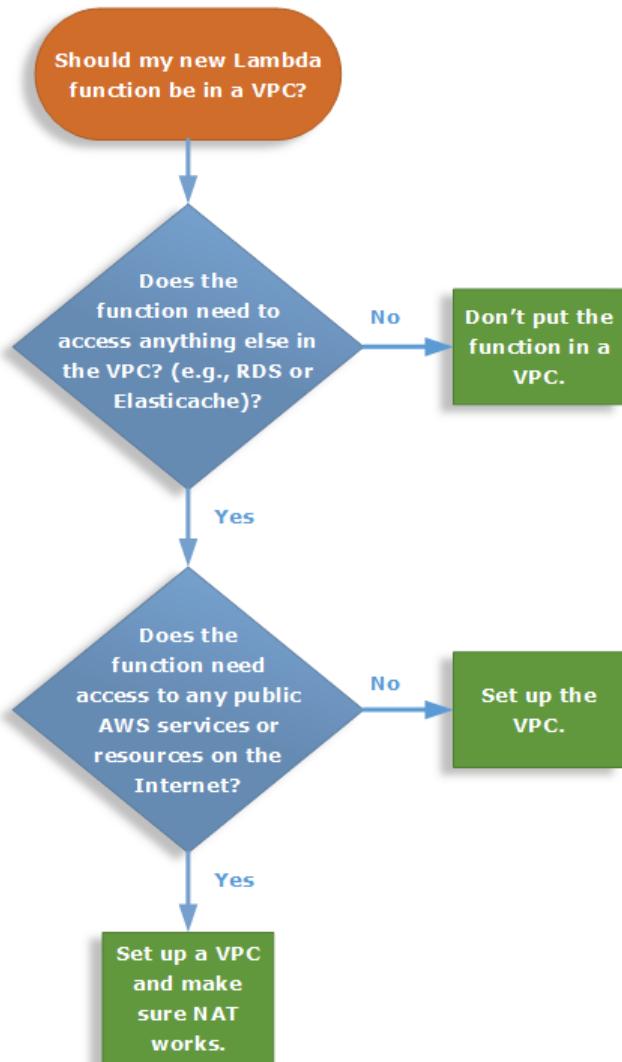
- Aumente el desempeño del procesamiento de flujos de Kinesis añadiendo fragmentos. Un flujo de Kinesis se compone de uno o varios fragmentos. Lambda sondeará cada fragmento que tenga como máximo una invocación simultánea. Por ejemplo, si el flujo tiene 100 fragmentos activos, habrá como máximo 100 invocaciones a la función de Lambda ejecutándose simultáneamente. Al aumentar el número de fragmentos, aumentará directamente el número máximo de invocaciones simultáneas a la función de Lambda, y también puede incrementarse el desempeño del procesamiento de flujos de Kinesis. Si aumenta el número de fragmentos de un flujo de Kinesis, asegúrese de que ha elegido una buena clave de partición ([Claves de partición](#)) para los datos, de forma que los registros relacionados acaben en los mismos fragmentos y los datos estén bien distribuidos.
- Utilice [Amazon CloudWatch](#) en IteratorAge para determinar si el flujo de Kinesis se está procesando. Por ejemplo, configure una alarma de CloudWatch con un valor máximo de 300 000 (30 segundos).

Invocaciones asíncronas

- Cree y utilice [Colas de mensajes fallidos \(p. 128\)](#) para solucionar y reproducir los errores de las funciones asíncronas.

VPC de Lambda

- El siguiente diagrama le guía a través de un árbol de decisiones sobre si debe utilizar una VPC (Virtual Private Cloud):



- No incluya la función de Lambda en una VPC a menos que se vea obligado a ello. No aportará ningún beneficio aparte de poder utilizarla para obtener acceso a recursos que no se pueden exponer públicamente, como una instancia privada de [Amazon Relational Database](#). Los servicios como Amazon Elasticsearch Service se pueden proteger a través de IAM mediante políticas de acceso, por lo que exponer el punto de enlace públicamente es seguro y no requiere que se ejecute la función en la VPC para protegerla.
- Lambda crea interfaces de red elásticas ([ENI](#)) en la VPC para obtener acceso a los recursos internos. Antes de solicitar un aumento de concurrencia, asegúrese de que dispone de capacidad de ENI (la fórmula para esta puede encontrarla aquí: [Configuración de una función de Lambda para acceder recursos en una Amazon VPC \(p. 108\)](#)) y de espacio de direcciones IP suficientes. Si no tiene suficiente capacidad de ENI, deberá solicitar un aumento. Si no tiene suficiente espacio de direcciones IP, es posible que necesite crear una subred de mayor tamaño.
- Cree subredes de Lambda dedicadas en la VPC:
 - Esto facilitará la aplicación de una tabla de ruteo personalizada para el tráfico de Gateway NAT sin cambiar las otras subredes públicas y privadas. Para obtener más información, consulte [Configuración de una función de Lambda para acceder recursos en una Amazon VPC \(p. 108\)](#)
 - También le permite dedicar un espacio de direcciones a Lambda sin compartirlo con otros recursos.

Límites de AWS Lambda

Como se ha explicado en [Funciones de Lambda \(p. 3\)](#), cuando haya empaquetado el código personalizado, incluidas las dependencias, y lo haya cargado en AWS Lambda, habrá creado una función de Lambda. Sin embargo, AWS Lambda impone ciertos límites, que afectan, por ejemplo, al tamaño del paquete de implementación o a la cantidad de memoria que se asigna la función de Lambda por cada invocación. En esta sección se explican dichos límites de AWS Lambda.

Temas

- [Límites de AWS Lambda \(p. 335\)](#)
- [Errores relacionados con los límites de AWS Lambda \(p. 336\)](#)

Límites de AWS Lambda

Límites de recursos de AWS Lambda por cada invocación

| Recurso | Límites |
|--|---|
| Intervalo de asignación de memoria | Mínimo = 128 MB/Máximo = 1 536 MB (en incrementos de 64 MB). Si se supera el uso de la máxima de memoria, se terminará la invocación de la función. |
| Capacidad de disco efímero (espacio “/tmp”) | 512 MB |
| Número de descriptores de archivo | 1 024 |
| Número de procesos y subprocesos (total combinado) | 1 024 |
| Duración máxima de la ejecución de cada solicitud | 300 segundos |
| Tamaño de carga del cuerpo de la solicitud Invoke (p. 423) (RequestResponse/invocación síncrona) | 6 MB |
| Tamaño de carga del cuerpo de la solicitud Invoke (p. 423) (Event/invocación asíncrona) | 128 K |

Límites de la cuenta de AWS Lambda por región

| Recurso | Límite predeterminado |
|--|-----------------------|
| Ejecuciones simultáneas (consulte Ejecuciones simultáneas de la función de Lambda (p. 189)) | 1 000 |

Para solicitar un aumento del límite de ejecuciones simultáneas

1. Abra la página [AWS Support Center](#), inicie sesión si es necesario y, a continuación, haga clic en Create case.
2. En Regarding, seleccione Service Limit Increase.
3. En Limit Type, seleccione Lambda, rellene los campos necesarios en el formulario y haga clic en el botón correspondiente al método de contacto que prefiera en la parte inferior de la página.

Note

AWS puede aumentar automáticamente el límite de ejecuciones simultáneas en su nombre para permitir que la función se adapte a la tasa de eventos entrantes, por ejemplo, cuando la función se activa desde un bucket de Amazon S3.

En la siguiente tabla se muestran los límites del servicio para implementar una función de Lambda.

Límites de implementación de AWS Lambda

| Elemento | Límite predeterminado |
|--|-----------------------|
| Tamaño del paquete de implementación de la función de Lambda (archivo comprimido .zip/.jar) | 50 MB |
| Tamaño total de todos los paquetes de implementación que se pueden cargar por cada región | 75 GB |
| Tamaño del código o de las dependencias que se pueden comprimir en un paquete de implementación (tamaño sin comprimir del contenido del archivo .zip/.jar). | 250 MB |
| Note Cada función de Lambda recibe 500 MB de espacio en disco no persistente en su propio directorio /tmp. El directorio /tmp se puede utilizar para cargar recursos adicionales, como bibliotecas de dependencias o conjuntos de datos, durante la inicialización de la función. | |
| Tamaño total de las variables de entorno establecidas | 4 KB |

Errores relacionados con los límites de AWS Lambda

Las funciones que sobrepasan cualquiera de los límites indicados anteriormente, producirán un error y generarán una excepción `exceeded limits`. Estos límites son fijos y no puede modificarse. Por ejemplo, si AWS Lambda genera la excepción `CodeStorageExceededException` o un mensaje de error similar a "Code storage limit exceeded", debe reducir el tamaño de almacenamiento utilizado por el código.

Para reducir el tamaño del almacenamiento del código

1. Elimine las funciones que ya no utiliza.
2. Reduzca el tamaño del código de las funciones que no desee eliminar. Puede encontrar el tamaño del código de una función de Lambda mediante la consola de AWS Lambda, la AWS Command Line Interface o los SDK de AWS.

Autenticación y control de acceso de AWS Lambda

El acceso a AWS Lambda requiere credenciales que AWS puede usar para autenticar las solicitudes. Estas credenciales deben tener permisos para obtener acceso a recursos de AWS como, por ejemplo, una función de AWS Lambda o un bucket de Amazon S3. En las secciones siguientes se incluye información detallada sobre cómo puede utilizar [AWS Identity and Access Management \(IAM\)](#) y Lambda para ayudar a proteger sus recursos controlando quién puede obtener acceso a ellos:

- Autenticación (p. 337)
- Control de acceso (p. 338)

Autenticación

Puede tener acceso a AWS como cualquiera de los siguientes tipos de identidades:

- Usuario raíz de la cuenta de AWS: Cuando crea por primera vez una cuenta de AWS, comienza únicamente por una identidad de inicio de sesión único que tiene acceso completo a todos los servicios y recursos de AWS de la cuenta. Esta identidad recibe el nombre de usuario raíz de la cuenta de AWS y se obtiene acceso a ella iniciando sesión con la dirección de correo electrónico y la contraseña que utilizó para crear la cuenta. Le recomendamos encarecidamente que no utilice el usuario raíz en sus tareas cotidianas, ni siquiera en las tareas administrativas. En lugar de ello, siga las [prácticas recomendadas para utilizar el usuario raíz únicamente cuando cree su primer usuario de IAM](#). A continuación, guarde en un lugar seguro las credenciales del usuario raíz y utilícelas solo para realizar unas pocas tareas de administración de servicios y cuentas.
- Usuario de IAM: un [usuario de IAM](#) es una identidad dentro de su cuenta de AWS que tiene permisos personalizados específicos (por ejemplo, permisos para crear a function in Lambda). Puede utilizar un nombre de usuario de IAM y una contraseña para iniciar sesión en páginas web seguras de AWS, como la de [Consola de administración de AWS](#), los [foros de discusión de AWS](#) o el [AWS Support Center](#).

Además de un nombre de usuario y una contraseña, también puede generar [claves de acceso](#) para cada usuario. Puede utilizar estas claves cuando obtenga acceso a los servicios de AWS mediante programación, ya sea a través de [uno de los varios SDK](#) o mediante la [AWS Command Line Interface \(CLI\)](#). El SDK y las herramientas de CLI usan claves de acceso para firmar criptográficamente su solicitud. Si no utiliza las herramientas de AWS, debe firmar usted mismo la solicitud. Lambda admite Signature Version 4, un protocolo para autenticar solicitudes de API de entrada. Para obtener más información sobre las solicitudes de autenticación, consulte [Signature Version 4 Signing Process](#) en la AWS General Reference.

- Rol de IAM: An [IAM role](#) is an IAM identity that you can create in your account that has specific permissions. It is similar to an IAM user, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. Los roles de IAM con credenciales temporales son útiles en las siguientes situaciones:

- Acceso de usuario federado: Instead of creating an IAM user, you can use existing user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an [identity provider](#). For more information about federated users, see [Federated Users and Roles](#) in the Guía del usuario de IAM.
- Acceso al servicio de AWS: You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see [Creating a Role to Delegate Permissions to an AWS Service](#) in the Guía del usuario de IAM.
- Aplicaciones que se ejecutan en Amazon EC2: You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances](#) in the Guía del usuario de IAM.

Control de acceso

Aunque disponga de credenciales válidas para autenticar las solicitudes, si no tiene permisos, no podrá crear recursos de AWS Lambda ni obtener acceso a ellos. Por ejemplo, debe disponer de permisos para crear una función de Lambda, añadir un origen de eventos y publicar una versión de una función de Lambda.

En las siguientes secciones se describe cómo administrar los permisos de AWS Lambda. Le recomendamos que lea primero la información general.

- [Información general sobre la administración de permisos de acceso a los recursos de AWS Lambda](#) (p. 338)
- [Uso de políticas basadas en identidad \(políticas de IAM\) para AWS Lambda](#) (p. 343)
- [Uso de políticas basadas en recursos para AWS Lambda \(políticas de funciones de Lambda\)](#) (p. 355)

Información general sobre la administración de permisos de acceso a los recursos de AWS Lambda

Cada recurso de AWS es propiedad de una cuenta de AWS, y los permisos para crear o tener acceso a un recurso se rigen por las políticas de permisos. Un administrador de la cuenta puede asociar políticas de permisos a identidades de IAM (es decir, usuarios, grupos y roles) y algunos servicios (como AWS Lambda) permiten también asociar políticas de permisos a recursos.

Note

Un administrador de la cuenta (o usuario administrador) es un usuario con privilegios de administrador. Para obtener más información, consulte la sección [IAM Best Practices](#) de la guía Guía del usuario de IAM.

Cuando concede permisos, decide quién debe obtener los permisos, para qué recursos se obtienen permisos y qué acciones específicas desea permitir en esos recursos.

Temas

- [Recursos y operaciones de AWS Lambda \(p. 339\)](#)
- [Titularidad de los recursos \(p. 339\)](#)
- [Administrar el acceso a los recursos \(p. 340\)](#)
- [Especificación de elementos de política: acciones, efectos, recursos y entidades principales \(p. 342\)](#)
- [Especificar condiciones en una política \(p. 342\)](#)

Recursos y operaciones de AWS Lambda

En AWS Lambda, los recursos principales son una función de Lambda y un mapeo de origen de eventos. Puede crear un mapeo de origen de eventos en el modelo de extracción de AWS Lambda para asociar una función de Lambda a un origen de eventos. Para obtener más información, consulte [Mapeo de orígenes de eventos \(p. 134\)](#).

AWS Lambda también admite otros tipos de recursos, el alias y la versión. Sin embargo, solo puede crear alias y versiones en el contexto de una función de Lambda existente. Estos elementos se denominan subrecursos.

Estos recursos y subrecursos tienen nombres de recursos de Amazon (ARN) únicos asociados a ellos, tal y como se muestra en la siguiente tabla:

| Tipo de recurso | Formato de ARN |
|----------------------------|---|
| Función | arn:aws:lambda: <i>región:id-de-cuenta</i> :function: <i>nombre-de-la-función</i> |
| Alias de función | arn:aws:lambda: <i>región:id-de-cuenta</i> :function: <i>nombre-de-la-función:nombre-de-alias</i> |
| Versión de función | arn:aws:lambda: <i>región:id-de-cuenta</i> :function: <i>nombre-de-la-función:versión</i> |
| Mapeo de origen de eventos | arn:aws:lambda: <i>región:id-de-cuenta</i> :event-source-mapping: <i>id-de-mapeo-de-origen-de-eventos</i> |

AWS Lambda cuenta con un conjunto de operaciones que permiten trabajar con los recursos de Lambda. Para ver la lista de las operaciones disponibles, consulte [Actions \(p. 370\)](#).

Titularidad de los recursos

El propietario de los recursos es la cuenta de AWS que ha creado el recurso. Es decir, el propietario de los recursos es la cuenta de AWS de la entidad principal (cuenta raíz, usuario de IAM o rol de IAM) que autentica la solicitud que crea el recurso. Los siguientes ejemplos ilustran cómo funciona:

- Si usa las credenciales de cuenta raíz de su cuenta de AWS para crear una función de Lambda, su cuenta de AWS es la propietaria del recurso (en Lambda, el recurso es la función de Lambda).
- Si crea un usuario de IAM en su cuenta de AWS y le concede permisos para crear una función de Lambda, el usuario podrá crear la función de Lambda. Sin embargo, su cuenta de AWS, a la que pertenece el usuario, será la propietaria del recurso de función de Lambda.

- Si crea un rol de IAM en su cuenta de AWS con permisos para crear una función de Lambda, cualquier persona que pueda asumir el rol podrá crear una función de Lambda. Su cuenta de AWS, a la que pertenece el rol, será la propietaria de la función de Lambda.

Administrar el acceso a los recursos

Una política de permisos describe quién tiene acceso a qué. En la siguiente sección, se explican las opciones disponibles para crear políticas de permisos.

Note

En esta sección se explica cómo se usa IAM en el contexto de AWS Lambda. No se proporciona información detallada sobre el servicio de IAM. Para ver la documentación completa de IAM, consulte [What Is IAM?](#) en la Guía del usuario de IAM. Para obtener más información sobre la sintaxis y descripciones de las políticas de IAM, consulte [AWS IAM Policy Reference](#) en la Guía del usuario de IAM.

Las políticas asociadas a una identidad de IAM se denominan políticas basadas en identidad (políticas de IAM) y las políticas asociadas a un recurso se denominan políticas basadas en recursos. AWS Lambda es compatible con las políticas basadas en identidad (políticas de IAM) y las políticas basadas en recursos.

Temas

- [Políticas basadas en identidad \(políticas de IAM\) \(p. 340\)](#)
- [Políticas basadas en recursos \(políticas de funciones de Lambda\) \(p. 341\)](#)

Políticas basadas en identidad (políticas de IAM)

Puede asociar políticas a identidades de IAM. Por ejemplo, puede hacer lo siguiente:

- Asociar una política de permisos a un usuario o un grupo de su cuenta: un administrador de la cuenta puede utilizar una política de permisos asociada a un usuario determinado para concederle permisos para crear una función de Lambda.
- Asociar una política de permisos a un rol (conceder permisos entre cuentas): puede asociar una política de permisos basada en identidad a un rol de IAM para conceder permisos entre cuentas. Por ejemplo, el administrador de la Cuenta A puede crear un rol para conceder permisos entre cuentas a otra cuenta de AWS (por ejemplo, a la Cuenta B) o a un servicio de AWS como se indica a continuación:
 1. El administrador de la Cuenta A crea un rol de IAM y asocia una política de permisos a dicho rol, que concede permisos sobre los recursos de la Cuenta A.
 2. El administrador de la Cuenta A asocia una política de confianza al rol que identifica la Cuenta B como la entidad principal que puede asumir el rol.
 3. A continuación, el administrador de la Cuenta B puede delegar permisos para asumir el rol a cualquier usuario de la Cuenta B. De este modo, los usuarios de la Cuenta B podrán crear recursos en la Cuenta A y obtener acceso a ellos. La entidad principal de la política de confianza también puede ser la entidad principal de un servicio de AWS si desea conceder permisos para asumir el rol a un servicio de AWS.

Para obtener más información sobre el uso de IAM para delegar permisos, consulte [Access Management](#) en la Guía del usuario de IAM.

A continuación, se muestra un ejemplo de política que concede permisos para la acción `Lambda:ListFunctions` en todos los recursos. En la implementación actual, Lambda no es compatible con la identificación de recursos concretos mediante los ARN de los recursos (también conocidos como permisos en el nivel de recursos) en algunas de las acciones de la API, por lo que deberá utilizar un carácter comodín (*).

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ListExistingFunctions",  
            "Effect": "Allow",  
            "Action": [  
                "lambda>ListFunctions"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Para obtener más información acerca del uso de políticas basadas en la identidad con Lambda, consulte [Uso de políticas basadas en identidad \(políticas de IAM\) para AWS Lambda \(p. 343\)](#). Para obtener más información acerca de los usuarios, grupos, roles y permisos, consulte [Identidades \(Users, Groups, and Roles\)](#) en la Guía del usuario de IAM.

Políticas basadas en recursos (políticas de funciones de Lambda)

Cada función de Lambda puede tener asociadas políticas de permisos basadas en recursos. Para Lambda, una función de Lambda es el recurso principal, y estas políticas se denominan políticas de funciones de Lambda. Puede utilizar una política de función de Lambda para conceder permisos entre cuentas como una alternativa a utilizar políticas basadas en la identidad con los roles de IAM. Por ejemplo, para conceder permisos a Amazon S3 para invocar una función de Lambda basta con añadir permisos a la política de función de Lambda en lugar de crear un rol de IAM.

Important

Las políticas de funciones de Lambda se utilizan principalmente cuando se está configurando un origen de eventos en AWS Lambda para conceder a un servicio o a un origen de eventos permisos para invocar una función de Lambda (consulte [Invoke \(p. 423\)](#)). Este no es el caso cuando un origen de eventos (como, por ejemplo, Amazon DynamoDB o Kinesis) utiliza el modelo de extracción, en el que los permisos se gestionan en el rol de ejecución de la función de Lambda. Para obtener más información, consulte [Mapeo de orígenes de eventos \(p. 134\)](#).

A continuación se muestra un ejemplo de política de función de Lambda que tiene una instrucción. La instrucción concede a la entidad principal del servicio de Amazon S3 permiso para realizar la acción `lambda:InvokeFunction` en una función de Lambda denominada `HelloWorld`. La condición garantiza que el bucket en el que se produjo el evento y la función de Lambda pertenecen a la misma cuenta.

```
{  
    "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":"65baafc90-6a1f-42a8-a7ab-8aa9bc877985",  
                "Condition":{  
                    "StringEquals":{  
                        "AWS:SourceAccount":"account-id"  
                    },  
                    "ArnLike":{  
                        "AWS:SourceArn":"arn:aws:s3:::ExampleBucket"  
                    }  
                }  
            }  
        ]  
    }  
}
```

```
        }  
    ]  
}  
}
```

Para obtener más información acerca del uso de políticas basadas en recursos con Lambda, consulte [Uso de políticas basadas en recursos para AWS Lambda \(políticas de funciones de Lambda\) \(p. 355\)](#). Para obtener más información acerca de cómo utilizar los roles de IAM (políticas basadas en identidad) y las políticas basadas en recursos, consulte [Cómo los roles de IAM difieren de las políticas basadas en recursos](#) en la Guía del usuario de IAM.

Especificación de elementos de política: acciones, efectos, recursos y entidades principales

Para cada recurso de AWS Lambda (consulte [Recursos y operaciones de AWS Lambda \(p. 339\)](#)), el servicio define un conjunto de operaciones de la API (consulte [Actions \(p. 370\)](#)). Para conceder permisos a estas operaciones de la API, Lambda define un conjunto de acciones que puede especificar en una política. Tenga en cuenta que la realización de una operación del API puede requerir permisos para más de una acción. Cuando se conceden permisos para acciones específicas, también debe identificar el recurso en el que las acciones se autorizan o deniegan.

A continuación se indican los elementos más básicos de la política:

- Recurso: en una política, se usa un nombre de recurso de Amazon (ARN) para identificar el recurso al que se aplica la política. Para obtener más información, consulte [Recursos y operaciones de AWS Lambda \(p. 339\)](#).
- Action: use palabras de clave de acción para identificar las operaciones del recurso que desea permitir o denegar. Por ejemplo, el permiso `lambda:InvokeFunction` concede a los usuarios permiso para realizar la operación `Invoke` de AWS Lambda.
- Efecto: especifique el efecto que se producirá cuando el usuario solicite la acción específica; puede ser permitir o denegar. Si no concede acceso de forma explícita (permitir) a un recurso, el acceso se deniega implícitamente. También puede denegar explícitamente el acceso a un recurso para asegurarse de que un usuario no pueda obtener acceso a él, aunque otra política le conceda acceso.
- Entidad principal: en las políticas basadas en identidad (políticas de IAM), el usuario al que se asocia esta política es la entidad principal implícita. Para las políticas basadas en recursos, debe especificar el usuario, la cuenta, el servicio u otra entidad que desee que reciba permisos (se aplica solo a las políticas basadas en recursos).

Para obtener más información sobre la sintaxis y descripciones de las políticas de IAM consulte [AWS IAM Policy Reference](#) de la Guía del usuario de IAM.

Para ver una tabla con todas las acciones de la API de AWS Lambda y los recursos a los que se aplican, consulte [Permisos de la API de Lambda: información sobre acciones, recursos y condiciones \(p. 359\)](#).

Especificar condiciones en una política

Al conceder permisos, puede utilizar el lenguaje de la política de IAM para especificar las condiciones en la que se debe aplicar una política. Por ejemplo, es posible que desee que solo se aplique una política después de una fecha específica. Para obtener más información sobre cómo especificar condiciones en un lenguaje de política, consulte [Condition](#) en la Guía del usuario de IAM.

Para expresar condiciones, se usan claves de condición predefinidas. No hay claves de condición específicas para Lambda. No obstante, existen claves de condición que se aplican a todo AWS que puede

utilizar cuando corresponda. Para ver una lista completa de claves generales de AWS, consulte [Claves disponibles para las condiciones](#) en la Guía del usuario de IAM.

Uso de políticas basadas en identidad (políticas de IAM) para AWS Lambda

En este tema, se ofrecen ejemplos de políticas basadas en identidad en las que un administrador de la cuenta puede asociar políticas de permisos a identidades de Identity and Access Management (IAM, Administración de identidades y accesos) (es decir, usuarios, grupos y roles).

Important

Recomendamos que consulte primero los temas de introducción en los que se explican los conceptos básicos y las opciones disponibles para administrar el acceso a sus recursos de AWS Lambda. Para obtener más información, consulte [Información general sobre la administración de permisos de acceso a los recursos de AWS Lambda](#) (p. 338).

En las secciones de este tema se explica lo siguiente:

- [Permisos necesarios para usar la consola de AWS Lambda](#) (p. 344)
- [Políticas administradas \(predefinidas\) de AWS para AWS Lambda](#) (p. 344)
- [Ejemplos de políticas administradas por el cliente](#) (p. 345)

A continuación, se muestra un ejemplo de una política de permisos.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "CreateFunctionPermissions",  
            "Effect": "Allow",  
            "Action": [  
                "lambda:CreateFunction"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "PermissionToPassAnyRole",  
            "Effect": "Allow",  
            "Action": [  
                "iam:PassRole"  
            ],  
            "Resource": "arn:aws:iam::account-id:role/*"  
        }  
    ]  
}
```

La política tiene dos instrucciones:

- La primera instrucción concede permisos para realizar la acción de AWS Lambda(`lambda:CreateFunction`) en un recurso utilizando el nombre de recurso de Amazon (ARN) de la función de Lambda. Actualmente, AWS Lambda no admite los permisos de esta acción concreta en el nivel de recurso. Por lo tanto, la política especifica un comodín (*) como valor de Resource.
- La segunda instrucción concede permisos para realizar la acción de IAM (`iam:PassRole`) con roles de IAM. El carácter comodín (*) usado al final del valor Resource significa que la instrucción concede

permisos para la acción `iam:PassRole` con cualquier rol de IAM. Para limitar este permiso a un rol específico, sustituya el carácter comodín (*) en el ARN del recurso por el nombre de rol específico.

La política no especifica el elemento `Principal`, ya que en una política basada en la identidad no se especifica la entidad principal que obtiene el permiso. Al asociar una política a un usuario, el usuario es la entidad principal implícita. Cuando se asocia una política de permisos a un rol de IAM, la entidad principal identificada en la política de confianza del rol obtiene los permisos.

Para ver una tabla con todas las acciones de la API de AWS Lambda y los recursos y las condiciones a los que se aplican, consulte [Permisos de la API de Lambda: información sobre acciones, recursos y condiciones \(p. 359\)](#).

Permisos necesarios para usar la consola de AWS Lambda

La consola de AWS Lambda ofrece un entorno integrado que permite crear y gestionar funciones de Lambda. La consola ofrece muchas características y flujos de trabajo que, a menudo, requieren permisos para crear una función de Lambda y permisos específicos de la API, los cuales se detallan en la [Permisos de la API de Lambda: información sobre acciones, recursos y condiciones \(p. 359\)](#). Para obtener más información acerca de estos permisos de consola adicionales, consulte [Permisos necesarios para usar la consola de AWS Lambda \(p. 348\)](#).

Políticas administradas (predefinidas) de AWS para AWS Lambda

AWS aborda muchos casos de uso comunes proporcionando políticas de IAM independientes creadas y administradas por AWS. Las políticas administradas por AWS conceden los permisos necesarios para casos de uso comunes, lo que le evita tener que investigar los permisos que se necesitan. Para obtener más información, consulte [AWS Managed Policies](#) en la Guía del usuario de IAM.

Las siguientes políticas administradas por AWS, que se pueden asociar a los usuarios de una cuenta, son específicas de AWS Lambda y están agrupadas por escenarios de casos de uso:

- `AWSLambdaReadOnlyAccess`: concede acceso de solo lectura a los recursos de AWS Lambda. Tenga en cuenta que esta política no concede permiso para la acción `Lambda:InvokeFunction`. Si desea que un usuario invoque una función de Lambda, también puede asociar la política administrada de AWS `AWSLambdaRole`.
- `AWSLambdaFullAccess`: concede acceso total a los recursos de AWS Lambda.
- `AWSLambdaRole`: concede permisos para invocar cualquier función de Lambda.

Note

Para consultar estas políticas de permisos, inicie sesión en la consola de IAM y busque las políticas específicas.

Además, existen otras políticas administradas por AWS que son adecuadas para su uso con el rol de IAM (rol de ejecución) que se especifica en el momento de crear una función de Lambda. Para obtener más información, consulte [Modelo de permisos de AWS Lambda \(p. 193\)](#).

También puede crear sus propias políticas personalizadas de IAM para conceder permisos sobre las acciones y los recursos de la API de AWS Lambda. Puede asociar estas políticas personalizadas a los usuarios o grupos de IAM que requieren esos permisos, o a los roles de ejecución personalizados (roles de IAM) que cree para sus recursos de Lambda.

Ejemplos de políticas administradas por el cliente

Los ejemplos que aparecen en esta sección muestran un grupo de políticas de ejemplo que puede asociar a un usuario. Si es la primera vez que crea una política, le recomendamos que en primer lugar cree un usuario de IAM en su cuenta y después asigne las políticas al usuario según se detalla en los pasos de esta sección.

Puede utilizar la consola para comprobar los efectos de cada política a medida que asocia la política al usuario. En un primer momento, como el usuario no tiene permisos, no podrá hacer nada más en la consola. Al asignar políticas al usuario, podrá verificar que este pueda realizar diversas acciones en la consola.

Le recomendamos que utilice dos ventanas del navegador: una para crear el usuario y concederle permisos y otra para iniciar sesión en la Consola de administración de AWS con las credenciales de usuario y comprobar los permisos a medida que se los concede al usuario.

Para ver ejemplos que ilustran cómo crear un rol de IAM que puede utilizar como rol de ejecución para una función de Lambda, consulte [Creación de roles de IAM](#) en la Guía del usuario de IAM.

Pasos de ejemplo

- [Paso 1: Crear un usuario de IAM \(p. 345\)](#)
- [Paso 2: Permitir a un usuario que muestre una lista de funciones de Lambda \(p. 345\)](#)
- [Paso 3: Permitir a un usuario que vea los detalles de una función de Lambda \(p. 346\)](#)
- [Paso 4: Permitir a un usuario que invoque una función de Lambda \(p. 346\)](#)
- [Paso 5: Permitir a un usuario que monitorice una función de Lambda y vea CloudWatch Logs \(p. 346\)](#)
- [Paso 6: Permitir a un usuario que cree una función de Lambda \(p. 347\)](#)

Paso 1: Crear un usuario de IAM

En primer lugar, cree un usuario de IAM, añádalo a un grupo de IAM con permisos administrativos y, a continuación, conceda permisos administrativos al usuario de IAM que ha creado. A continuación, podrá obtener acceso a AWS mediante una dirección URL especial y esas credenciales de usuario de IAM.

Para obtener instrucciones, consulte [Creación del primer grupo y usuario administrador de IAM](#) en la Guía del usuario de IAM.

Paso 2: Permitir a un usuario que muestre una lista de funciones de Lambda

Para poder ver algo en la consola, un usuario de IAM de su cuenta debe tener permisos para la acción `lambda>ListFunctions`. Cuando el usuario dispone de estos permisos, la consola puede mostrar la lista de funciones de Lambda de la cuenta de AWS que se han creado en la región de AWS concreta a la que pertenece el usuario.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ListExistingFunctions",  
            "Effect": "Allow",  
            "Action": [  
                "lambda>ListFunctions"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

```
        ]
    }
```

Paso 3: Permitir a un usuario que vea los detalles de una función de Lambda

Un usuario puede seleccionar una función de Lambda y ver los detalles de la función (como los alias, las versiones y demás información de configuración), siempre que tenga permisos para las siguientes acciones de AWS Lambda:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DisplayFunctionDetailsPermissions",
      "Effect": "Allow",
      "Action": [
        "lambda>ListVersionsByFunction",
        "lambda>ListAliases",
        "lambda:GetFunction",
        "lambda:GetFunctionConfiguration",
        "lambda>ListEventSourceMapping",
        "lambda:GetPolicy"
      ],
      "Resource": "*"
    }
  ]
}
```

Paso 4: Permitir a un usuario que invoque una función de Lambda

Si desea otorgar a un usuario permisos para invocar manualmente una función, debe concederle permisos para la acción `lambda:InvokeFunction`, tal como se muestra a continuación:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "InvokePermission",
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction"
      ],
      "Resource": "*"
    }
  ]
}
```

Paso 5: Permitir a un usuario que monitorice una función de Lambda y vea CloudWatch Logs

Cuando un usuario invoca una función de Lambda, AWS Lambda la ejecuta y devuelve los resultados. El usuario necesita permisos adicionales para monitorizar la función de Lambda.

Para permitir al usuario que vea las métricas de CloudWatch de la función de Lambda en la pestaña Monitoring de la consola, o en la vista de cuadrícula de la página de inicio de la consola, debe concederle los siguientes permisos:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "CloudWatchPermission",  
            "Effect": "Allow",  
            "Action": [  
                "cloudwatch:GetMetricStatistics"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Para permitir que un usuario haga clic en los enlaces a CloudWatch Logs en la consola de AWS Lambda y que vea la salida de log en CloudWatch Logs, debe concederle los siguientes permisos:

```
{  
    "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/*"  
        }  
    ]  
}
```

Paso 6: Permitir a un usuario que cree una función de Lambda

Si desea que un usuario pueda crear una función de Lambda, debe concederle los siguientes permisos. Los permisos para las acciones relacionadas con IAM son necesarios, ya que cuando un usuario crea una función de Lambda, debe seleccionar un rol de ejecución de IAM, que es el que AWS Lambda asume para ejecutar la función de Lambda.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "ListExistingRolesAndPolicies",  
            "Effect": "Allow",  
            "Action": [  
                "iam>ListRolePolicies",  
                "iam>ListRoles"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "CreateFunctionPermissions",  
            "Effect": "Allow",  
            "Action": [  
                "lambda>CreateFunction",  
                "lambda:ListFunctions",  
                "lambda:InvokeFunction",  
                "lambda:UpdateFunctionConfiguration",  
                "lambda:UpdateFunctionCode"  
            ],  
            "Resource": "arn:aws:lambda:region:account-id:function:function-name"  
        }  
    ]  
}
```

```
        "lambda>CreateFunction"
    ],
    "Resource": "*"
},
{
    "Sid": "PermissionToPassAnyRole",
    "Effect": "Allow",
    "Action": [
        "iam:PassRole"
    ],
    "Resource": "arn:aws:iam::account-id:role/*"
}
]
```

Si desea que un usuario pueda crear un rol de IAM al crear una función de Lambda, el usuario necesita permisos para realizar la acción `iam:PutRolePolicy`, como se muestra a continuación:

```
{
    "Sid": "CreateARole",
    "Effect": "Allow",
    "Action": [
        "iam>CreateRole",
        "iam>CreatePolicy",
        "iam:AttachRolePolicy"
    ],
    "Resource": "arn:aws:iam::account-id:role/*"
}
```

Important

Cada rol de IAM tiene asociada una política de permisos, que le concede permisos específicos. Independientemente de si el usuario crea un rol nuevo o utiliza uno existente, debe tener permisos para todas las acciones concedidas en la política de permisos asociada al rol. Debe conceder al usuario los permisos adicionales que sean necesarios.

Permisos necesarios para usar la consola de AWS Lambda

Para aprovechar la experiencia integrada proporcionada por la consola de AWS Lambda, dependiendo de lo que se desee que pueda hacer el usuario, a menudo deberá asignarle más permisos que los específicos de la API, que se describen en la tabla de referencias. Para obtener más información sobre las operaciones de la API de Lambda, consulte [Permisos de la API de Lambda: información sobre acciones, recursos y condiciones \(p. 359\)](#).

Por ejemplo, supongamos que concede a un usuario de IAM de su cuenta permisos para crear una función de Lambda para procesar eventos de creación de objetos de Amazon S3. Para permitir que el usuario configure Amazon S3 como origen de eventos, la lista desplegable de la consola mostrará una lista de buckets. Sin embargo, la consola solo puede mostrar la lista de buckets si el usuario que ha iniciado sesión tiene permisos para las acciones de Amazon S3 correspondientes.

En las siguientes secciones se describen los permisos adicionales necesarios para los distintos puntos de integración. Para obtener información acerca de los puntos de integración, consulte [Funcionamiento \(p. 188\)](#).

Si es la primera vez que utiliza la administración de permisos, le recomendamos que comience por el tutorial de ejemplo, en el que se crea un usuario de IAM, se le conceden permisos adicionales y se

verifica que los permisos funcionan mediante la consola de AWS Lambda (consulte [Ejemplos de políticas administradas por el cliente \(p. 345\)](#)).

Temas

- [Amazon API Gateway \(p. 349\)](#)
- [Amazon CloudWatch Events \(p. 350\)](#)
- [Amazon CloudWatch Logs \(p. 350\)](#)
- [Amazon Cognito \(p. 351\)](#)
- [Amazon DynamoDB \(p. 352\)](#)
- [Amazon Kinesis Data Streams \(p. 353\)](#)
- [Amazon S3 \(p. 354\)](#)
- [Amazon SNS \(p. 354\)](#)
- [AWS IoT \(p. 355\)](#)

Note

Todas estas políticas de permisos conceden a los servicios específicos de AWS permisos para invocar una función de Lambda. El usuario que configura esta integración debe tener permisos para invocar la función de Lambda. De lo contrario, no podrá definir la configuración. Para proporcionarle estos permisos al usuario, puede asociarle la política de permisos `AWSLambdaRole` administrada por AWS (predefinida).

Amazon API Gateway

Cuando se configura un punto de enlace de la API en la consola, esta hace varias llamadas a la API de API Gateway. Estas llamadas necesitan permisos para la acción `apigateway:*`, como se muestra a continuación:

```
{  
    "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

Puede programar cuándo se debe invocar una función de Lambda. Después de seleccionar una regla de Eventos de CloudWatch existente (o de crear una nueva) AWS Lambda crea un destino nuevo en CloudWatch que invoca la función de Lambda. Para que pueda crearse el destino, debe conceder los siguientes permisos adicionales:

```
{
    "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

Puede hacer que el servicio de Amazon CloudWatch Logs publique eventos e invoque la función de Lambda. Cuando se configura este servicio como un origen de eventos, la consola muestra los grupos de logs de su cuenta. Para que se muestren, debe conceder los permisos `logs:DescribeLogGroups`, como se muestra a continuación:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CloudWatchLogsPerms",
            "Effect": "Allow",
            "Action": [
                "logs>FilterLogEvents",
                "logs>DescribeLogGroups",
                "logs>PutSubscriptionFilter",
                "logs>DescribeSubscriptionFilters",
                "logs>DeleteSubscriptionFilter",
                "logs>GetLogGroupStatistics"
            ],
            "Resource": "arn:aws:logs:region:account-id:*:log-group:*:*"
        }
    ]
}
```

```
        "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

Los permisos adicionales que se muestran son necesarios para administrar los filtros de suscripción.

Amazon Cognito

La consola muestra los grupos de identidades de su cuenta. Después de seleccionar un grupo, puede configurarlo para que tenga Cognito sync trigger como tipo de origen de eventos. Para ello, debe conceder los siguientes permisos adicionales:

```
{
    "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"
            ],
            "Resource": "arn:aws:lambda:region:account-id:*:function:*
```

```
        "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

La consola muestra todas las tablas de su cuenta. Después de seleccionar una tabla, la consola comprueba si existe un flujo de DynamoDB para esa tabla. En caso contrario, crea el flujo. Si desea que el usuario pueda configurar un flujo de DynamoDB como origen de eventos para una función de Lambda debe conceder los siguientes permisos adicionales:

```
{
    "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:PutEventSourceMapping"
            ]
        }
    ]
}
```

```
        "lambda:UpdateEventSourceMapping"
    ],
    "Resource": "*"
}
]
```

Important

Para que una función de Lambda lea desde un flujo de DynamoDB, el rol de ejecución asociado a la función de Lambda debe tener los permisos correctos. Por lo tanto, el usuario también debe contar con los mismos permisos antes de poder conceder los permisos al rol de ejecución. Para conceder estos permisos, puede asociar la política predefinida `AWSLambdaDynamoDBExecutionRole`, primero al usuario y, a continuación, al rol de ejecución.

Amazon Kinesis Data Streams

La consola muestra todos los flujos de Kinesis de la cuenta. Después de seleccionar un flujo, la consola crea mapeos de orígenes de eventos en AWS Lambda. Para que esto funcione, debe conceder los siguientes permisos adicionales:

```
{
    "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

La consola rellena previamente la lista de buckets de la cuenta de AWS y busca la ubicación de cada bucket. Cuando se configura Amazon S3 como origen de eventos, la consola actualiza la configuración de notificaciones de los buckets. Para que esto funcione, debe conceder los siguientes permisos adicionales:

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

Amazon SNS

La consola muestra todos los temas de Amazon Simple Notification Service (Amazon SNS) de la cuenta. Cuando se selecciona un tema, AWS Lambda suscribe esa función de Lambda a ese tema de Amazon SNS. Para que esto funcione, debe conceder los siguientes permisos adicionales:

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

```
        },
        {
            "Sid": "LambdaListESMappingsPerms",
            "Effect": "Allow",
            "Action": [
                "lambda>ListEventSourceMappings"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS IoT

La consola muestra todas las reglas de AWS IoT. Cuando se selecciona una regla, la consola rellena el resto de la información asociada a esta regla en la interfaz de usuario. Si selecciona una regla, la consola la actualiza con información para que los eventos se envíen a AWS Lambda. También puede crear una regla nueva. Para hacer esto, el usuario debe tener los siguientes permisos adicionales:

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

Uso de políticas basadas en recursos para AWS Lambda (políticas de funciones de Lambda)

Una función de Lambda es uno de los recursos de AWS Lambda. Puede agregar permisos a la política asociada a una función de Lambda. Las políticas de permisos asociadas a las funciones de Lambda se

denominan políticas basadas en recursos (o políticas de funciones de Lambda en Lambda). Las políticas de funciones de Lambda se utilizan para administrar los permisos de invocación de funciones de Lambda (consulte [Invoke \(p. 423\)](#)).

Important

Antes de crear políticas basadas en recursos, recomendamos que consulte los temas de introducción en los que se explican los conceptos básicos y las opciones disponibles para administrar el acceso a los recursos de AWS Lambda. Para obtener más información, consulte [Información general sobre la administración de permisos de acceso a los recursos de AWS Lambda \(p. 338\)](#).

Las políticas de funciones de Lambda se utilizan principalmente cuando se está configurando un origen de eventos en AWS Lambda para conceder a un servicio o a un origen de eventos permisos para invocar una función de Lambda (consulte [Invoke \(p. 423\)](#)). Este no es el caso cuando un origen de eventos (como, por ejemplo, Amazon DynamoDB o Kinesis) utiliza el modelo de extracción, en el que los permisos se gestionan en el rol de ejecución de la función de Lambda. Para obtener más información, consulte [Mapeo de orígenes de eventos \(p. 134\)](#).

Las políticas de funciones de Lambda también facilitan la concesión de permisos entre cuentas para invocar funciones de Lambda. Supongamos que desea conceder permisos entre cuentas (por ejemplo, permisos a Amazon S3) para invocar una función de Lambda. En lugar de crear un rol de IAM para conceder permisos entre cuentas, puede agregar los permisos necesarios en una política de función de Lambda.

Note

Si la aplicación personalizada y la función de Lambda que invoca pertenecen a la misma cuenta de AWS, no es necesario conceder permisos explícitos utilizando la política asociada a la función de Lambda.

AWS Lambda ofrece las siguientes operaciones de la API para administrar una política de permisos asociada a una función de Lambda:

- [AddPermission \(p. 372\)](#)
- [GetPolicy \(p. 420\)](#)
- [RemovePermission \(p. 452\)](#)

Note

La consola de AWS Lambda es la forma más sencilla de administrar orígenes de eventos y sus permisos en una política de función de Lambda. Si la consola del servicio de AWS para el origen de eventos permite la configuración de mapeos de orígenes de eventos, puede utilizarla. A medida que se configuran orígenes de eventos nuevos o se modifican los existentes, la consola modifica automáticamente la política de permisos asociada a la función de Lambda.

Para ver la política de funciones utilizando la consola, elija la pestaña Triggers en la página de detalles de la función y, a continuación, elija View function policy. La consola no permite modificar directamente los permisos de una política de función. Debe utilizar la AWS CLI o los SDK de AWS. A continuación se muestran ejemplos realizados con la AWS CLI para las operaciones de la API mencionadas anteriormente en este tema:

Ejemplos

- [Ejemplo 1: Permitir que Amazon S3 invoque una función de Lambda \(p. 357\)](#)
- [Ejemplo 2: Permitir que Amazon API Gateway invoque una función de Lambda \(p. 358\)](#)

- Ejemplo 3: Permitir que una aplicación creada por el usuario en otra cuenta de AWS invoque una función de Lambda (escenario con varias cuentas) (p. 358)
- Ejemplo 4: Recuperar una política de función de Lambda (p. 359)
- Ejemplo 5: Eliminar permisos de una política de función de Lambda (p. 359)
- Ejemplo 6: Trabajar con el control de versiones, los alias y los permisos de las funciones de Lambda (p. 359)

Ejemplo 1: Permitir que Amazon S3 invoque una función de Lambda

Para conceder a Amazon S3 permiso para invocar una función de Lambda, debe configurar los permisos como se indica a continuación:

- Especifique `s3.amazonaws.com` como valor de `principal`.
- Especifique `lambda:InvokeFunction` en el parámetro `action` para conceder permisos para esta acción.

Para asegurarse de que el evento se genera en un bucket específico que pertenece a una cuenta de AWS concreta, especifique también lo siguiente:

- Especifique el ARN del bucket como valor de `source-arn` para tener en cuenta únicamente los eventos procedentes de un bucket específico.
- Especifique el ID de la cuenta de AWS propietaria del bucket, para asegurarse de que el bucket pertenece a la cuenta.

En el siguiente ejemplo de comando de la AWS CLI se añade un permiso a la política de la función de Lambda `helloworld` que concede a Amazon S3 permisos para invocar la función.

```
aws lambda add-permission \
--region us-west-2 \
--function-name helloworld \
--statement-id 1 \
--principal s3.amazonaws.com \
--action lambda:InvokeFunction \
--source-arn arn:aws:s3:::examplebucket \
--source-account 111111111111 \
--profile adminuser
```

En el ejemplo se presupone que el permiso lo añade el usuario `adminuser` (que tiene permisos completos). Por lo tanto, el parámetro `--profile` especifica el perfil `adminuser`.

Como respuesta, AWS Lambda devuelve el siguiente código JSON. El valor de `Statement` es una versión de cadena JSON de la instrucción que se añade a la política de la función de Lambda:

```
{
  "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    \"Effect\":\"Allow\", \"Principal\":{\"Service\":\"s3.amazonaws.com\"},\n    \"Sid\":\"1\"}"}
```

}

Para obtener información acerca del modelo de inserción, consulte [Mapeo de orígenes de eventos \(p. 134\)](#).

Ejemplo 2: Permitir que Amazon API Gateway invoque una función de Lambda

Para conceder permisos a Amazon API Gateway para que pueda invocar una función de Lambda, haga lo siguiente:

- Especifique `apigateway.amazonaws.com` como valor de `principal`.
- Especifique `lambda:InvokeFunction` como la acción para la que concede permisos.
- Especifique el ARN del punto de enlace de API Gateway como valor de `source-arn`.

En el siguiente ejemplo de comando de la AWS CLI se añade un permiso a la política de la función de Lambda `helloworld` que concede a API Gateway permisos para invocar la función.

```
aws lambda add-permission \
--region us-west-2 \
--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
```

Como respuesta, AWS Lambda devuelve el siguiente código JSON. El valor de `Statement` es una versión de cadena JSON de la instrucción que se añade a la política de la función de Lambda:

```
{
  "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\"}"
}
```

Ejemplo 3: Permitir que una aplicación creada por el usuario en otra cuenta de AWS invoque una función de Lambda (escenario con varias cuentas)

Para conceder permisos a otra cuenta de AWS (es decir, para crear un escenario con varias cuentas), debe especificar el ID de la cuenta de AWS como valor de `principal`, tal como se muestra en el siguiente comando de la AWS CLI:

```
aws lambda add-permission \
--region us-west-2 \
--function-name helloworld \
--statement-id 3 \
--principal 111111111111 \
```

```
--action lambda:InvokeFunction \
--profile adminuser
```

Como respuesta, AWS Lambda devuelve el siguiente código JSON. El valor de `Statement` es una versión de cadena JSON de la instrucción que se añade a la política de la función de Lambda:

```
{
  "Statement": "{\"Action\": [\"lambda:InvokeFunction\"],\n    \"Resource\": \"arn:aws:lambda:us-west-2:account-id:function:helloworld\n\", \n    \"Effect\": \"Allow\", \n    \"Principal\": {\"AWS\": \"account-id\"}, \n    \"Sid\": \"3\"}"
}
```

Ejemplo 4: Recuperar una política de función de Lambda

Para recuperar la política de una función de Lambda, utilice el comando `get-policy`:

```
aws lambda get-policy \
--function-name example \
--profile adminuser
```

Ejemplo 5: Eliminar permisos de una política de función de Lambda

Para eliminar permisos de una política de función de Lambda, utilice el comando `remove-permission`, especificando el nombre de la función y el ID de la instrucción:

```
aws lambda remove-permission \
--function-name example \
--statement-id 1 \
--profile adminuser
```

Ejemplo 6: Trabajar con el control de versiones, los alias y los permisos de las funciones de Lambda

Para obtener más información sobre las políticas de permisos para las versiones y los alias de las funciones de Lambda, consulte [Control de versiones, alias y políticas de recursos \(p. 94\)](#).

Permisos de la API de Lambda: información sobre acciones, recursos y condiciones

A la hora de configurar el [Control de acceso \(p. 338\)](#) y escribir una política de permisos que pueda asociar a una identidad de IAM (políticas basadas en identidad), puede utilizar la siguiente tabla como referencia. La lista incluye cada operación de la API de AWS Lambda, las acciones correspondientes para cuya realización puede conceder permisos, el recurso de AWS para el que puede conceder los

permisos y las claves de condición para las acciones de la API especificadas. Las acciones se especifican en el campo **Action** de la política, el valor del recurso en el campo **Resource** de la política y la clave de condición en el campo **Condition keys** de la política.

Para especificar una acción, use el prefijo `lambda:` seguido del nombre de operación de API (por ejemplo, `lambda:CreateFunction`).

Note

Los permisos de la API `Invoke` de AWS Lambda de la siguiente tabla también se pueden conceder a través de políticas basadas en recursos. Para obtener más información, consulte [Uso de políticas basadas en recursos para AWS Lambda \(políticas de funciones de Lambda\) \(p. 355\)](#).

Puede utilizar claves de condición generales de AWS en las políticas de AWS Lambda para expresar condiciones. Para ver una lista completa de claves generales de AWS, consulte [Claves disponibles para las condiciones](#) en la Guía del usuario de IAM.

AWS Lambda también ofrece claves de condición predefinidas para un conjunto limitado de operaciones de la API. Por ejemplo, puede hacer lo siguiente:

- Restringir el acceso basándose en el ARN (nombre de recurso de Amazon) de la función de Lambda para las siguientes operaciones:
 - `CreateEventSourceMapping`
 - `DeleteEventSourceMapping`
 - `UpdateEventSourceMapping`

A continuación se muestra un ejemplo de política que aplica esta condición:

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

- Restringir el mapeo en función de la entidad principal del servicio de AWS a las siguientes operaciones:
 - `AddPermission`
 - `RemovePermission`

A continuación se muestra un ejemplo de política que aplica esta condición:

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

```
    ] }
```

API de AWS Lambda y permisos necesarios para realizar acciones

[AddPermission \(p. 372\)](#)

Acciones: lambda:AddPermission

Recurso: arn:aws:lambda:*region:account-id*:?/*

[CreateEventSourceMapping \(p. 380\)](#)

Acciones: lambda>CreateEventSourceMapping

Recurso: arn:aws:lambda:*region:account-id*:?

[CreateFunction \(p. 385\)](#)

Acciones: lambda>CreateFunction

Recurso: arn:aws:lambda:*region:account-id*:?

[DeleteEventSourceMapping \(p. 395\)](#)

Acciones: lambda>DeleteEventSourceMapping

Recurso: arn:aws:lambda:*region:account-id*:?

[DeleteFunction \(p. 398\)](#)

Acciones: lambda>DeleteFunction

Recurso: arn:aws:lambda:*region:account-id*:?

[GetEventSourceMapping \(p. 408\)](#)

Acciones: lambda:GetEventSourceMapping

Recurso: arn:aws:lambda:*region:account-id*:?

[GetFunction \(p. 411\)](#)

Acciones: lambda:GetFunction

Recurso: arn:aws:lambda:*region:account-id*:?

[GetFunctionConfiguration \(p. 415\)](#)

Acciones: lambda:DescribeMountTargetSecurityGroups

Recurso: arn:aws:lambda:*region:account-id*:?

[GetPolicy \(p. 420\)](#)

Acciones: lambda:DescribeMountTargets

Recurso: arn:aws:lambda:*region:account-id*:?

[Invoke \(p. 423\)](#)

Acciones: lambda:DescribeTags

Recurso: arn:aws:lambda:*region:account-id*:?

[InvokeAsync \(p. 428\)](#)

Acciones: lambda:ModifyMountTargetSecurityGroups

Recurso: arn:aws:lambda:**region:account-id:?**

[ListEventSourceMappings \(p. 434\)](#)

Acciones: lambda>ListEventSourceMappings

Recurso: arn:aws:lambda:**region:account-id:?**

[ListFunctions \(p. 437\)](#)

Acciones: lambda>ListFunctions

Recurso: arn:aws:lambda:**region:account-id:?**

[RemovePermission \(p. 452\)](#)

Acciones: lambda>RemovePermission

Recurso: arn:aws:lambda:**region:account-id:?**

[UpdateEventSourceMapping \(p. 462\)](#)

Acciones: lambda>UpdateEventSourceMapping

Recurso: arn:aws:lambda:**region:account-id:?**

[UpdateFunctionCode \(p. 466\)](#)

Acciones: lambda>UpdateFunctionCode

Recurso: arn:aws:lambda:**region:account-id:?**

[UpdateFunctionConfiguration \(p. 472\)](#)

Acciones: lambda>UpdateFunctionConfiguration

Recurso: arn:aws:lambda:**region:account-id:?**

Plantillas de política

Cuando se crea una función de AWS Lambda en la consola utilizando uno de los proyectos, Lambda permite crear un rol para la función a partir de una lista de plantillas de política de Lambda. Si se selecciona una de estas plantillas, la función de Lambda crea automáticamente el rol con los permisos necesarios asociados a dicha política.

A continuación se enumeran los permisos que se aplican a cada una de las plantillas de política de la lista Policy templates. Los nombres de las plantillas de política se basan en los de los proyectos correspondientes. Lambda rellena automáticamente los marcadores de posición (por ejemplo, región y accountID) con la información pertinente. Para obtener más información sobre cómo crear una función de Lambda utilizando plantillas de política, consulte [Paso 2.1: Creación de una función Hello World de Lambda \(p. 203\)](#).

Las siguientes plantillas se aplican automáticamente dependiendo del tipo de función de Lambda que se esté creando:

Basic: “Permisos básicos de Lambda”

```
{  
  "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: "Permisos de acceso a VPC de Lambda"

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2>CreateNetworkInterface",
                "ec2>DeleteNetworkInterface",
                "ec2>DescribeNetworkInterfaces"
            ],
            "Resource": "*"
        }
    ]
}
```

Kinesis: "Permisos de sondeador de flujos de Kinesis de Lambda"

```
{
    "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: “Permisos de sondeador de flujos de DynamoDB de Lambda”

```
{  
    "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: “Permisos básicos de Lambda Edge”

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs>CreateLogGroup",  
                "logs>CreateLogStream",  
                "logs:PutLogEvents"  
            ],  
            "Resource": [  
                "arn:aws:logs:*:*:*"  
            ]  
        }  
    ]  
}
```

RedrivePolicySNS: “Permisos de cola de mensajes fallidos de SNS”

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sns:Publish"  
            ]  
        }  
    ]  
}
```

```
        ],
        "Resource": "arn:aws:sns:region:accountId:topicName"
    }
}
```

RedrivePolicySQS: “Permisos de cola de mensajes fallidos de SQS”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:SendMessage"
            ],
            "Resource": "arn:aws:sqs:region:accountId:queueName"
        }
    ]
}
```

Las siguientes plantillas se seleccionan según el proyecto que elija. También puede seleccionarlas desde el menú desplegable para añadir permisos adicionales:

CloudFormation: “Permisos de solo lectura de la pila de CloudFormation”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudformation:DescribeStacks"
            ],
            "Resource": "*"
        }
    ]
}
```

AMI: “Permisos de solo lectura de AMI”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeImages"
            ],
            "Resource": "*"
        }
    ]
}
```

KMS: “Permisos de descifrado de KMS”

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "kms:Decrypt"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

S3: “Permisos de solo lectura de objetos de S3”

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Resource": "arn:aws:s3:::/*"  
        }  
    ]  
}
```

Elasticsearch: “Permisos de Elasticsearch”

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "es:ESHttpPost"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

SES: “Permisos de rebote de SES”

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ses:SendBounce"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

```
    ]  
}
```

TestHarness: “Permisos para pruebas TestHarness”

```
{  
    "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: “Permisos para microservicio sencillo”

```
{  
    "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: “Permisos de monitorización de conexiones VPN”

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "cloudwatch:PutMetricData"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:CreateLogStream",  
                "logs:PutLogEvents"  
            ],  
            "Resource": "arn:aws:logs::log-group:/aws/lambda/*"  
        }  
    ]  
}
```

```
        "Effect": "Allow",
        "Action": [
            "ec2:DescribeRegions",
            "ec2:DescribeVpnConnections"
        ],
        "Resource": "*"
    }
]
```

SQS: “Permisos de sondeador SQS”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:DeleteMessage",
                "sns:ReceiveMessage"
            ],
            "Resource": "arn:aws:sqs:*
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:region:accountId:function: functionName*"
        }
    ]
}
```

IoTButton: “Permisos de botón IoT de AWS”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns>ListSubscriptionsByTopic",
                "sns>CreateTopic",
                "sns:SetTopicAttributes",
                "sns:Subscribe",
                "sns:Publish"
            ],
            "Resource": "*"
        }
    ]
}
```

RekognitionNoDataAccess: “Permisos sin datos de Amazon Rekognition”

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:CompareFaces",
                "rekognition:DetectFaces",
                "rekognition:DetectLabels"
            ],
            "Resource": "*"
        }
    ]
}
```

RekognitionReadOnlyAccess: “Permisos de solo lectura de Amazon Rekognition”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition>ListCollections",
                "rekognition>ListFaces",
                "rekognition>SearchFaces",
                "rekognition>SearchFacesByImage"
            ],
            "Resource": "*"
        }
    ]
}
```

RekognitionWriteOnlyAccess: “Permisos de solo escritura de Amazon Rekognition”

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition>CreateCollection",
                "rekognition>IndexFaces"
            ],
            "Resource": "*"
        }
    ]
}
```

Referencia de la API

Esta sección contiene la documentación de referencia de la API de AWS Lambda. Cuando haga llamadas a la API, deberá autenticar la solicitud proporcionando una firma. AWS Lambda es compatible con la versión 4 de la firma. Para obtener más información, consulte [Signature Version 4 Signing Process](#) en la Referencia general de Amazon Web Services.

Para obtener información general acerca del servicio, consulte [¿Qué es AWS Lambda? \(p. 1\)](#). Para obtener más información acerca de cómo funciona el servicio, consulte [Funcionamiento \(p. 188\)](#).

Puede utilizar la AWS CLI para explorar la API de AWS Lambda. Esta guía proporciona varios tutoriales que utilizan la AWS CLI.

Temas

- [Actions \(p. 370\)](#)
- [Data Types \(p. 478\)](#)

Actions

The following actions are supported:

- [AddPermission \(p. 372\)](#)
- [CreateAlias \(p. 376\)](#)
- [CreateEventSourceMapping \(p. 380\)](#)
- [CreateFunction \(p. 385\)](#)
- [DeleteAlias \(p. 393\)](#)
- [DeleteEventSourceMapping \(p. 395\)](#)
- [DeleteFunction \(p. 398\)](#)
- [DeleteFunctionConcurrency \(p. 401\)](#)
- [GetAccountSettings \(p. 403\)](#)
- [GetAlias \(p. 405\)](#)
- [GetEventSourceMapping \(p. 408\)](#)
- [GetFunction \(p. 411\)](#)
- [GetFunctionConfiguration \(p. 415\)](#)
- [GetPolicy \(p. 420\)](#)
- [Invoke \(p. 423\)](#)
- [InvokeAsync \(p. 428\)](#)
- [ListAliases \(p. 431\)](#)
- [ListEventSourceMappings \(p. 434\)](#)
- [ListFunctions \(p. 437\)](#)
- [ListTags \(p. 440\)](#)
- [ListVersionsByFunction \(p. 442\)](#)
- [PublishVersion \(p. 445\)](#)
- [PutFunctionConcurrency \(p. 450\)](#)
- [RemovePermission \(p. 452\)](#)

- [TagResource \(p. 454\)](#)
- [UntagResource \(p. 456\)](#)
- [UpdateAlias \(p. 458\)](#)
- [UpdateEventSourceMapping \(p. 462\)](#)
- [UpdateFunctionCode \(p. 466\)](#)
- [UpdateFunctionConfiguration \(p. 472\)](#)

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 [AWS Lambda: How it Works](#).

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",
  "SourceAccount": "string",
  "SourceArn": "string",
  "StatementId": "string"
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 372)

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. 372)

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. 372)

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. 372)

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. 372)

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

SourceAccount (p. 372)

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 \(p. 372\)](#)

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 \(p. 372\)](#)

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 \(p. 374\)](#)

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

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 \(p. 376\)](#)

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-zA-Z0-9-_]{1,2}-[a-zA-Z0-9-_]+\d{1}:[a-zA-Z0-9-_]{1,12})(function:[a-zA-Z0-9-_]+)(:$LATEST|[a-zA-Z0-9-_]+)`

Request Body

The request accepts the following data in JSON format.

[Description \(p. 376\)](#)

Description of the alias.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

Required: No

[FunctionVersion \(p. 376\)](#)

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 \(p. 376\)](#)

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 \(p. 376\)](#)

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 [???](#).

Type: [AliasRoutingConfiguration \(p. 485\)](#) object

Required: No

Response Syntax

```
HTTP/1.1 201
Content-type: application/json

{
  "AliasArn": "string",
  "Description": "string",
  "FunctionVersion": "string",
  "Name": "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 \(p. 377\)](#)

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 \(p. 377\)](#)

Alias description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 377\)](#)

Function version to which the alias points.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST | [0-9]+)

[Name \(p. 377\)](#)

Alias name.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\\$)([a-zA-Z0-9-_]+)

[RoutingConfig \(p. 377\)](#)

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 [???](#).

Type: [AliasRoutingConfiguration \(p. 485\)](#) 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.

Important

This event source mapping is relevant only in the AWS Lambda pull model, where AWS Lambda invokes the function. For more information, see [AWS Lambda: How it Works](#) in the AWS Lambda Developer Guide.

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 function. 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. 380)

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 \(p. 380\)](#)

Indicates whether AWS Lambda should begin polling the event source. By default, `Enabled` is true.

Type: Boolean

Required: No

[EventSourceArn \(p. 380\)](#)

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 \(p. 380\)](#)

The Lambda function to invoke when AWS Lambda detects an event on the stream.

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). 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 \(p. 380\)](#)

The position in the stream where AWS Lambda should start reading. Valid only for Kinesis streams. For more information, see [ShardIteratorType](#) in the Amazon Kinesis API Reference.

Type: String

Valid Values: `TRIM_HORIZON` | `LATEST` | `AT_TIMESTAMP`

Required: Yes

[StartingPositionTimestamp \(p. 380\)](#)

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 \(p. 382\)](#)

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.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 10000.

[EventSourceArn \(p. 382\)](#)

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 \(p. 382\)](#)

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 \(p. 382\)](#)

The UTC time string indicating the last time the event mapping was updated.

Type: Timestamp

[LastProcessingResult \(p. 382\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 382\)](#)

The state of the event source mapping. It can be `Creating`, `Enabled`, `Disabled`, `Enabling`, `Disabling`, `Updating`, or `Deleting`.

Type: String

[StateTransitionReason \(p. 382\)](#)

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 \(p. 382\)](#)

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 \(p. 385\)](#)

The code for the Lambda function.

Type: [FunctionCode \(p. 493\)](#) object

Required: Yes

[DeadLetterConfig \(p. 385\)](#)

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic.

Type: [DeadLetterConfig \(p. 487\)](#) object

Required: No

[Description \(p. 385\)](#)

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 \(p. 385\)](#)

The parent object that contains your environment's configuration settings.

Type: [Environment \(p. 488\)](#) object

Required: No

[FunctionName \(p. 385\)](#)

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 \(p. 437\)](#) API. Function names are used to specify functions to other AWS Lambda API operations, such as [Invoke \(p. 423\)](#). 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 \(p. 385\)](#)

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 \(p. 385\)](#)

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-.]+:\w*)|()`

Required: No

[MemorySize \(p. 385\)](#)

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 \(p. 385\)](#)

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 \(p. 385\)](#)

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 \(p. 385\)](#)

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".

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 \(p. 385\)](#)

The list of tags (key-value pairs) assigned to the new function.

Type: String to string map

Required: No

[Timeout \(p. 385\)](#)

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 \(p. 385\)](#)

The parent object that contains your function's tracing settings.

Type: [TracingConfig \(p. 499\)](#) object

Required: No

[VpcConfig \(p. 385\)](#)

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 \(p. 501\)](#) 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,
"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 \(p. 388\)](#)

It is the SHA256 hash of your function deployment package.

Type: String

[CodeSize \(p. 388\)](#)

The size, in bytes, of the function .zip file you uploaded.

Type: Long

[DeadLetterConfig \(p. 388\)](#)

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic.

Type: [DeadLetterConfig \(p. 487\)](#) object

[Description \(p. 388\)](#)

The user-provided description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 388\)](#)

The parent object that contains your environment's configuration settings.

Type: [EnvironmentResponse \(p. 490\)](#) object

[FunctionArn \(p. 388\)](#)

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 \(p. 388\)](#)

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 \(p. 388\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 388\)](#)

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 \(p. 388\)](#)

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 \(p. 388\)](#)

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 \(p. 388\)](#)

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.

[Role \(p. 388\)](#)

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 \(p. 388\)](#)

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 \(p. 388\)](#)

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 \(p. 388\)](#)

The parent object that contains your function's tracing settings.

Type: [TracingConfigResponse \(p. 500\)](#) object

[Version \(p. 388\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[VpcConfig \(p. 388\)](#)

VPC configuration associated with your Lambda function.

Type: [VpcConfigResponse \(p. 502\)](#) object

Errors

`CodeStorageExceeded`

You have exceeded your maximum total code size per account. [Limits](#)

HTTP Status Code: 400

`InvalidParameterValue`

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

`ResourceConflict`

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. 393)

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. 393)

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 \(p. 395\)](#)

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 \(p. 395\)](#)

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 \(p. 395\)](#)

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 \(p. 395\)](#)

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 \(p. 395\)](#)

The UTC time string indicating the last time the event mapping was updated.

Type: Timestamp

[LastProcessingResult \(p. 395\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 395\)](#)

The state of the event source mapping. It can be Creating, Enabled, Disabled, Enabling, Disabling, Updating, or Deleting.

Type: String

[StateTransitionReason \(p. 395\)](#)

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 \(p. 395\)](#)

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. 398)

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. 398)

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 [Ejecuciones simultáneas de la función de Lambda \(p. 189\)](#).

Request Syntax

```
DELETE /2017-10-31/functions/FunctionName/concurrency HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 401)

The name of the function you are removing concurrent execution limits from. For more information, see [Ejecuciones simultáneas de la función de Lambda \(p. 189\)](#).

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 \(p. 403\)](#)

Provides limits of code size and concurrency associated with the current account and region.

Type: [AccountLimit \(p. 480\)](#) object

[AccountUsage \(p. 403\)](#)

Provides code size usage and function count associated with the current account and region.

Type: [AccountUsage \(p. 482\)](#) 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. 405)

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-zA-Z0-9-_]{2}-[a-zA-Z0-9-_]+\d{1}:\d{12}:(function:[a-zA-Z0-9-_]+)(:$LATEST|[a-zA-Z0-9-_]+))?`

[Name](#) (p. 405)

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-_]{1,128}`

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "AliasArn": "string",
  "Description": "string",
  "FunctionVersion": "string",
  "Name": "string",
  "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 \(p. 405\)](#)

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 \(p. 405\)](#)

Alias description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 405\)](#)

Function version to which the alias points.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[Name \(p. 405\)](#)

Alias name.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: `(?!^[\0-9]+$)([a-zA-Z0-9-_]+)`

[RoutingConfig \(p. 405\)](#)

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 [???](#).

Type: [AliasRoutingConfiguration \(p. 485\)](#) 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](#)

GetEventSourceMapping

Returns configuration information for the specified event source mapping (see [CreateEventSourceMapping \(p. 380\)](#)).

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 \(p. 408\)](#)

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 \(p. 408\)](#)

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 \(p. 408\)](#)

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 \(p. 408\)](#)

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 \(p. 408\)](#)

The UTC time string indicating the last time the event mapping was updated.

Type: Timestamp

[LastProcessingResult \(p. 408\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 408\)](#)

The state of the event source mapping. It can be Creating, Enabled, Disabled, Enabling, Disabling, Updating, or Deleting.

Type: String

[StateTransitionReason \(p. 408\)](#)

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 \(p. 408\)](#)

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 \(p. 385\)](#) 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. 411)

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. 411)

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,
    "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 \(p. 411\)](#)

The object for the Lambda function location.

Type: [FunctionCodeLocation \(p. 494\)](#) object

Concurrency (p. 411)

The concurrent execution limit set for this function. For more information, see [Ejecuciones simultáneas de la función de Lambda \(p. 189\)](#).

Type: [Concurrency \(p. 486\)](#) object

Configuration (p. 411)

A complex type that describes function metadata.

Type: [FunctionConfiguration \(p. 495\)](#) object

Tags (p. 411)

Returns the list of tags associated with the function.

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 \(p. 385\)](#).

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. 415)

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-_`.]+)(:(`\$LATEST`|[`a-zA-Z0-9-_`+]))?

`Qualifier` (p. 415)

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,  
    "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. 415)

It is the SHA256 hash of your function deployment package.

Type: String

[CodeSize](#) (p. 415)

The size, in bytes, of the function .zip file you uploaded.

Type: Long

[DeadLetterConfig](#) (p. 415)

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic.

Type: [DeadLetterConfig](#) (p. 487) object

[Description](#) (p. 415)

The user-provided description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 415\)](#)

The parent object that contains your environment's configuration settings.

Type: [EnvironmentResponse \(p. 490\)](#) object

[FunctionArn \(p. 415\)](#)

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 \(p. 415\)](#)

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 \(p. 415\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 415\)](#)

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 \(p. 415\)](#)

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 \(p. 415\)](#)

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 \(p. 415\)](#)

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.

[Role \(p. 415\)](#)

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 \(p. 415\)](#)

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 \(p. 415\)](#)

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 \(p. 415\)](#)

The parent object that contains your function's tracing settings.

Type: [TracingConfigResponse \(p. 500\)](#) object

[Version \(p. 415\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[VpcConfig \(p. 415\)](#)

VPC configuration associated with your Lambda function.

Type: [VpcConfigResponse \(p. 502\)](#) object

Errors

InvalidOperationException

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. 420)

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. 420)

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"
```

}

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 \(p. 420\)](#)

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

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. 423)

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. 423)

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 \(p. 423\)](#)

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 \(p. 423\)](#)

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 \(p. 423\)](#)

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 \(p. 423\)](#)

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 \(p. 424\)](#)

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 \(p. 424\)](#)

The function version that has been executed. This value is returned only if the invocation type is RequestResponse. For more information, see [???](#).

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (\\$LATEST | [0-9]+)

[FunctionError \(p. 424\)](#)

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 \(p. 424\)](#)

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 \(p. 424\)](#)

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 \(p. 423\)](#)).

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 \(p. 428\)](#)

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 \(p. 428\)](#)

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 \(p. 428\)](#)

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. 431)

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. 431)

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. 431)

Optional string. An opaque pagination token returned from a previous `ListAliases` operation. If present, indicates where to continue the listing.

[MaxItems](#) (p. 431)

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",  
            "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 \(p. 431\)](#)

A list of aliases.

Type: Array of [AliasConfiguration \(p. 483\)](#) objects

[NextMarker \(p. 431\)](#)

A string, present if there are more aliases.

Type: String

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](#)

ListEventSourceMappings

Returns a list of event source mappings you created using the [CreateEventSourceMapping](#) (see [CreateEventSourceMapping \(p. 380\)](#)).

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 \(p. 434\)](#)

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 \(p. 434\)](#)

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 \(p. 434\)](#)

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 \(p. 434\)](#)

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 \(p. 435\)](#)

An array of `EventSourceMappingConfiguration` objects.

Type: Array of [EventSourceMappingConfiguration \(p. 491\)](#) objects

[NextMarker \(p. 435\)](#)

A string, present if there are more event source mappings.

Type: String

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](#)

ListFunctions

Returns a list of your Lambda functions. For each function, the response includes the function configuration information. You must use [GetFunction \(p. 411\)](#) 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 \(p. 437\)](#)

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 \(p. 437\)](#)

Optional string. An opaque pagination token returned from a previous `ListFunctions` operation. If present, indicates where to continue the listing.

[MasterRegion \(p. 437\)](#)

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 \(p. 437\)](#)

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,
            "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 \(p. 438\)](#)

A list of Lambda functions.

Type: Array of [FunctionConfiguration \(p. 495\)](#) objects

[NextMarker \(p. 438\)](#)

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).

Request Syntax

```
GET /2017-03-31/tags/ARN HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[Resource \(p. 440\)](#)

The ARN (Amazon Resource Name) of the function.

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 \(p. 440\)](#)

The list of tags assigned to the function.

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. 442)

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. 442)

Optional string. An opaque pagination token returned from a previous `ListVersionsByFunction` operation. If present, indicates where to continue the listing.

[MaxItems](#) (p. 442)

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,
"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. 442)

A string, present if there are more function versions.

Type: String

[Versions](#) (p. 442)

A list of Lambda function versions.

Type: Array of [FunctionConfiguration](#) (p. 495) 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"
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName](#) (p. 445)

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-_]+)(:(\\$\text{LATEST}|[a-zA-Z0-9-_]+))?

Request Body

The request accepts the following data in JSON format.

[CodeSha256](#) (p. 445)

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. 466) to verify the hash value that will be returned before publishing your new version.

Type: String

Required: No

[Description](#) (p. 445)

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

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,
    "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 \(p. 446\)](#)

It is the SHA256 hash of your function deployment package.

Type: String

[CodeSize \(p. 446\)](#)

The size, in bytes, of the function .zip file you uploaded.

Type: Long

[DeadLetterConfig \(p. 446\)](#)

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic.

Type: [DeadLetterConfig \(p. 487\)](#) object

[Description \(p. 446\)](#)

The user-provided description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 446\)](#)

The parent object that contains your environment's configuration settings.

Type: [EnvironmentResponse \(p. 490\)](#) object

[FunctionArn \(p. 446\)](#)

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 \(p. 446\)](#)

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 \(p. 446\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 446\)](#)

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 \(p. 446\)](#)

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 \(p. 446\)](#)

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 \(p. 446\)](#)

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.

[Role \(p. 446\)](#)

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 \(p. 446\)](#)

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 \(p. 446\)](#)

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 \(p. 446\)](#)

The parent object that contains your function's tracing settings.

Type: [TracingConfigResponse \(p. 500\)](#) object

[Version \(p. 446\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[VpcConfig \(p. 446\)](#)

VPC configuration associated with your Lambda function.

Type: [VpcConfigResponse \(p. 502\)](#) object

Errors

CodeStorageExceededException

HTTP Status Code: 400
You have exceeded your maximum total code size per account. [Limits](#)

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](#)

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 [Ejecuciones simultáneas de la función de Lambda \(p. 189\)](#).

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 \(p. 450\)](#)

The name of the function you are setting concurrent execution limits on. For more information, see [Ejecuciones simultáneas de la función de Lambda \(p. 189\)](#).

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 \(p. 450\)](#)

The concurrent execution limit reserved for this function. For more information, see [Ejecuciones simultáneas de la función de Lambda \(p. 189\)](#).

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 \(p. 450\)](#)

The number of concurrent executions reserved for this function. For more information, see [Ejecuciones simultáneas de la función de Lambda \(p. 189\)](#).

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 HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 452)

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-z]{2}-[a-z]+-\d{1}:)?(\d{12}:)?(function:)?(([a-zA-Z0-9-_]+)(:(\\$\#LATEST|[a-zA-Z0-9-_]+))?

Qualifier (p. 452)

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\$-_]+)

StatementId (p. 452)

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

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.

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. 454)

The ARN (Amazon Resource Name) of the Lambda function.

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. 454)

The list of tags (key-value pairs) you are assigning to the Lambda function.

Type: String to string map

Required: Yes

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. 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).

Request Syntax

```
DELETE /2017-03-31/tags/ARN?tagKeys=TagKeys HTTP/1.1
```

URI Request Parameters

The request requires the following URI parameters.

[Resource \(p. 456\)](#)

The ARN (Amazon Resource Name) of the function.

Pattern: `arn:aws:lambda:[a-z]{2}-[a-z]+\d{1}:\d{12}:function:[a-zA-Z0-9-_]+(:(\$LATEST|[a-zA-Z0-9-_]+))?`

[TagKeys \(p. 456\)](#)

The list of tag keys to be deleted from the function.

Request Body

The request does not have a request body.

Response Syntax

```
HTTP/1.1 204
```

Response Elements

If the action is successful, the service sends back an HTTP 204 response with an empty HTTP body.

Errors

InvalidParameterValueException

One of the parameters in the request is invalid. 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",
  "RoutingConfig": {
    "AdditionalVersionWeights": {
      "string" : number
    }
  }
}
```

URI Request Parameters

The request requires the following URI parameters.

[FunctionName \(p. 458\)](#)

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 \(p. 458\)](#)

The alias name.

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (?![0-9]+\\$)([a-zA-Z0-9-_]+)

Request Body

The request accepts the following data in JSON format.

[Description \(p. 458\)](#)

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 \(p. 458\)](#)

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

[RoutingConfig \(p. 458\)](#)

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 [???](#).

Type: [AliasRoutingConfiguration \(p. 485\)](#) object

Required: No

Response Syntax

```
HTTP/1.1 200
Content-type: application/json

{
  "AliasArn": "string",
  "Description": "string",
  "FunctionVersion": "string",
  "Name": "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 \(p. 459\)](#)

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 \(p. 459\)](#)

Alias description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[FunctionVersion \(p. 459\)](#)

Function version to which the alias points.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: (`\$LATEST` | [0-9]+)

[Name \(p. 459\)](#)

Alias name.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 128.

Pattern: (? !^ [0-9]+ \$) ([a-zA-Z0-9-_]+)

[RoutingConfig \(p. 459\)](#)

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 [???](#).

Type: [AliasRoutingConfiguration \(p. 485\)](#) 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](#)

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. 462)

The event source mapping identifier.

Request Body

The request accepts the following data in JSON format.

[BatchSize](#) (p. 462)

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. 462)

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 \(p. 462\)](#)

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 \(p. 463\)](#)

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 \(p. 463\)](#)

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 \(p. 463\)](#)

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 \(p. 463\)](#)

The UTC time string indicating the last time the event mapping was updated.

Type: Timestamp

[LastProcessingResult \(p. 463\)](#)

The result of the last AWS Lambda invocation of your Lambda function.

Type: String

[State \(p. 463\)](#)

The state of the event source mapping. It can be `Creating`, `Enabled`, `Disabled`, `Enabling`, `Disabling`, `Updating`, or `Deleting`.

Type: String

[StateTransitionReason \(p. 463\)](#)

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 \(p. 463\)](#)

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,
  "S3Bucket": "string",
  "S3Key": "string",
  "S3ObjectVersion": "string",
  "ZipFile": blob
}
```

URI Request Parameters

The request requires the following URI parameters.

FunctionName (p. 466)

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. 466)

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 \(p. 466\)](#)

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

[S3Bucket \(p. 466\)](#)

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 \(p. 466\)](#)

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 \(p. 466\)](#)

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 \(p. 466\)](#)

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

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,
    "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 \(p. 467\)](#)

It is the SHA256 hash of your function deployment package.

Type: String

[CodeSize \(p. 467\)](#)

The size, in bytes, of the function .zip file you uploaded.

Type: Long

[DeadLetterConfig \(p. 467\)](#)

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic.

Type: [DeadLetterConfig \(p. 487\)](#) object

[Description \(p. 467\)](#)

The user-provided description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 467\)](#)

The parent object that contains your environment's configuration settings.

Type: [EnvironmentResponse \(p. 490\)](#) object

[FunctionArn \(p. 467\)](#)

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 \(p. 467\)](#)

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 \(p. 467\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 467\)](#)

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 \(p. 467\)](#)

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 \(p. 467\)](#)

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 \(p. 467\)](#)

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.

[Role \(p. 467\)](#)

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 \(p. 467\)](#)

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 \(p. 467\)](#)

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 \(p. 467\)](#)

The parent object that contains your function's tracing settings.

Type: [TracingConfigResponse \(p. 500\)](#) object

[Version \(p. 467\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[VpcConfig \(p. 467\)](#)

VPC configuration associated with your Lambda function.

Type: [VpcConfigResponse \(p. 502\)](#) 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

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,
  "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. 472)

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 \(p. 472\)](#)

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic.

Type: [DeadLetterConfig \(p. 487\)](#) object

Required: No

[Description \(p. 472\)](#)

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 \(p. 472\)](#)

The parent object that contains your environment's configuration settings.

Type: [Environment \(p. 488\)](#) object

Required: No

[Handler \(p. 472\)](#)

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 \(p. 472\)](#)

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 \(p. 472\)](#)

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

[Role \(p. 472\)](#)

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 \(p. 472\)](#)

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 Python runtime v3.6, set the value to "python3.6".

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 \(p. 472\)](#)

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 \(p. 472\)](#)

The parent object that contains your function's tracing settings.

Type: [TracingConfig \(p. 499\)](#) object

Required: No

[VpcConfig \(p. 472\)](#)

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 \(p. 501\)](#) 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,
    "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 \(p. 475\)](#)

It is the SHA256 hash of your function deployment package.

Type: String

[CodeSize \(p. 475\)](#)

The size, in bytes, of the function .zip file you uploaded.

Type: Long

[DeadLetterConfig \(p. 475\)](#)

The parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic.

Type: [DeadLetterConfig \(p. 487\)](#) object

[Description \(p. 475\)](#)

The user-provided description.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 256.

[Environment \(p. 475\)](#)

The parent object that contains your environment's configuration settings.

Type: [EnvironmentResponse \(p. 490\)](#) object

[FunctionArn \(p. 475\)](#)

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 \(p. 475\)](#)

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 \(p. 475\)](#)

The function Lambda calls to begin executing your function.

Type: String

Length Constraints: Maximum length of 128.

Pattern: `[^\s]+`

[KMSKeyArn \(p. 475\)](#)

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 \(p. 475\)](#)

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 \(p. 475\)](#)

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 \(p. 475\)](#)

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.

[Role \(p. 475\)](#)

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 \(p. 475\)](#)

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 \(p. 475\)](#)

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 \(p. 475\)](#)

The parent object that contains your function's tracing settings.

Type: [TracingConfigResponse \(p. 500\)](#) object

[Version \(p. 475\)](#)

The version of the Lambda function.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 1024.

Pattern: `(\$LATEST|[0-9]+)`

[VpcConfig \(p. 475\)](#)

VPC configuration associated with your Lambda function.

Type: [VpcConfigResponse \(p. 502\)](#) 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](#)

Data Types

The following data types are supported:

- [AccountLimit \(p. 480\)](#)
- [AccountUsage \(p. 482\)](#)
- [AliasConfiguration \(p. 483\)](#)
- [AliasRoutingConfiguration \(p. 485\)](#)

- [Concurrency \(p. 486\)](#)
- [DeadLetterConfig \(p. 487\)](#)
- [Environment \(p. 488\)](#)
- [EnvironmentError \(p. 489\)](#)
- [EnvironmentResponse \(p. 490\)](#)
- [EventSourceMappingConfiguration \(p. 491\)](#)
- [FunctionCode \(p. 493\)](#)
- [FunctionCodeLocation \(p. 494\)](#)
- [FunctionConfiguration \(p. 495\)](#)
- [TracingConfig \(p. 499\)](#)
- [TracingConfigResponse \(p. 500\)](#)
- [VpcConfig \(p. 501\)](#)
- [VpcConfigResponse \(p. 502\)](#)

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 [Ejecuciones simultáneas de la función de Lambda \(p. 189\)](#).

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

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 [???](#).

Type: [AliasRoutingConfiguration \(p. 485\)](#) 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 [???](#).

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 [???](#).

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 [Ejecuciones simultáneas de la función de Lambda \(p. 189\)](#).

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 parent object that contains the target ARN (Amazon Resource Name) of an Amazon SQS queue or Amazon SNS topic.

Contents

TargetArn

The Amazon Resource Name (ARN) of an Amazon SQS queue or Amazon SNS topic you specify as your Dead Letter Queue (DLQ).

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 \(p. 489\)](#) 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.

Type: [DeadLetterConfig \(p. 487\)](#) 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 \(p. 490\)](#) 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-z]{2}-[a-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-z]{2}-[a-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

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 \(p. 500\)](#) 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 \(p. 502\)](#) 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 you 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](#)

Historial de revisión

En la siguiente tabla se describen los cambios importantes que se han realizado en la Guía para desarrolladores de AWS Lambda.

Fechas pertinentes para este historial:

- Versión del producto actual: 2015-03-31
- Última actualización de la documentación: 11 de agosto de 2017

| Cambio | Descripción | Fecha |
|---|--|------------------------|
| Presentación de SAM Local | AWS Lambda presenta SAM Local, una herramienta de la AWS CLI que proporciona un entorno para desarrollar, probar y analizar localmente aplicaciones sin servidor antes de cargarlas en el tiempo de ejecución de Lambda. Para obtener más información, consulte Prueba local de aplicaciones sin servidor mediante SAM Local (en versión beta pública) (p. 169). | 11 de agosto de 2017 |
| Región Canadá (Central) | AWS Lambda ya está disponible en la Región Canadá (Central). Para obtener más información acerca de las regiones y los puntos de enlace de Lambda, consulte Regiones y puntos de enlace en la AWS General Reference. | 22 de junio de 2017 |
| Región América del Sur (São Paulo) | AWS Lambda ya está disponible en la Región América del Sur (São Paulo). Para obtener más información acerca de las regiones y los puntos de enlace de Lambda, consulte Regiones y puntos de enlace en la AWS General Reference. | 6 de junio de 2017 |
| Compatibilidad de AWS Lambda con AWS X-Ray | Lambda introduce la compatibilidad con X-Ray, que permite detectar, analizar y optimizar problemas de desempeño con las aplicaciones de Lambda. Para obtener más información, consulte Solución de problemas de aplicaciones basadas en Lambda (p. 176). | 19 de abril de 2017 |
| Región Asia Pacífico (Mumbai) | AWS Lambda ya está disponible en la Región Asia Pacífico (Mumbai). Para obtener más información acerca de las regiones y los puntos de enlace de Lambda, consulte Regiones y puntos de enlace en la AWS General Reference. | 28 de marzo de 2017 |
| AWS Lambda ahora es compatible con el tiempo de ejecución v6.10 de Node.js | AWS Lambda incorpora compatibilidad con el tiempo de ejecución v6.10 de Node.js. Para obtener más información, consulte Modelo de programación (Node.js) (p. 9). | 22 de marzo de 2017 |
| Región UE (Londres) | AWS Lambda ya está disponible en la Región UE (Londres). Para obtener más información acerca de las regiones y los puntos de enlace de Lambda, consulte Regiones y puntos de enlace en la AWS General Reference. | 1 de febrero de 2017 |
| Compatibilidad de AWS Lambda con el tiempo de ejecución de .NET, Lambda@Edge (vista previa), las colas de | AWS Lambda presenta las características siguientes: <ul style="list-style-type: none"> • AWS Lambda incorpora compatibilidad con C#. Para obtener más información, consulte Modelo de programación para crear funciones de Lambda en C# (p. 53). | 3 de diciembre de 2016 |

| Cambio | Descripción | Fecha |
|--|--|-------------------------|
| mensajes fallidos y la implementación automatizada de las aplicaciones sin servidor | <ul style="list-style-type: none"> Lambda@Edge (vista previa) le permite ejecutar funciones de Lambda en las ubicaciones de borde de AWS en respuesta a eventos de CloudFront. Para obtener más información, consulte AWS Lambda@Edge (p. 317). Se ha añadido un tutorial para la automatización de la implementación de aplicaciones sin servidor mediante AWS CodePipeline, AWS CodeBuild y AWS CloudFormation. Para obtener más información, consulte Automatización de la implementación de aplicaciones basadas en Lambda (p. 164). Se ha actualizado Solución de problemas y monitorización de funciones de AWS Lambda con Amazon CloudWatch (p. 120) para incluir una sección sobre las Colas de mensajes fallidos (p. 128), que se pueden configurar para recuperar información sobre las invocaciones asíncronas de funciones de Lambda que generan errores. | |
| AWS Lambda añade Amazon Lex como origen de eventos compatible. | Con Lambda y Amazon Lex, puede crear rápidamente bots de chat para diversos servicios, como Slack y Facebook. Para obtener más información, consulte Amazon Lex (p. 144) . | 30 de noviembre de 2016 |
| EE.UU. Oeste (Norte de California) | AWS Lambda ya está disponible en la EE.UU. Oeste (Norte de California). Para obtener más información acerca de las regiones y los puntos de enlace de Lambda, consulte Regiones y puntos de enlace en la AWS General Reference. | 21 de noviembre de 2016 |
| Se presenta AWS Serverless Application Model, que permite crear e implementar aplicaciones basadas en Lambda y utilizar variables de entorno para los ajustes de configuración de las funciones de Lambda. | <p>AWS Lambda presenta las siguientes características en esta versión:</p> <ul style="list-style-type: none"> AWS Serverless Application Model: ahora puede utilizar AWS SAM para definir la sintaxis para expresar recursos dentro de una aplicación sin servidor. Para implementar la aplicación, solo tiene que especificar los recursos que necesita que formen parte de ella, junto con sus políticas de permisos asociadas en un archivo de plantilla de AWS CloudFormation (escrito en JSON o YAML), empaquetar sus artefactos de implementación e implementar la plantilla. Para obtener más información, consulte Implementación de aplicaciones basadas en Lambda (p. 156). Variables de entorno: puede utilizar variables de entorno para especificar las opciones de configuración de una función de Lambda fuera del código de la función. Para obtener más información, consulte Variables de entorno (p. 97). | 18 de noviembre de 2016 |
| Se ha añadido un tutorial en la Introducción (p. 199) para la creación de un punto de enlace de Amazon API Gateway mediante la consola de Lambda. | El tutorial explica cómo integrar fácilmente una función de Lambda con una API a través de las características nuevas presentadas en Configuración de la integración del proxy de API Gateway . Para obtener más información, consulte Paso 3: Creación de un microservicio sencillo utilizando Lambda y API Gateway (p. 211) . | 29 de agosto de 2016 |

| Cambio | Descripción | Fecha |
|---|---|------------------------|
| Región Asia Pacífico (Seúl) | AWS Lambda ya está disponible en la Región Asia Pacífico (Seúl). Para obtener más información acerca de las regiones y los puntos de enlace de Lambda, consulte Regiones y puntos de enlace en la AWS General Reference. | 29 de agosto de 2016 |
| Región Asia Pacífico (Sídney) | Lambda ya está disponible en la Región Asia Pacífico (Sídney). Para obtener más información acerca de las regiones y los puntos de enlace de Lambda, consulte Regiones y puntos de enlace en la AWS General Reference. | 23 de junio de 2016 |
| Actualizaciones de la consola de Lambda | La consola de Lambda se ha actualizado para simplificar el proceso de creación de roles. Para obtener más información, consulte Paso 2.1: Creación de una función Hello World de Lambda (p. 203) . | 23 de junio de 2016 |
| AWS Lambda ahora es compatible con el tiempo de ejecución v4.3 de Node.js | AWS Lambda incorpora compatibilidad con el tiempo de ejecución v4.3 de Node.js. Para obtener más información, consulte Modelo de programación (Node.js) (p. 9) . | 07 de abril de 2016 |
| Región UE (Fráncfort) | Lambda ya está disponible en la región UE (Fráncfort). Para obtener más información acerca de las regiones y los puntos de enlace de Lambda, consulte Regiones y puntos de enlace en la AWS General Reference. | 14 de marzo de 2016 |
| Compatibilidad con VPC | Ahora puede configurar una función de Lambda para obtener acceso a los recursos de una VPC. Para obtener más información, consulte Configuración de una función de Lambda para acceder recursos en una Amazon VPC (p. 108) . Para ver tutoriales de ejemplo, consulte Tutoriales: Configuración de una función de Lambda para acceder los recursos de una Amazon VPC (p. 111) . | 11 de febrero de 2016 |
| Reorganización del contenido | El contenido reorganizado ahora incluye lo siguiente: <ul style="list-style-type: none"> • Introducción (p. 199): contiene un ejercicio basado en la consola en el que se crea una función de Lambda denominada Hello World. Puede explorar las características de la consola de AWS Lambda, incluidos los proyectos, que le permiten crear funciones de Lambda con tan solo unos clics. • Casos de uso (p. 214): proporciona ejemplos de cómo utilizar AWS Lambda con otros servicios de AWS o con aplicaciones personalizadas como orígenes de eventos, invocar a través de HTTPS y configurar AWS Lambda para invocar la función de Lambda a intervalos programados. • Modelo de programación (p. 8): explica los conceptos principales del modelo de programación y describe detalles específicos de los lenguajes. Independientemente del lenguaje que elija, existe un patrón común para escribir el código de una función de Lambda. • Creación de un paquete de implementación (p. 64): explica cómo crear paquetes de implementación para el código de una función de Lambda escrito en los lenguajes compatibles con AWS Lambda (Python, Java y Node.js). | 9 de diciembre de 2015 |

| Cambio | Descripción | Fecha |
|--|--|------------------------|
| Se ha actualizado el tiempo de ejecución de AWS Lambda. | <p>El tiempo de ejecución de AWS Lambda se ha actualizado con las siguientes versiones de los SDK y del kernel de Linux en esta versión:</p> <ul style="list-style-type: none"> • AWS SDK para JavaScript: 2.2.12 • SDK de Boto: 1.2.1 • Versión del kernel de Linux: 3.14.48-33.39.amzn1.x86_6. <p>Para obtener más información, consulte Entorno de ejecución de Lambda y bibliotecas disponibles (p. 195).</p> | 4 de noviembre de 2015 |
| Compatibilidad con el control de versiones, Python para desarrollar código para las funciones de Lambda, eventos programados y aumento del tiempo de ejecución | <p>AWS Lambda presenta las siguientes características en esta versión:</p> <ul style="list-style-type: none"> • Python: ahora puede desarrollar el código de sus funciones de Lambda con Python. Para obtener más información, consulte Modelo de programación (p. 8). • Control de versiones: puede mantener una o varias versiones de una función de Lambda. El control de versiones permite controlar qué versión de una función de Lambda se ejecuta en los distintos entornos (por ejemplo, desarrollo, pruebas, o producción). Para obtener más información, consulte Control de versiones y alias de las funciones de AWS Lambda (p. 80). • Eventos programados: también puede configurar AWS Lambda para que invoque el código de forma periódica y programada utilizando la consola de AWS Lambda. Puede especificar una frecuencia fija (un número de horas, días o semanas) o una expresión cron. Para ver un ejemplo, consulte Uso de AWS Lambda con eventos programados (p. 302). • Aumento del tiempo de ejecución: a partir de ahora, puede configurar las funciones de Lambda para que se ejecuten durante un máximo de cinco minutos, lo que permite utilizar funciones con tiempos de ejecución más largos como, por ejemplo, trabajos de adquisición y procesamiento de grandes volúmenes de datos. | 08 de octubre de 2015 |
| Dos nuevos tutoriales | <p>Se han añadido los siguientes tutoriales. Ambos utilizan una función de Lambda en Java.</p> <p>Tutorial: Uso de AWS Lambda con Amazon DynamoDB (p. 244)</p> <p>Uso de AWS Lambda como backend de aplicaciones móviles (origen de eventos personalizados: Android) (p. 289)</p> | 27 de agosto de 2015 |

| Cambio | Descripción | Fecha |
|--|---|---------------------|
| Compatibilidad con DynamoDB Streams | DynamoDB Streams ya está disponible con carácter general y se puede utilizar en todas las regiones en las que está disponible. Puede activar DynamoDB Streams para una tabla y utilizar una función de Lambda como un disparador para la tabla. Los disparadores son acciones personalizadas que se llevan a cabo en respuesta a las actualizaciones realizadas en la tabla de DynamoDB. Para ver un tutorial de ejemplo, consulte Tutorial: Uso de AWS Lambda con Amazon DynamoDB (p. 244) . | 14 de julio de 2015 |
| AWS Lambda ahora permite invocar funciones de Lambda con clientes compatibles con REST. | Hasta ahora, para invocar una función de Lambda desde aplicaciones web, móviles o IoT, se necesitaban los SDK de AWS (por ejemplo, AWS SDK para Java, AWS SDK para Android o AWS SDK para iOS). Ahora, AWS Lambda permite invocar una función de Lambda con clientes compatibles con REST a través de una API personalizada que puede crear con Amazon API Gateway. Puede enviar solicitudes a la URL del punto de enlace de una función de Lambda. Puede configurar la seguridad en el punto de enlace para permitir el acceso libre, utilizar AWS Identity and Access Management (IAM) para autorizar el acceso, o utilizar claves de API para medir el acceso de otros usuarios a sus funciones de Lambda. Para ver un ejemplo de ejercicio de introducción, consulte Uso de AWS Lambda con Amazon API Gateway (bajo demanda a través de HTTPS) (p. 275) . Para obtener más información sobre Amazon API Gateway, consulte https://aws.amazon.com/api-gateway/ . | 09 de julio de 2015 |
| La consola de AWS Lambda ahora ofrece proyectos para crear fácilmente funciones de Lambda y probarlas. | La consola de AWS Lambda proporciona un conjunto de proyectos. Cada proyecto ofrece una configuración de origen de eventos de muestra y código de muestra para la función de Lambda que puede utilizar para crear fácilmente aplicaciones basadas en Lambda. Ahora, todos los ejercicios de introducción de AWS Lambda utilizan proyectos. Para obtener más información, consulte Introducción (p. 199) . | En esta versión |
| AWS Lambda ahora permite utilizar Java para crear funciones de Lambda. | A partir de ahora, puede escribir el código de Lambda en Java. Para obtener más información, consulte Modelo de programación (p. 8) . | 15 de junio de 2015 |
| AWS Lambda ahora permite especificar un objeto de Amazon S3 como archivo .zip de la función al crear o actualizar una función de Lambda. | Puede cargar el paquete de implementación (archivo.zip) de una función de Lambda en un bucket de Amazon S3 de la misma región en la que desea crear la función de Lambda. A continuación, puede especificar el nombre del bucket y un nombre de la clave de objeto al crear o actualizar una función de Lambda. | 28 de mayo de 2015 |

| Cambio | Descripción | Fecha |
|--|---|-------------------------|
| AWS Lambda ahora está disponible con carácter general con soporte para backends para móviles | <p>AWS Lambda ya se encuentra disponible con carácter general para su uso con fines de producción. La versión también presenta características nuevas que facilitan la creación de backends para móviles, tablets e Internet de las cosas (IoT) con AWS Lambda que se escalan automáticamente sin necesidad de aprovisionar ni administrar infraestructura. AWS Lambda ahora es compatible con eventos en tiempo real (síncronos) y asíncronos. También incluye una configuración y administración de orígenes de eventos más sencilla. El modelo de permisos y el modelo de programación se han simplificado mediante la introducción de políticas de recursos para las funciones de Lambda.</p> <p>La documentación se ha actualizado en consecuencia. Para obtener información, consulte los siguientes temas:</p> <p>Funcionamiento (p. 188)</p> <p>Introducción (p. 199)</p> <p>AWS Lambda</p> | 9 de abril de 2015 |
| Versión de prueba | Versión de vista previa de la Guía para desarrolladores de AWS Lambda. | 13 de noviembre de 2014 |

AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the AWS General Reference.