



**Autor: Herber Antonio Batres Santé**



## Índice

Capítulo I.....	3
Introducción .....	3
1.1 ¿Qué es AWS Lambda? .....	3
1.2 ¿Qué es una API Serverless? .....	3
1.3 Beneficios del Uso de AWS Lambda para APIs.....	3
1.4 Objetivo del Manual .....	4
Capitulo II .....	6
Conceptos Básicos de AWS Lambda .....	6
2.1 Cómo Funciona AWS Lambda .....	6
2.1.1 Características Clave .....	6
2.1.2 Arquitectura.....	7
2.2 Principales Componentes de AWS Lambda .....	7
2.3 Casos de Uso Comunes .....	8
Capitulo III .....	9
Primeros Pasos con AWS Lambda .....	9
3.1 Creación de una Cuenta de AWS.....	9
3.2 Configuración Inicial de AWS Lambda.....	11
3.2.1 Creación de una Cuenta de AWS .....	11
3.2.2 Acceso a la Consola de AWS Lambda .....	11
3.3 Configuración de AWS CLI y SDK.....	14
3.3.1 Instalación y Configuración de AWS CLI.....	14
3.3.2 Instalación y Configuración de AWS SDK .....	16
3.3.3 Uso de AWS CLI y SDKs para Automatización y Desarrollo .....	19
Capitulo IV .....	20
Desarrollo de una API Básica con AWS Lambda .....	20
4.1 Creación de una Función Lambda Sencilla .....	20
4.2 Configuración del AWS API Gateway .....	21
4.3 Despliegue de la API .....	22
4.4 Prueba de la API en el Navegador .....	23
Capitulo V .....	24

Implementación de Funciones Lambda.....	24
5.1 Estructura de una Función Lambda .....	24
5.1.1 Componentes Clave.....	24
5.2 Programación de Funciones con Node.js y Python .....	25
5.2.1 Ejemplo en Node.js .....	25
5.2.2 Ejemplo en Python .....	26
5.3 Gestión de Dependencias.....	27
5.3.1 Node.js.....	27
5.3.2 Python.....	27
Capítulo VI.....	29
Manejo de Eventos y Triggers.....	29
6.1 Configuración de Eventos en API Gateway .....	29
6.1.1 Creación de una API.....	29
6.1.2 Definición de Recursos y Métodos .....	29
6.1.3 Configuración de Eventos de Solicitud: .....	30
6.1.4 Transformación de Solicitudes y Respuestas .....	31
6.1.5 Configuración de CORS (Cross-Origin Resource Sharing).....	31
6.1.6 Beneficios de Usar Eventos en API Gateway .....	32
6.2 Uso de Triggers para Automatización .....	32
6.2.1 Uso de Triggers en AWS Lambda .....	32
6.2.2 Trigger en DynamoDB .....	33
6.2.3 Integración con CloudWatch Events .....	34
Bibliografía.....	35

## Capítulo I

### Introducción

#### 1.1 ¿Qué es AWS Lambda?

AWS Lambda es un servicio de computación que permite ejecutar funciones de código en lenguajes como Node.js, Python, Ruby, Java, Go y .NET. Este servicio elimina la necesidad de aprovisionar, gestionar o escalar servidores manualmente. Las funciones de Lambda se ejecutan en contenedores aislados y administrados, que se activan en respuesta a eventos específicos, los cuales pueden ser desencadenados por varios orígenes programáticos que AWS ofrece (Services, 2021).

#### 1.2 ¿Qué es una API Serverless?

API Serverless es un término que describe un enfoque en la creación y gestión de APIs utilizando arquitecturas serverless. Estas APIs permiten a los desarrolladores concentrarse en escribir el código y la lógica de la aplicación sin preocuparse por la infraestructura del servidor. En un entorno serverless, las funciones de backend se ejecutan en respuesta a eventos y son administradas automáticamente por un proveedor de servicios en la nube, lo que elimina la necesidad de gestionar servidores físicos o virtuales. Los beneficios de las APIs serverless incluyen una mayor escalabilidad, reducción de costos de operación y una simplificación en el desarrollo de aplicaciones, ya que los desarrolladores pueden centrarse en la lógica de negocio en lugar de en la infraestructura (Amazon Web Services, 2023).

#### 1.3 Beneficios del Uso de AWS Lambda para APIs

AWS Lambda es un servicio de computación sin servidor que ofrece una serie de beneficios al utilizarlo para desarrollar y gestionar APIs. Aquí están algunos de los principales beneficios:

- **Escalabilidad Automática:** AWS Lambda se encarga automáticamente de escalar las funciones según la demanda. No es necesario preocuparse por el aprovisionamiento de servidores adicionales, ya que Lambda puede manejar cualquier cantidad de solicitudes concurrentes al mismo tiempo.

- **Costo Efectivo:** Con AWS Lambda, solo pagas por el tiempo de cómputo que utilizas. No existen costos asociados al tiempo de inactividad, ya que las funciones solo se ejecutan en respuesta a eventos y solicitudes.
- **Gestión Simplificada:** Al utilizar Lambda, se elimina la necesidad de gestionar servidores o infraestructura, permitiendo a los desarrolladores centrarse exclusivamente en la lógica de negocio y el desarrollo de la API.
- **Integración con Otros Servicios AWS:** AWS Lambda se integra fácilmente con otros servicios de AWS, como API Gateway, DynamoDB, y S3, facilitando la construcción de aplicaciones completas y escalables en la nube.
- **Alta Disponibilidad:** Las funciones de Lambda se ejecutan en múltiples regiones y zonas de disponibilidad de AWS, garantizando una alta disponibilidad y redundancia para las APIs.
- **Tiempo de Respuesta Rápido:** Al estar diseñado para ejecutarse en respuesta a eventos, AWS Lambda permite tiempos de respuesta rápidos, lo que es ideal para aplicaciones en tiempo real.
- **Seguridad Mejorada:** Lambda permite implementar medidas de seguridad robustas mediante el uso de AWS Identity and Access Management (IAM) para controlar el acceso a las funciones y servicios de AWS asociados.
- **Desarrollo Ágil:** La capacidad de desplegar rápidamente funciones y actualizar la lógica de la API permite un ciclo de desarrollo ágil, facilitando iteraciones rápidas y la implementación continua.
- **Soporte Multilenguaje:** AWS Lambda admite varios lenguajes de programación, incluidos Node.js, Python, Java, Go, Ruby, y .NET, proporcionando flexibilidad en el desarrollo.
- **Monitorización y Análisis:** Con AWS CloudWatch, puedes monitorizar el rendimiento de tus funciones Lambda y analizar métricas clave para mejorar el funcionamiento de tus APIs.

## 1.4 Objetivo del Manual

El objetivo de este manual es proporcionar una guía sencilla y práctica para el desarrollo y despliegue de APIs serverless utilizando AWS Lambda. Este documento está diseñado para facilitar el entendimiento y la aplicación de tecnologías serverless, destacando los beneficios de

la automatización y la escalabilidad que ofrecen. A través de ejemplos prácticos y explicaciones claras, los desarrolladores podrán:

- **Comprender los conceptos básicos de AWS Lambda** y cómo se integra en la arquitectura serverless.
- **Aprender a configurar y desarrollar APIs serverless** desde cero, utilizando herramientas y servicios de AWS.
- **Implementar prácticas de seguridad y gestión** para asegurar el correcto funcionamiento y protección de las APIs.
- **Optimizar y mantener APIs de manera eficiente**, mejorando costos y rendimiento en aplicaciones serverless.

Este manual está dirigido a desarrolladores, ingenieros de software y entusiastas de la tecnología que buscan explorar y adoptar el enfoque serverless para construir aplicaciones modernas y escalables. El objetivo final es equipar a los lectores con las habilidades necesarias para crear APIs eficientes en entornos serverless, aprovechando la flexibilidad y el poder de AWS Lambda.

## Capítulo II

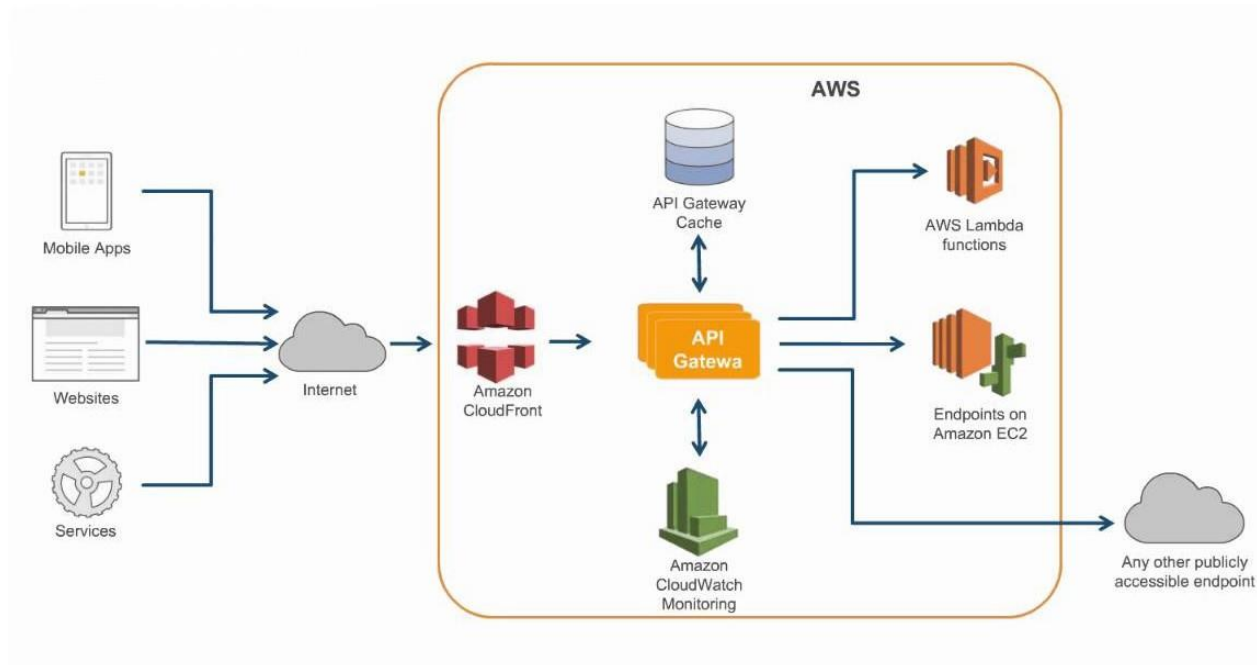
### Conceptos Básicos de AWS Lambda

#### 2.1 Cómo Funciona AWS Lambda

AWS Lambda es un servicio de computación serverless proporcionado por Amazon Web Services (AWS) que permite a los desarrolladores ejecutar código en respuesta a eventos sin gestionar servidores. La arquitectura serverless de AWS Lambda facilita la ejecución de código en un entorno gestionado y escalable (Amazon Web Services, 2023).

Cuando se produce un evento, como la carga de un archivo en un bucket de S3 o una actualización en una base de datos DynamoDB, AWS Lambda invoca automáticamente la función asociada a ese evento. La infraestructura subyacente, incluida la asignación de recursos y el escalado, es gestionada completamente por AWS, permitiendo a los desarrolladores centrarse solo en el código.

*Ilustración 1. Arquitectura Serverless*



Nota: (Amazon Web Services, 2023).

##### 2.1.1 Características Clave

- Ejecución bajo demanda: El código se ejecuta solo cuando se produce un evento.



- Escalabilidad automática: AWS Lambda escala automáticamente para manejar el volumen de eventos.
- Modelo de pago por uso: Solo pagas por el tiempo de ejecución que consumes.

### 2.1.2 Arquitectura

La arquitectura de AWS Lambda está diseñada para simplificar el desarrollo y la gestión de aplicaciones serverless:

- **Eventos:** AWS Lambda responde a eventos generados por diversos servicios, como Amazon S3, DynamoDB, SNS, y API Gateway. Estos eventos disparan la ejecución de funciones Lambda.
- **Funciones Lambda:** Son el código que se ejecuta en respuesta a eventos. Cada función está compuesta por el código, el handler (punto de entrada), y configuraciones como la memoria y el tiempo de ejecución.
- **Entorno de Ejecución:** AWS Lambda proporciona un entorno gestionado donde el código se ejecuta. Este entorno es reutilizable y se escala automáticamente en función de la demanda.
- **Capas:** Permiten la inclusión de librerías y dependencias compartidas entre varias funciones, facilitando la modularidad y gestión del código.
- **Integraciones:** Lambda se integra con otros servicios de AWS y gestiona la infraestructura subyacente para una ejecución eficiente del código.

## 2.2 Principales Componentes de AWS Lambda

- **Función Lambda:** El núcleo del servicio, que contiene el código a ejecutar. Cada función tiene un handler que especifica el punto de entrada y puede ser configurada con memoria y tiempo de ejecución.
- **Eventos:** Son las fuentes que disparan la ejecución de funciones Lambda, tales como cambios en buckets de S3 o mensajes en colas de SQS.
- **Entorno de Ejecución:** El entorno donde se ejecuta el código. AWS Lambda proporciona un runtime específico para cada lenguaje de programación soportado.

- **Capas:** Componentes reutilizables que permiten incluir librerías o dependencias compartidas entre funciones Lambda.
- **IAM Roles:** Permiten la gestión de permisos para que las funciones Lambda interactúen con otros servicios de AWS de forma segura.
- **Monitoreo y Logging:** AWS Lambda se integra con Amazon CloudWatch para proporcionar logs y métricas sobre la ejecución de funciones.

### 2.3 Casos de Uso Comunes

AWS Lambda se utiliza en una variedad de casos de uso comunes, tales como:

- **Procesamiento de Datos en Tiempo Real:** Lambda puede procesar datos de streams en tiempo real, como los datos de Amazon Kinesis o DynamoDB Streams.
- **Automatización de Infraestructura:** Permite la automatización de tareas de infraestructura, como la gestión de recursos en respuesta a eventos específicos.
- **Backend para Aplicaciones Móviles y Web:** Lambda se usa para construir APIs RESTful mediante Amazon API Gateway, proporcionando un backend sin servidor para aplicaciones móviles y web.
- **Automatización de Respuestas a Eventos:** Permite ejecutar funciones en respuesta a eventos de servicios de AWS, como la carga de archivos en S3 o la actualización de registros en DynamoDB.
- **Notificaciones y Mensajería:** Lambda puede procesar mensajes de SNS o SQS para enviar notificaciones o realizar tareas de procesamiento de mensajes.

## Capítulo III

### Primeros Pasos con AWS Lambda

#### 3.1 Creación de una Cuenta de AWS

Cuando desees crear una cuenta en AWS, sigue estos pasos esenciales:

1. **Accede a la Página Principal de AWS:** Comienza abriendo tu navegador web y dirigiéndote al sitio principal de [Amazon Web Services \(AWS\)](https://aws.amazon.com/).
2. **Inicia el Proceso de Registro:** Haz clic en la opción para crear una nueva cuenta de AWS.

**Nota:** Si ya has iniciado sesión previamente en AWS, selecciona la opción de "Iniciar sesión". Si no ves la opción de crear una nueva cuenta, haz clic en "Iniciar sesión en otra cuenta" y posteriormente selecciona "Crear una nueva Cuenta de AWS".

3. **Rellena la Información de la Cuenta:** Ingresa los datos solicitados en el formulario de registro y luego selecciona la opción para verificar tu dirección de correo electrónico. Un código de verificación será enviado a la dirección proporcionada.
4. **Verificación del Correo Electrónico:** Introduce el código de verificación que has recibido y presiona "Verificar" para continuar con el proceso.
5. **Establece una Contraseña Segura:** Crea una contraseña sólida para el usuario root, confírmala y haz clic en "Continuar". AWS requiere que la contraseña cumpla con las siguientes condiciones:
  - Debe tener entre 8 y 128 caracteres.
  - Debe incluir al menos tres de los siguientes tipos de caracteres: letras mayúsculas, letras minúsculas, números y caracteres especiales (!@#\$%^&\*()<>[]{}\_+~=).
  - No debe coincidir con el nombre de la cuenta de AWS ni con la dirección de correo electrónico proporcionada.

6. **Selecciona el Tipo de Cuenta:** Elige entre una cuenta "Empresarial" o "Personal".  
Ambos tipos de cuentas ofrecen las mismas funcionalidades, pero el proceso de registro requiere diferentes tipos de información.
7. **Proporciona la Información de Contacto:** Introduce la información de contacto que se solicita, asegurándote de revisar las recomendaciones anteriores respecto a la dirección de correo electrónico y el número de teléfono.
8. **Acepta el Contrato de Cliente de AWS:** Lee detenidamente los términos del Acuerdo de Cliente de AWS. Una vez que los comprendas, marca la casilla para aceptar los términos.
9. **Completa el Registro Inicial:** Haz clic en el botón "Continuar". En este momento, recibirás un correo electrónico de confirmación que indicará que tu cuenta está lista para ser utilizada. Podrás iniciar sesión utilizando la dirección de correo electrónico y la contraseña que configuraste, aunque todavía no tendrás acceso a los servicios de AWS hasta que la activación esté completamente finalizada.
10. **Introduce los Datos de Pago:** Proporciona los detalles de tu método de pago. Si necesitas usar una dirección diferente para la facturación, selecciona "Usar una dirección nueva".
11. **Verifica el Método de Pago:** Selecciona la opción para verificar y continuar.
12. **Proporciona Información de Contacto Telefónico:** Elige tu código de país o región de la lista, luego ingresa un número de teléfono de contacto y el código CAPTCHA antes de enviarlo.
13. **Completa la Verificación Telefónica:** Cuando el sistema automatizado te llame, introduce el PIN que recibiste y sigue las instrucciones para enviarlo.
14. **Selecciona un Plan de Soporte de AWS:** Escoge el plan de soporte que mejor se adapte a tus necesidades. Puedes consultar la descripción de los planes en la sección "Comparar planes de soporte de AWS".
15. **Finaliza el Registro:** Haz clic en "Completar registro". Verás una página de confirmación indicando que tu cuenta se está activando.

16. **Espera la Confirmación de Activación:** Revisa tu bandeja de entrada y tu carpeta de correo no deseado para encontrar el correo de confirmación que indique que tu cuenta ha sido activada. Este proceso puede tardar unos minutos, pero en ocasiones puede tomar hasta 24 horas.
17. **Acceso a los Servicios de AWS:** Una vez que recibas el correo de activación, tendrás acceso completo a todos los servicios de AWS.

### 3.2 Configuración Inicial de AWS Lambda

AWS Lambda es un servicio de computación sin servidor que te permite ejecutar código sin necesidad de aprovisionar o gestionar servidores. Este servicio facilita la ejecución de aplicaciones y servicios de forma ágil y eficiente. A continuación, se detallan los pasos para la configuración inicial de AWS Lambda, adaptados y parafraseados para facilitar su comprensión y aplicación (Amazon Web Services, 2023).

#### 3.2.1 Creación de una Cuenta de AWS

Antes de comenzar con AWS Lambda, asegúrate de tener una cuenta de AWS. Si aún no la tienes, sigue los pasos indicados en la sección anterior para crear una nueva cuenta. Una vez que tu cuenta esté activa, sigue los siguientes pasos para configurar AWS Lambda:

#### 3.2.2 Acceso a la Consola de AWS Lambda

##### 1. Inicia Sesión en la Consola de AWS:

Accede a la [Consola de Administración de AWS](#) ingresando con tu dirección de correo electrónico y contraseña asociada a tu cuenta de AWS.

##### 2. Navega al Servicio AWS Lambda:

Una vez dentro de la consola, localiza el servicio **AWS Lambda** utilizando la barra de búsqueda o en la sección de **Servicios** bajo la categoría de **Compute**.

##### 3. Creación de una Nueva Función Lambda:

Haz clic en **Create function** para iniciar el proceso de configuración de tu primera función Lambda.

##### 4. Selecciona el Método de Creación:

Puedes elegir entre varios métodos de creación, como **Author from scratch** (Crear desde cero), **Use a blueprint** (Usar una plantilla), o **Browse serverless app repository** (Explorar el repositorio de aplicaciones serverless). Para este ejemplo, seleccionaremos **Author from scratch**.

#### 5. Define los Detalles de la Función:

- **Nombre de la Función:** Proporciona un nombre único para tu función Lambda. Por ejemplo, "MiPrimeraFuncionLambda".
- **Tiempo de Ejecución (Runtime):** Selecciona el entorno de ejecución para tu función. AWS Lambda admite diversos lenguajes de programación como Node.js, Python, Java, Go, y más. Elige el lenguaje que mejor se adapte a tu aplicación.
- **Permisos de Ejecución:** Configura el rol de ejecución (IAM Role) que otorga los permisos necesarios para ejecutar la función. Puedes crear un nuevo rol o utilizar uno existente. Si eliges crear un nuevo rol, AWS creará uno con permisos básicos para Lambda.

#### 6. Configuración Avanzada (Opcional):

- **VPC:** Si necesitas que tu función acceda a recursos dentro de una VPC, selecciona la opción de VPC y configura los subnets y grupos de seguridad necesarios.
- **Configuración de Tiempo de Ejecución y Memoria:** Ajusta los parámetros de memoria y el tiempo máximo de ejecución para tu función según las necesidades de tu aplicación.

#### 7. Crear la Función:

Haz clic en **Create function** para finalizar el proceso de configuración. Ahora tu función Lambda está lista para ser utilizada.

#### 8. Edición del Código de la Función:

- Dentro de la consola de AWS Lambda, accede al editor de código integrado para escribir o modificar el código de tu función. AWS Lambda permite la edición de código directamente desde la consola, facilitando el proceso de desarrollo.

- **Carga de Código desde un Archivo ZIP o un Bucket de S3:** Si prefieres trabajar en un entorno local, puedes empaquetar tu código en un archivo ZIP y cargarlo directamente o utilizar un bucket de Amazon S3.

#### 9. Configuración de Variables de Entorno:

Define variables de entorno que tu función Lambda pueda utilizar durante la ejecución. Estas variables son útiles para almacenar configuraciones y credenciales que no deseas incluir directamente en el código fuente.

#### 10. Prueba de la Función Lambda:

Utiliza la opción **Test** dentro de la consola para crear eventos de prueba y verificar que tu función Lambda se ejecuta correctamente. Puedes definir eventos de prueba personalizados que simulen situaciones reales para asegurar que tu función funciona como se espera.

#### 11. Agregar Desencadenadores:

AWS Lambda puede integrarse con otros servicios de AWS para ejecutar tu función automáticamente en respuesta a ciertos eventos. Agrega desencadenadores como Amazon S3, DynamoDB, o API Gateway, dependiendo de las necesidades de tu aplicación.

#### 12. Configuración de Permisos Adicionales:

Revisa y ajusta los permisos de IAM asociados a tu función Lambda para asegurarte de que tenga acceso a los recursos necesarios y cumpla con las políticas de seguridad de tu organización.

#### 13. Despliegue de la Función:

Una vez configurada y probada, despliega tu función Lambda para que esté disponible en el entorno de producción. Considera utilizar herramientas de automatización como AWS CloudFormation o AWS CDK para gestionar el ciclo de vida de tu función.

#### 14. Monitoreo y Registro:

Implementa monitoreo y registro utilizando servicios como Amazon CloudWatch para rastrear el rendimiento de tu función Lambda, identificar posibles problemas y optimizar su ejecución.

## 15. Finalización

Con estos pasos, has configurado exitosamente una función AWS Lambda y puedes comenzar a integrar este servicio en tus aplicaciones. La flexibilidad de Lambda permite ejecutar funciones a demanda, optimizando costos y recursos de infraestructura.

### 3.3 Configuración de AWS CLI y SDK

La AWS Command Line Interface (CLI) y los AWS SDKs son herramientas esenciales que permiten interactuar con los servicios de Amazon Web Services de forma programática. La CLI facilita la gestión de recursos a través de comandos de terminal, mientras que los SDKs proporcionan interfaces de programación para trabajar con diferentes lenguajes. A continuación, se detallan los pasos para instalar y configurar estas herramientas.

#### 3.3.1 Instalación y Configuración de AWS CLI

La AWS CLI es una herramienta que permite ejecutar comandos para los servicios de AWS desde tu terminal, lo que facilita la automatización de tareas. A continuación, se describen los pasos para instalar y configurar la CLI:

##### 3.3.1.1 Instalación de AWS CLI

###### 1. Descarga del Instalador:

Accede al sitio web oficial de [AWS CLI](#) y descarga el instalador correspondiente a tu sistema operativo. AWS CLI es compatible con Windows, macOS, y Linux.

###### 2. Instalación en Windows:

Ejecuta el archivo .msi descargado y sigue las instrucciones del asistente de instalación.

###### 3. Instalación en macOS:

Abre la terminal y utiliza el gestor de paquetes Homebrew para instalar AWS CLI con el siguiente comando:

```
brew install awscli
```

###### 4. Instalación en Linux:



En Linux, puedes usar el paquete de administración apt o yum según tu distribución. Por ejemplo, para Ubuntu:

```
sudo apt update
```

```
sudo apt install awscli
```

## 5. Verificar la Instalación:

Una vez instalada, verifica que AWS CLI esté correctamente instalada ejecutando el siguiente comando en la terminal:

```
aws --version
```

La salida debe mostrar la versión instalada de AWS CLI.

### 3.3.1.2 Configuración de AWS CLI

#### 1. Configuración de Credenciales:

Para configurar la AWS CLI, abre tu terminal y ejecuta el comando:

```
aws configure
```

- Proporciona la **Access Key ID** y **Secret Access Key** de tu cuenta de AWS cuando se te solicite. Puedes encontrar estas credenciales en la [Consola de IAM](#).
- Ingresa la región predeterminada (por ejemplo, us-east-1) y el formato de salida (json, text, o table).

#### 2. Verificar la Configuración:

Para verificar que la configuración sea correcta, ejecuta un comando simple como listar los buckets de S3:

```
aws s3 ls
```

Si todo está configurado correctamente, verás una lista de tus buckets de S3.

### 3.3.2 Instalación y Configuración de AWS SDK

AWS SDKs son librerías que permiten interactuar con los servicios de AWS desde aplicaciones escritas en varios lenguajes de programación, como Python, JavaScript, Java, Ruby, y más. A continuación, se detalla cómo instalar y configurar AWS SDK para diferentes lenguajes:

#### 3.3.2.1 AWS SDK para Python (Boto3)

##### 1. Instalación de Boto3:

Abre tu terminal y utiliza el siguiente comando para instalar Boto3, el SDK de AWS para Python, utilizando pip:

```
pip install boto3
```

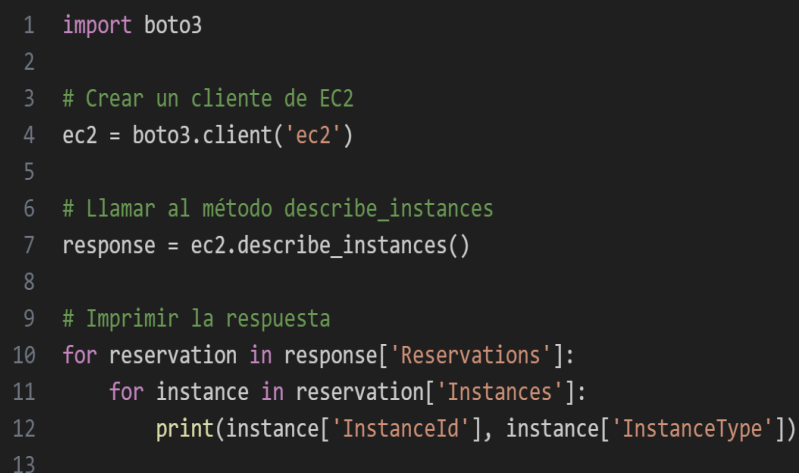
##### 2. Configuración de Credenciales:

Las credenciales para Boto3 se pueden configurar de manera similar a AWS CLI, utilizando el archivo de configuración en ~/.aws/credentials.

##### 3. Ejemplo de Uso de Boto3:

Aquí tienes un ejemplo simple para listar tus instancias de EC2 utilizando Boto3:

*Ilustración 2. Código de Instancia EC2*



```
1 import boto3
2
3 # Crear un cliente de EC2
4 ec2 = boto3.client('ec2')
5
6 # Llamar al método describe_instances
7 response = ec2.describe_instances()
8
9 # Imprimir la respuesta
10 for reservation in response['Reservations']:
11     for instance in reservation['Instances']:
12         print(instance['InstanceId'], instance['InstanceType'])
13
```

Nota: Fuente Propia.

### 3.3.2.2 AWS SDK para JavaScript (Node.js)

#### 1. Instalación de AWS SDK para JavaScript:

Si utilizas Node.js, instala el SDK de AWS utilizando npm:

```
npm install aws-sdk
```

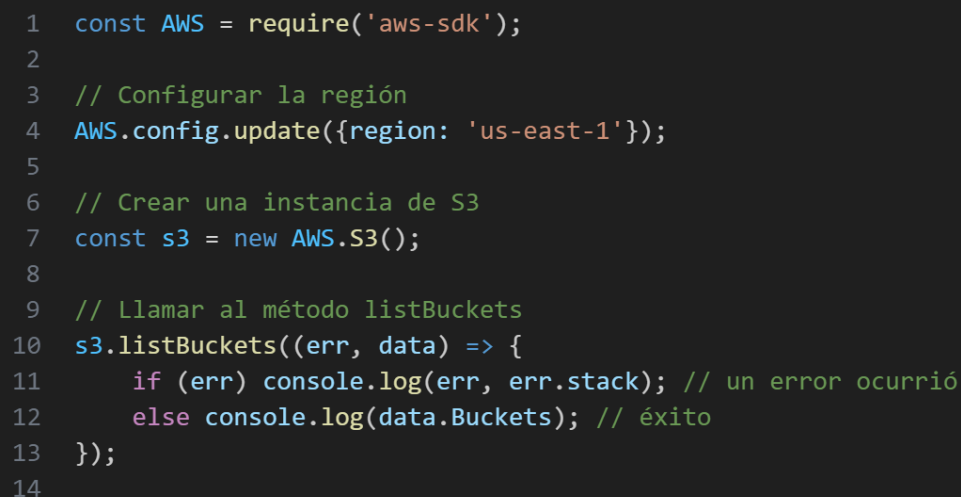
#### 2. Configuración de Credenciales:

Similar a otros SDKs, configura las credenciales utilizando el archivo ~/.aws/credentials o mediante variables de entorno.

#### 3. Ejemplo de Uso del SDK para JavaScript:

Aquí tienes un ejemplo de cómo listar tus buckets de S3 utilizando Node.js:

*Ilustración 3. Buckets de S3*



```
1  const AWS = require('aws-sdk');
2
3  // Configurar la región
4  AWS.config.update({region: 'us-east-1'});
5
6  // Crear una instancia de S3
7  const s3 = new AWS.S3();
8
9  // Llamar al método listBuckets
10 s3.listBuckets((err, data) => {
11     if (err) console.log(err, err.stack); // un error ocurrió
12     else console.log(data.Buckets); // éxito
13 });
14
```

Nota: Fuente Propia

### 3.3.2.3 AWS SDK para Java

#### 1. Instalación de AWS SDK para Java:

Si estás trabajando en un proyecto de Java, puedes agregar el SDK de AWS a tu proyecto mediante Maven o Gradle. Aquí te muestro cómo hacerlo con Maven:

Ilustración 4. AWS SDK para Java

```
1 import software.amazon.awssdk.regions.Region;
2 import software.amazon.awssdk.services.ec2.Ec2Client;
3 import software.amazon.awssdk.services.ec2.model.DescribeInstancesRequest;
4 import software.amazon.awssdk.services.ec2.model.DescribeInstancesResponse;
5 import software.amazon.awssdk.services.ec2.model.Instance;
6 import software.amazon.awssdk.services.ec2.model.Reservation;
7 import software.amazon.awssdk.services.ec2.model.Ec2Exception;
8
9 public class ListInstances {
10
11     public static void main(String[] args) {
12         // Crear un cliente de EC2
13         Ec2Client ec2 = Ec2Client.builder()
14             .region(Region.US_EAST_1) // Especifica la región
15             .build();
16
17         try {
18             // Crear una solicitud para describir las instancias
19             DescribeInstancesRequest request = DescribeInstancesRequest.builder().build();
20
21             // Obtener la respuesta con las instancias
22             DescribeInstancesResponse response = ec2.describeInstances(request);
23
24             // Iterar sobre las reservas y las instancias
25             for (Reservation reservation : response.reservations()) {
26                 for (Instance instance : reservation.instances()) {
27                     // Imprimir la información de la instancia
28                     System.out.printf(
29                         "Instance ID: %s, Instance Type: %s, State: %s\n",
30                         instance.getInstanceId(),
31                         instance.getInstanceType(),
32                         instance.getState().getName()
33                     );
34                 }
35             }
36
37         } catch (Ec2Exception e) {
38             System.err.println("Error al describir las instancias: " + e.awsErrorDetails().errorMessage());
39             System.exit(1);
40         } finally {
41             ec2.close();
42         }
43     }
44 }
45
```

Nota: Fuente Propia

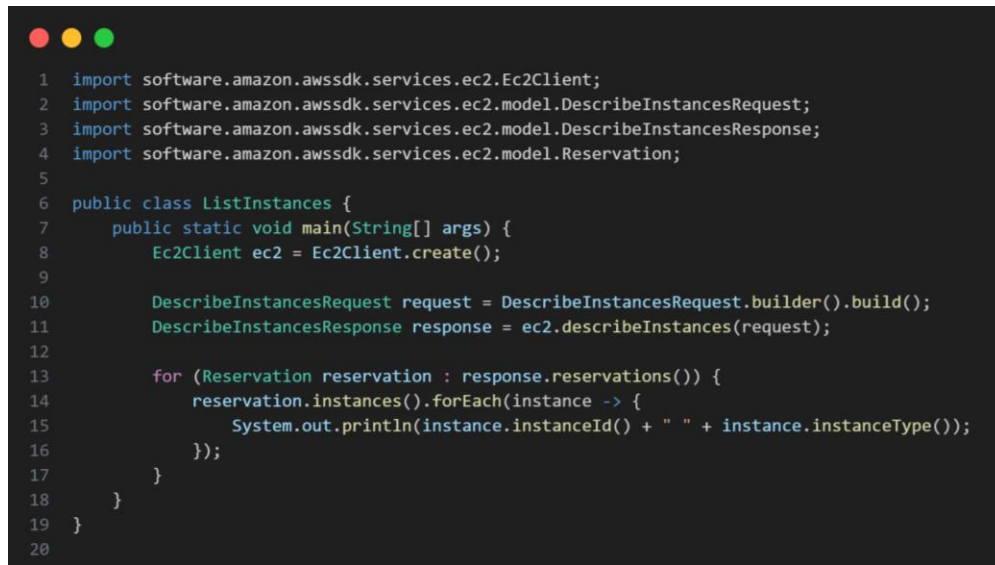
## 2. Configuración de Credenciales:

Puedes configurar las credenciales utilizando el archivo `~/.aws/credentials` o mediante variables de entorno como en los SDKs anteriores.

### 3. Ejemplo de Uso del SDK para Java:

Aquí tienes un ejemplo simple de cómo listar tus instancias de EC2 en Java:

*Ilustración 5. Instancias de EC2*

A screenshot of a code editor with a dark background and light-colored text. The code is a Java program that uses the AWS SDK to list EC2 instances. It includes imports for the EC2 client and related models, a class definition for ListInstances, and a main method that creates an EC2 client, builds a DescribeInstancesRequest, and iterates through the response to print instance IDs and types.

```
1 import software.amazon.awssdk.services.ec2.Ec2Client;
2 import software.amazon.awssdk.services.ec2.model.DescribeInstancesRequest;
3 import software.amazon.awssdk.services.ec2.model.DescribeInstancesResponse;
4 import software.amazon.awssdk.services.ec2.model.Reservation;
5
6 public class ListInstances {
7     public static void main(String[] args) {
8         Ec2Client ec2 = Ec2Client.create();
9
10        DescribeInstancesRequest request = DescribeInstancesRequest.builder().build();
11        DescribeInstancesResponse response = ec2.describeInstances(request);
12
13        for (Reservation reservation : response.reservations()) {
14            reservation.instances().forEach(instance -> {
15                System.out.println(instance.getInstanceId() + " " + instance.getInstanceType());
16            });
17        }
18    }
19 }
20
```

Nota: Fuente Propia

#### 3.3.3 Uso de AWS CLI y SDKs para Automatización y Desarrollo

Una vez que hayas instalado y configurado AWS CLI y los SDKs, puedes comenzar a utilizarlos para automatizar tareas y desarrollar aplicaciones que interactúen con los servicios de AWS de manera eficiente.

- **Automatización de Tareas:**

Con AWS CLI, puedes escribir scripts para automatizar tareas comunes, como la gestión de instancias de EC2, la carga de archivos a S3, y la implementación de funciones Lambda.

- **Desarrollo de Aplicaciones:**

Utilizando AWS SDKs, puedes desarrollar aplicaciones robustas que integren servicios de AWS como reconocimiento de voz, análisis de datos, y servicios de mensajería.

## Capítulo IV

### Desarrollo de una API Básica con AWS Lambda

El desarrollo de una API utilizando AWS Lambda es un enfoque moderno y eficiente para construir aplicaciones serverless. AWS Lambda, combinado con API Gateway, permite crear APIs altamente escalables y económicas sin la necesidad de gestionar servidores o infraestructura compleja. Este proceso involucra la creación de funciones Lambda que actúan como puntos finales de la API, la configuración de API Gateway para enrutar las solicitudes HTTP a estas funciones, y finalmente el despliegue y prueba de la API (Services, 2021).

#### 4.1 Creación de una Función Lambda Sencilla

Para empezar a desarrollar una API con AWS Lambda, es necesario crear una función Lambda que maneje las solicitudes HTTP. Esta función actuará como el backend que procesará las solicitudes entrantes.

1. **Accede a la Consola de AWS Lambda:**

Dirígete al servicio AWS Lambda en la consola de AWS y selecciona "Create Function" para iniciar la creación de una nueva función.

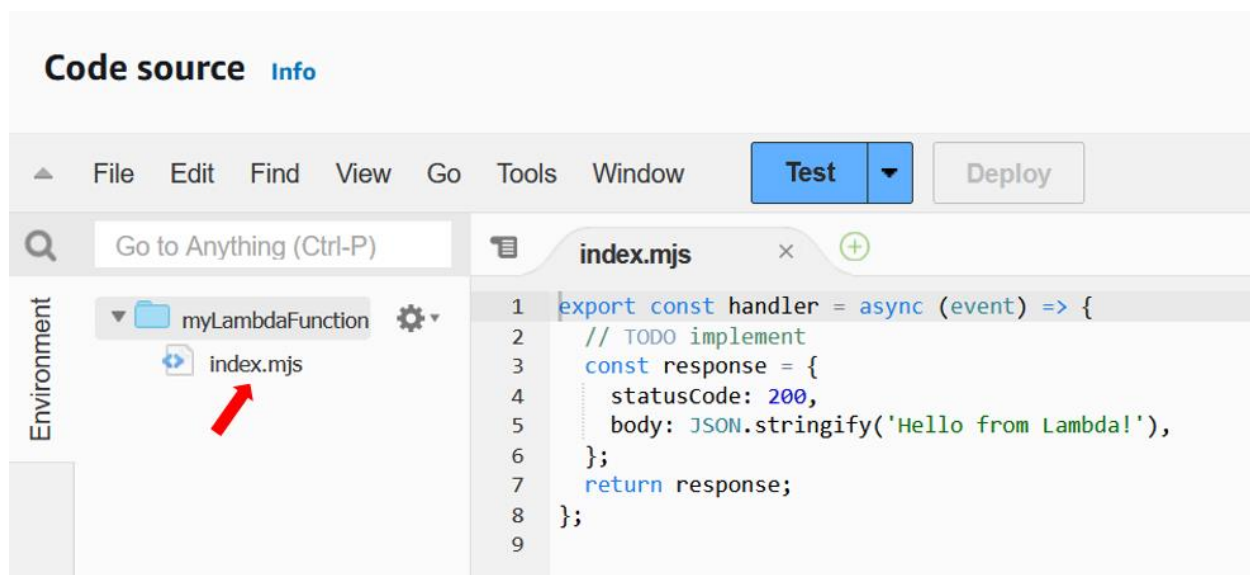
2. **Selecciona un Método de Creación:**

Elige "Author from scratch" para crear una función desde cero. Proporciona un nombre para la función y selecciona el tiempo de ejecución (runtime) que prefieras, como Python, Node.js, o Java.

3. **Configura el Handler y Código de la Función:**

Define el handler, que es el punto de entrada de la función. A continuación, ingresa el código que se ejecutará en respuesta a las solicitudes. Por ejemplo, un código básico en Node.js podría ser:

Ilustración 6. Código de la función



Nota: (Documentation, s.f.).

#### 4. Asignar Roles y Permisos:

Crea un rol de ejecución de AWS Identity and Access Management (IAM) con permisos necesarios para que la función Lambda acceda a otros servicios de AWS si es necesario.

#### 5. Configura las Opciones de la Función:

Especifica la cantidad de memoria asignada y el tiempo de ejecución máximo para la función, de acuerdo con las necesidades de tu aplicación.

#### 6. Guardar y Probar la Función:

Guarda la configuración y prueba la función con un evento de prueba para verificar que responde correctamente.

Con estos pasos, habrás creado una función Lambda básica que puede ser invocada por eventos y servir como backend de la API.

### 4.2 Configuración del AWS API Gateway

AWS API Gateway es un servicio que facilita la creación, publicación y mantenimiento de APIs RESTful. Actúa como un proxy que enruta las solicitudes HTTP a las funciones Lambda y gestiona la autorización, caché y escalabilidad de las API (Guide, s.f.).

1. **Accede a la Consola de API Gateway:**

En la consola de AWS, navega a API Gateway y selecciona "Create API". Elige "HTTP API" o "REST API" según tus necesidades.

2. **Define la API y Recursos:**

Proporciona un nombre para tu API y define los recursos (endpoints) que la API manejará. Por ejemplo, puedes crear un recurso /hello que corresponda a la función Lambda creada anteriormente.

3. **Configura Métodos HTTP:**

Asigna métodos HTTP a los recursos, como GET, POST, PUT, DELETE. Esto determina cómo las solicitudes serán manejadas y qué acciones dispararán en las funciones Lambda.

4. **Integración con Lambda:**

Vincula cada método HTTP a la función Lambda correspondiente. Esto se hace seleccionando "Lambda Function" como el tipo de integración y especificando el ARN (Amazon Resource Name) de la función.

5. **Configuración de Seguridad y Autorización:**

Define las políticas de seguridad y métodos de autenticación, como AWS IAM roles, API keys, o Amazon Cognito, para proteger los endpoints de la API.

6. **Implementar la API:**

Una vez configurada, implementa la API en un stage, como "dev" o "prod". Esto genera un endpoint de URL para la API que puede ser utilizado para realizar solicitudes.

### 4.3 Despliegue de la API

El despliegue de la API es el proceso de hacerla accesible públicamente para los clientes o aplicaciones que la consumirán. En AWS API Gateway, esto implica publicar los cambios configurados y proporcionar una URL de acceso.

1. **Crear un Stage de Implementación:**

En la consola de API Gateway, selecciona "Stages" y crea un nuevo stage, asignándole un nombre como "dev" o "prod". Este stage representa una versión específica de la API.



## 2. **Desplegar la API:**

Selecciona "Deploy API" desde el menú de acciones y elige el stage creado. Esto hace que la configuración de la API esté disponible en la URL pública del stage.

## 3. **Obtener la URL del Endpoint:**

Una vez desplegada, obtén la URL del endpoint que API Gateway ha generado. Esta URL se utilizará para acceder a los recursos de la API.

## 4. **Configurar Opciones Adicionales (Opcional):**

Configura opciones avanzadas, como la caché de respuestas, las cuotas de solicitudes, y las políticas de límite de tarifas para controlar el uso de la API.

### 4.4 Prueba de la API en el Navegador

Probar la API es crucial para garantizar que los endpoints funcionan como se espera y que la integración con AWS Lambda es correcta. Esto se puede hacer utilizando un navegador o herramientas de prueba de API como Postman o URL (Documentation, s.f.).

#### 1. **Acceder a la URL del Endpoint:**

Usa un navegador web o una herramienta de cliente HTTP para acceder a la URL del endpoint de la API. Por ejemplo, si has creado un recurso /hello, accede a `https://{api-id}.execute-api.{region}.amazonaws.com/dev/hello`.

#### 2. **Enviar Solicitudes HTTP:**

Realiza solicitudes HTTP utilizando el método configurado (GET, POST, etc.) y verifica la respuesta. Por ejemplo, una solicitud GET debería devolver la respuesta definida en la función Lambda, como "Hello from AWS Lambda!".

#### 3. **Verificar Respuestas y Logs:**

Comprueba que las respuestas de la API sean correctas y que los logs en AWS CloudWatch muestren la actividad esperada. Esto incluye verificar los códigos de estado HTTP y los datos devueltos.

#### 4. **Pruebas Adicionales:**

Si has configurado seguridad o autenticación, asegúrate de probar los endpoints con las credenciales adecuadas para validar el acceso controlado.

## Capítulo V

### Implementación de Funciones Lambda

La implementación de funciones Lambda en AWS permite crear aplicaciones serverless eficaces y escalables. AWS Lambda soporta múltiples lenguajes de programación y facilita la ejecución de código en respuesta a eventos, sin la necesidad de gestionar infraestructura. Esta sección cubre la estructura básica de una función Lambda, la programación de funciones con Node.js y Python, y la gestión de dependencias en AWS Lambda (Amazon Web Services, 2023).

#### 5.1 Estructura de una Función Lambda

Una función Lambda está compuesta por varios elementos que definen su comportamiento y cómo interactúa con otros servicios de AWS:

##### 5.1.1 Componentes Clave

1. **Handler:**

El handler es el punto de entrada de la función Lambda. Es la función que AWS Lambda invoca para ejecutar el código. El handler debe seguir una firma específica según el lenguaje de programación utilizado. Por ejemplo, en Node.js, el handler es una función exportada, mientras que en Python es un método definido en un script.

2. **Código de la Función:**

Este es el bloque de código que realiza la lógica de negocio o tarea específica que la función debe ejecutar. El código puede incluir operaciones de lectura/escritura en bases de datos, procesamiento de archivos, llamadas a APIs externas, etc.

3. **Evento:**

Un evento es un objeto JSON que contiene datos de entrada para la función Lambda. El evento puede ser generado por otros servicios de AWS, como Amazon S3, DynamoDB, o SNS, o puede ser una entrada directa desde API Gateway.

4. **Contexto:**

El objeto de contexto proporciona información sobre la ejecución de la función, como el tiempo restante, los límites de memoria, y los detalles de la solicitud de AWS Lambda.

## 5. Configuración:

Incluye ajustes como la cantidad de memoria asignada a la función, el tiempo máximo de ejecución permitido, y las variables de entorno que se utilizan para almacenar información sensible o de configuración.

## 6. Roles de IAM:

AWS Identity and Access Management (IAM) roles son necesarios para proporcionar permisos a la función Lambda para acceder a otros recursos de AWS, como S3 buckets o bases de datos RDS.

## 5.2 Programación de Funciones con Node.js y Python

AWS Lambda soporta varios lenguajes de programación, pero Node.js y Python son dos de los más populares debido a su simplicidad y amplia comunidad de desarrolladores. A continuación, se presentan ejemplos básicos de cómo programar funciones Lambda utilizando estos lenguajes (Documentation, s.f.).

### 5.2.1 Ejemplo en Node.js

El siguiente ejemplo muestra cómo crear una función Lambda en Node.js que procesa un evento simple y devuelve un mensaje:

*Ilustración 7. Ejemplo con Node.js*



```
1 // index.js
2
3 exports.handler = async (event) => {
4     // Log del evento de entrada
5     console.log('Received event:', JSON.stringify(event, null, 2));
6
7     // Procesamiento del evento
8     const name = event.name || 'Mundo';
9     const message = `¡Hola, ${name} desde AWS Lambda!`;
10
11     // Respuesta HTTP
12     const response = {
13         statusCode: 200,
14         body: JSON.stringify({ message }),
15     };
16
17     return response;
18 };
19
```

Nota: Fuente Propia.

- **Imports y Log:** La función comienza exportando el handler y registrando el evento recibido para depuración.

- **Procesamiento de Eventos:** Extrae un parámetro del evento y genera un mensaje personalizado.
- **Respuesta HTTP:** Devuelve un objeto JSON con un mensaje de respuesta que incluye un código de estado HTTP.

Para desplegar esta función en AWS Lambda, guarda el archivo `index.js` y crea un paquete ZIP que incluya todas las dependencias necesarias.

### 5.2.2 Ejemplo en Python

Aquí se presenta un ejemplo de una función Lambda escrita en Python que responde a un evento:

*Ilustración 8. Ejemplo con Python*

A screenshot of a code editor with a dark background and light-colored text. At the top left, there are three colored circles: red, yellow, and green. The code is a Python function named `lambda_handler` that takes `event` and `context` as arguments. It prints the received event, processes it by extracting the 'name' field (defaulting to 'Mundo') and creating a message, and then returns an HTTP response with a 200 status code and a JSON body containing the message. The code is numbered from 1 to 20.

```
1  # lambda_function.py
2
3  def lambda_handler(event, context):
4      # Registro del evento de entrada
5      print('Received event:', event)
6
7      # Procesamiento del evento
8      name = event.get('name', 'Mundo')
9      message = f'¡Hola, {name} desde AWS Lambda!'
10
11     # Respuesta HTTP
12     response = {
13         'statusCode': 200,
14         'body': {
15             'message': message
16         }
17     }
18
19     return response
20
```

Nota: Fuente Propia.

- **Imports y Log:** El script Python define el `lambda_handler` que actúa como el punto de entrada, registrando el evento.
- **Procesamiento de Eventos:** Utiliza el método `get()` para extraer parámetros del evento, similar al enfoque en Node.js.
- **Respuesta HTTP:** Devuelve un diccionario Python que representa la respuesta JSON, incluyendo el código de estado.

## 5.3 Gestión de Dependencias

La gestión de dependencias en AWS Lambda es esencial para garantizar que tu código tenga acceso a las librerías y módulos necesarios para ejecutarse correctamente. Dependiendo del lenguaje de programación, hay diferentes maneras de incluir dependencias en tu función Lambda.

### 5.3.1 Node.js

#### 5.3.1.1 Package.json

Las dependencias en Node.js se gestionan a través del archivo `package.json`. Puedes instalar módulos utilizando `npm` y asegurarte de que se incluyan en el archivo ZIP de implementación:

```
npm init -y
```

```
npm install axios
```

Luego, puedes importar y utilizar la librería en tu código:

```
const axios = require('axios');
```

#### 5.3.1.2 Creación del Paquete ZIP

Asegúrate de empaquetar el archivo `node_modules` junto con el código al crear el paquete ZIP:

```
zip -r function.zip index.js node_modules/
```

### 5.3.2 Python

#### 5.3.2.1 Archivo Requirements.txt

Las dependencias de Python se listan en un archivo `requirements.txt`. Puedes utilizar `pip` para instalar librerías en un directorio local:

```
pip install requests -t .
```

Luego, usa la librería en tu código:

```
import requests
```

#### **5.3.2.2 Creación del Paquete ZIP**

Empaqueta todo el directorio, incluyendo las dependencias, en un archivo ZIP

```
zip -r function.zip lambda_function.py
```

## Capítulo VI

### Manejo de Eventos y Triggers

Amazon API Gateway es un servicio gestionado por AWS que permite a los desarrolladores crear, publicar, mantener, monitorear y asegurar APIs de manera sencilla y escalable. Un aspecto crucial de API Gateway es su capacidad para gestionar eventos, que son desencadenadores que activan acciones específicas en respuesta a solicitudes entrantes. Esta sección cubre cómo configurar eventos en API Gateway para facilitar la interacción con otros servicios y sistemas.

#### 6.1 Configuración de Eventos en API Gateway

##### 6.1.1 Creación de una API

Comienza creando una nueva API en la consola de AWS API Gateway. Puedes elegir entre crear una API REST, HTTP o WebSocket, dependiendo de los requisitos de tu aplicación.

- REST API: Ideal para integrar con sistemas basados en REST, ofrece más características de gestión de tráfico y seguridad.
- HTTP API: Proporciona un camino más rápido para construir APIs HTTP de bajo costo.
- WebSocket API: Permite la comunicación bidireccional en tiempo real.

##### Pasos:

- Accede a la consola de API Gateway.
- Haz clic en Create API y selecciona el tipo de API que deseas crear.
- Proporciona un nombre y una descripción opcional para la API.

##### 6.1.2 Definición de Recursos y Métodos

Una vez que la API esté creada, define los **recursos** (endpoints) y **métodos** (GET, POST, PUT, DELETE, etc.) que formarán parte de la API.

##### Ejemplo:

- Resource: /usuarios

- Method: POST

### Configuración:

- Navega al menú de recursos en la consola.
- Haz clic en **Actions** y selecciona **Create Resource** para agregar un nuevo recurso.
- Define el método deseado para cada recurso, como GET o POST, seleccionando **Create Method**.

#### 6.1.3 Configuración de Eventos de Solicitud:

API Gateway puede desencadenar acciones específicas cuando se recibe una solicitud. Esto se configura a través de integraciones con otros servicios, como AWS Lambda, Amazon S3 o AWS Step Functions.

### Configuración de Integración:

- Selecciona el método que deseas configurar y haz clic en Integration Request.
- Elige el tipo de integración, como Lambda Function o AWS Service.
- Proporciona el ARN (Amazon Resource Name) del servicio o recurso que deseas integrar.
- Configura parámetros de solicitud y encabezados para personalizar el comportamiento.

### Ejemplo de Integración con Lambda:

```
1 {  
2   "type": "AWS_PROXY",  
3   "httpMethod": "POST",  
4   "uri": "arn:aws:apigateway:{region}:lambda:path/2015-03-31/functions/{lambdaArn}/invocations",  
5   "requestParameters": {  
6     "integration.request.header.X-Amz-Target": "'AWSStepFunctions.StartExecution'"  
7   }  
8 }  
9
```

Nota: Fuente Propia.



### 6.1.4 Transformación de Solicitudes y Respuestas

La transformación de solicitudes y respuestas es fundamental para asegurar que la API maneje los datos correctamente. API Gateway permite transformar el formato de las solicitudes entrantes y las respuestas salientes para cumplir con los requisitos específicos del backend.

#### Mapping Templates:

- Utiliza VTL (Velocity Template Language) para definir plantillas de mapeo que transforman los datos.
- Define plantillas de entrada y salida para convertir JSON, XML, o cualquier otro formato necesario.

#### Ejemplo de Plantilla de Mapeo de Solicitud:



```
1  {
2      "usuarioId": "$input.params('id')",
3      "accion": "$input.json('$.accion')"
4  }
5
```

Nota: Fuente Propia.

### 6.1.5 Configuración de CORS (Cross-Origin Resource Sharing)

Si la API será accedida desde un navegador web en un dominio diferente, es necesario configurar CORS para permitir solicitudes entre dominios.

#### Configuración de CORS:

- Selecciona el método y haz clic en Enable CORS.
- Configura los encabezados permitidos, métodos y orígenes según sea necesario.

### **6.1.6 Beneficios de Usar Eventos en API Gateway**

- Flexibilidad: Permite definir cómo las solicitudes se manejan y procesan de manera granular.
- Escalabilidad: API Gateway maneja automáticamente el escalado para soportar cargas variables.
- Seguridad: Soporta múltiples mecanismos de autenticación, como AWS IAM, Cognito, y API keys.
- Facilidad de Integración: Ofrece integración nativa con numerosos servicios de AWS, lo que facilita la construcción de aplicaciones serverless complejas.

## **6.2 Uso de Triggers para Automatización**

AWS Lambda y API Gateway se destacan por su capacidad de automatizar procesos utilizando triggers. Estos triggers permiten que una función Lambda se ejecute automáticamente en respuesta a eventos específicos generados por otros servicios de AWS o aplicaciones externas. Aquí exploraremos cómo configurar triggers para automatizar tareas y cómo estos contribuyen a la eficiencia y la reactividad de una solución serverless.

### **6.2.1 Uso de Triggers en AWS Lambda**

Un trigger en Lambda es un evento o acción que inicia la ejecución de una función. Estos pueden provenir de diversos servicios como S3, DynamoDB, Kinesis, SNS, SQS, CloudWatch Events, entre otros.

#### **6.2.1.1 Configuración de Triggers en la Consola de AWS**

- Navega a la consola de AWS Lambda y selecciona la función que deseas configurar.
- En la sección de triggers, haz clic en Add trigger.
- Selecciona el servicio que actuará como trigger (e.g., S3, DynamoDB).
- Configura los parámetros específicos del trigger, como el bucket de S3 o la tabla de DynamoDB, y guarda la configuración.

### 6.2.1.2 Ejemplo de Configuración de Triggers

- Evento: Carga de un archivo en un bucket S3.
- Acción Lambda: Procesamiento del archivo para generar miniaturas de imágenes.

#### 6.2.1.2.1 Configuración

- Bucket S3: my-images-bucket
- Tipo de evento: s3:ObjectCreated:\*
- Función Lambda: ImageThumbnailGeneratorFunction



```
1  {
2    "Records": [
3      {
4        "s3": {
5          "bucket": {
6            "name": "my-images-bucket"
7          },
8          "object": {
9            "key": "image.jpg"
10         }
11       }
12     ]
13   }
14 }
15
```

Nota: Fuente Propia.

### 6.2.2 Trigger en DynamoDB

- Evento: Nuevo registro insertado en una tabla DynamoDB.
- Acción Lambda: Actualización de un índice de búsqueda.
- Configuración:
- Tabla DynamoDB: Usuarios
- Evento de Streams: INSERT
- Función Lambda: UpdateSearchIndexFunction

```
1 {
2   "Records": [
3     {
4       "eventName": "INSERT",
5       "dynamodb": {
6         "NewImage": {
7           "id": {
8             "S": "123"
9           },
10          "nombre": {
11            "S": "Jane Smith"
12          }
13        }
14      }
15    ]
16  }
17 }
```

Nota: Fuente Propia.

### 6.2.3 Integración con CloudWatch Events

CloudWatch Events permite programar tareas y monitorear eventos de cambios en la infraestructura, lo cual es ideal para automatizar tareas basadas en eventos recurrentes o condiciones específicas.

#### 6.2.3.1 Ejemplo de Uso de CloudWatch Events

- Evento: Ejecución programada diaria.
- Acción Lambda: Generación de reportes y envío por correo electrónico.
- Configuración:
- Regla de CloudWatch Events: rate(1 day)
- Función Lambda: DailyReportGeneratorFunction

```
1 {
2   "source": "aws.events",
3   "detail-type": "Scheduled Event",
4   "time": "2024-08-07T12:00:00Z",
5   "region": "us-east-1"
6 }
7
```

Nota: Fuente Propia.

### **Bibliografía**

Amazon Web Services. (2023). *Creación de API sin servidor en AWS*.

Documentation, A. A. (s.f.). *AWS*.

Guide, A. L. (s.f.). *AWS*.

Services, A. W. (2021). *Arquitecturas de varios niveles sin servidor de AWS con Amazon API Gateway y AWS Lambda*. Documento técnico de AWS.