

Desafío Manguear: Creación de una API REST con AWS Lambda y Serverless Framework

Durante el siguiente informe me enfocaré en explicar mi entendimiento y resolución al desafío planteado en este challenge laboral.

En este, se me pidió la creación, configuración y despliegue de una API REST sencilla utilizando **AWS Lambda** y el **Serverless Framework** y que responda con el mensaje "Hello world" al recibir una solicitud HTTP.

Antes de comenzar, debemos descargar y/o configurar las siguientes tecnologías:

- **Node.js.**
- **NPM** (gestor de paquetes de Node.js).
- **Serverless Framework.**
- **AWS CLI y acceso con credenciales adecuadas para AWS.**

Una vez configurado el entorno, aquí comienza el desarrollo y resolución de este desafío:

Creación del proyecto

Dada la documentación de Amazon Web Service, procedí a la interacción y navegación del framework Serverless desde mi consola de Ubuntu, para la creación de la plantilla y su configuración, y su posterior migración al directorio del proyecto (Al cual llame "DesafioManguear"). Esta configuración consistió en la creación de dependencias en el directorio, con el esquema AWS + Python + HTTP API, lo que derivó en la creación automática de un directorio con archivos **handler.py** para la API y donde se realizará la función lambda, un archivo **serverless.yml** como endpoint de esta, un **meta.json** que contiene los metadatos necesarios para la funcionalidad de Serverless, y un archivo de texto **requirements.txt**, que de manera automática contiene librerías y dependencias necesarias para el uso de entornos automatizados como AWS Lambda.

Desarrollo de la API y configuración del Endpoint

Ahora bien, el desafío principal corresponde a justamente este título. Como mencioné antes, el framework Serverless configurado me brindó en el directorio, la creación de los esqueletos para todos los archivos, pero por supuesto que estos no están desarrollados, probados y siquiera tiene generados los endpoints. De esta manera, lo primero que hice fue definir una API en Python pero importando la biblioteca JSON, para que al compilar, esta tenga un tipado y formato de impresión adecuado a este modelo.

Aquí, definí una función `hello_world` (a la cual la llame así por el objetivo de la devolución de respuesta) en donde primero define una variable mensaje a la que le pase por parametro esta cadena a devolver ("Hello world!"). Debajo una variable `body`, que representa el cuerpo del mensaje y que recibe el valor de la cadena de mensaje, para luego devolver "message": "Hello world!". Por último, y dada la configuración automática del serverless al definir el estatus del código (definida en 200), cree una ultima variable respuesta, que contenga este

estatus y un body, que recibe el cuerpo del mensaje para la respuesta por consola, y la genera como JSON. De esta manera retornamos a la respuesta para que la función de la API retorne al contenido del mensaje a recibir y su estatus.

```

import json

def hello_world(evento, contexto):

    mensaje = "Hello world!"

    body = {
        "message": mensaje
    }

    respuesta = {
        "statusCode": 200,
        "body": json.dumps(body)
    }

    return respuesta

```

BLEMS OUTPUT TERMINAL PORTS

```

ri@DESKTOP-VEJJ5UV:~/DesafioManguear/desafioManguear$ serverless invoke local --function helloworld

{"statusCode": 200,
 "body": "{\"message\": \"Hello world!\"}"

```

Hecha la API, procedemos a configurar el endpoint en serveless.yml. Este fue más breve y sencillo ya que cada dependencia del framework y AWS estaba ya escrita y generada automáticamente en la previa configuración. Sin embargo por supuesto que faltaba definir el endpoint a interactuar desde el handler (el cual fue la función hello_world) y definir el PATH hello para que al recibir la solicitud http en el endpoint devuelva este.

```

1 # "service" is the name of this project. This will also be added to your AWS resource names.
2 service: desafioManguear
3
4 provider:
5   name: aws
6   runtime: python3.12
7
8 functions:
9   helloworld:
10     handler: handler.hello_world
11     events:
12       - httpApi:
13         path: /hello
14         method: get

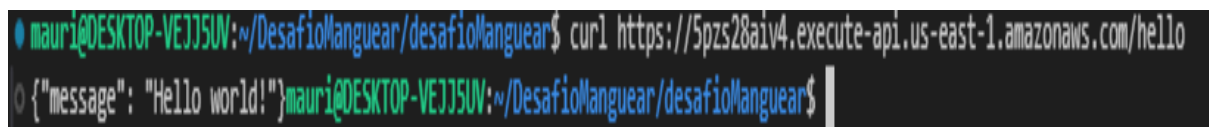
```

Deploy, despliegue y prueba del proyecto

Definidos todos los contenidos de los archivos del directorio, primero desde el comando `serverless invoke local --function helloworld`, probamos localmente nuestro proyecto para verificar que funciona, dando sensaciones positivas antes de desplegarla ya que devolvió el contenido esperado por el challenge:

```
{
  "statusCode": 200,
  "body": "{\"message\": \"Hello world!\"}"
}
```

Posteriormente se procedió al deploy y prueba de la API en aws, a través del comando `serverless deploy` para no solo crear un endpoint publico, sino que ademas devolver un url publico donde probar esta devolución de la api que interactúa con el get del endpoint. por ultimo, a traves del comando `curl` <https://5pzs28aiv4.execute-api.us-east-1.amazonaws.com/hello> procedi a probar el contenido de la API, resultando en un desenlace positivo para el final del proyecto ya que devolvio lo que se esperaba:

A terminal window with a dark background. The prompt is 'mauri@DESKTOP-VEJJ5UV: ~/DesafioManguear/desafioManguear\$'. The command entered is 'curl https://5pzs28aiv4.execute-api.us-east-1.amazonaws.com/hello'. The output is '{\"message\": \"Hello world!\"}' followed by the prompt again. A vertical cursor is visible at the end of the output line.

```
mauri@DESKTOP-VEJJ5UV: ~/DesafioManguear/desafioManguear$ curl https://5pzs28aiv4.execute-api.us-east-1.amazonaws.com/hello
{"message": "Hello world!"}mauri@DESKTOP-VEJJ5UV: ~/DesafioManguear/desafioManguear$
```

Fuentes y documentación utilizada para este proyecto:

https://docs.aws.amazon.com/es_es/serverless-application-model/latest/developerguide/serverless-getting-started-hello-world.html

https://docs.aws.amazon.com/es_es/serverless-application-model/latest/developerguide/what-is-sam.html