Guía de despliegue de aplicación en AWS Lambda con Serverless.js

Introducción

Esta documentación describe los pasos necesarios para desplegar una aplicación en Serverless utilizando el Serverless Framework y AWS como proveedor de servicios.

Requisitos Previos

Antes de comenzar, se deben tener instalados y configurados los siguientes elementos:

1. **Node.js y npm**

   Verificar que Node.js y npm estén instalados en tu máquina ejecutando los siguientes comandos en la terminal:

   ```
   node -v
   npm -v
   ```

2. **Serverless Framework**: Instala Serverless Framework globalmente mediante el siguiente comando:

   ```
   npm install -g serverless
   ```

3. **Cuenta de AWS**: Necesitarás una cuenta de AWS para desplegar tu aplicación. Importante asegurarse que nuestras credenciales de AWS estén configuradas correctamente en tu máquina local.

Pasos para Desplegar la Aplicación

1. **Crear un Nuevo Proyecto Serverless:** En la terminal, navega a la carpeta donde deseas crear tu proyecto y ejecuta el siguiente comando:

```
serverless create --template aws-nodejs --path nombre-de-tu-proyecto
```

Con <nombre-de-tu-proyecto> siendo pdf-generator.

2. **Instalar Dependencias:** Instalar pdfkit y aws-sdk ejecutando:

```
npm install pdfkit aws-sdk
```

3. **Configurar los archivos handler.js y serverless.yml:** Estos archivos están dentro del directorio del proyecto creado con serverless. Para que las llamadas a las funciones se ejecuten de manera correcta, es necesario configurara el archivo serverless.yml para que se conecte con nuestra bucket que almacenará las boletas, que en mi caso se llama 'boletas-de-compras', mientras que handler.js posee las funciones que se ejecutarán cuando se llame a esta función lambda mediante requests a su api. Los archivos serverless.yml y handler.js se encuentran en el anexo.

4. **Hacer deploy a AWS Lambda:** El último paso corresponde a ejecutar el siguiente comando, que creará creará la infraestructura necesaria en AWS y desplegará tus funciones Lambda:

```
serverless deploy
```

Anexo

Serverless.yml:

serverless.yml

```yaml
service: generacion-boletas-pdf

provider:
  name: aws
  runtime: nodejs18.x
  region: us-east-1
  deploymentBucket:
    name: boletas-de-compra
  environment:
    BUCKET_NAME: boletas-de-compra
  iamRoleStatements:
    - Effect: "Allow"
      Action:
        - "s3:PutObject"
        - "s3:GetObject"
      Resource:
        - "arn:aws:s3:::boletas-de-compra/*"

functions:
  generatePdf:
    handler: handler.generatePdf
    events:
      - http:
          path: pdf
          method: post
          cors: true
  getPdfs:
    handler: handler.getPdfs
    events:
      - http:
          path: pdfs
          method: get
          cors: true

plugins:
  - serverless-offline

resources:
  Resources:
    PdfBucketPolicy:
      Type: AWS::S3::BucketPolicy
      Properties:
        Bucket: boletas-de-compra
        PolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Effect: Allow
              Principal: "*"
              Action: "s3:GetObject"
              Resource: "arn:aws:s3:::boletas-de-compra/*"
```

Handler.js:

```
const PDFDocument = require('pdfkit');
const AWS = require('aws-sdk');

const s3 = new AWS.S3();
const BUCKET_NAME = 'boletas-de-compra';

// Función para generar un PDF con la información de la compra y postearlo en el Bucket de S3

module.exports.generatePdf = async (event) => {
  const { groupName, userName, teamsInfo } = JSON.parse(event.body);

  const sanitizedUserName = userName.replace(/\s+/g, '-');
  const doc = new PDFDocument();
  let buffers = [];

  doc.on('data', (chunk) => buffers.push(chunk));

  const generatePdfPromise = new Promise((resolve, reject) => {
    doc.on('end', async () => {
      const pdfData = Buffer.concat(buffers);
      const params = {
        Bucket: BUCKET_NAME,
        Key: `${sanitizedUserName}-${Date.now()}.pdf`,
        Body: pdfData,
        ContentType: 'application/pdf',
      };

      try {
        await s3.putObject(params).promise();
        const pdfUrl = `https://${BUCKET_NAME}.s3.amazonaws.com/${params.Key}`;
        resolve(pdfUrl);
      } catch (error) {
        reject(error);
      }
    });
  });

  doc.text('Boleta de compra');
  doc.text(`Nombre del grupo: ${groupName}`);
  doc.text(`Usuario que realizó la compra: ${userName}`);
  doc.text('Equipos:');
  teamsInfo.forEach((team) => {
    doc.text(`- ${team}`);
  });
  doc.text('Fecha y hora de compra: ' + new Date().toLocaleString('es-CL', { timeZone: 'America/Santiago' }));
  doc.text('Gracias por su compra');
  doc.end();

  try {
    const pdfUrl = await generatePdfPromise;
    return {
      statusCode: 200,
      body: JSON.stringify({ message: 'PDF generado exitosamente', url: pdfUrl }),
    };
  } catch (error) {
    return {
      statusCode: 500,
      body: JSON.stringify({ error: 'Error al subir el PDF a S3', details: error.message }),
    };
  }
};
```

(continua en la siguiente página)

```
62
63  // Función para obtener los PDFs de un usuario en particular
64
65  module.exports.getPdfs = async (event) => {
66    const { userName } = event.queryStringParameters;
67    const sanitizedUserName = userName.replace(/\s+/g, '-');
68
69    const params = {
70      Bucket: BUCKET_NAME,
71      Prefix: sanitizedUserName
72    };
73
74    try {
75      const data = await s3.listObjectsV2(params).promise();
76
77      // Crear un array de objetos que contenga la fecha y la URL
78      const pdfData = data.Contents.map(item => ({
79        date: item.LastModified,
80        url: `https://${BUCKET_NAME}.s3.amazonaws.com/${item.Key}`
81      }));
82
83      // Ordenar por fecha (más reciente primero)
84      pdfData.sort((a, b) => new Date(b.date) - new Date(a.date));
85
86      return {
87        statusCode: 200,
88        body: JSON.stringify({ pdfs: pdfData }),
89      };
90    } catch (error) {
91      return {
92        statusCode: 500,
93        body: JSON.stringify({ error: `Error al listar PDFs: ${error.message}` }),
94      };
95    }
96  };
97
98
```