Trabajo 8: Función Lambda Dirigida por Eventos

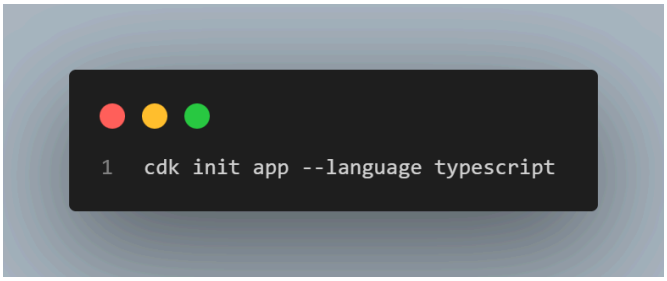
Implementación de un Bucket para Contabilización de Palabras en Archivos mediante Infraestructura como Código con TypeScript

Ismael Illán García
Daniel Herrero Pardo

Repositorio: https://github.com/danihp6/icpwork

Creación de la infraestructura

Primero se ha iniciado el proyecto con el cli de cdk.



Se ha creado un stack “DatabaseStack” para el bucket que contiene los textos, con política DESTROY para que cuando se borre el stack no dejé rastro (no recomendado para proyectos reales).

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as s3 from 'aws-cdk-lib/aws-s3';
import { Context } from '../../context';
import { CONSTANTS } from '../../constants';

export class DatabaseStack extends cdk.Stack {
  public readonly photosBucket: s3.Bucket;

  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);
    const context = new Context();

    this.photosBucket = new s3.Bucket(this, 'PhotosBucket', {
      bucketName: context.getFullName(CONSTANTS.phothosBucketName),
      removalPolicy: cdk.RemovalPolicy.DESTROY,
    });
  }
}
```

También un stack “ComputeStack” para la lambda que aplica la contabilización de palabras, creamos una función en nodejs con el path donde está el código de la lambda, con los permisos para obtener y borrar los archivos de texto en input/* y para que pueda subir las contabilizaciones de palabras sobre output/*.

```
import * as cdk from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as lambdanodejs from 'aws-cdk-lib/aws-lambda-nodejs';
import * as iam from 'aws-cdk-lib/aws-iam';
import { Context } from '../../context';
import { CONSTANTS } from '../../constants';
import * as path from 'path';
import * as logs from 'aws-cdk-lib/aws-logs';

const lambdasPath = path.join(__dirname, '../../src/lambdas');

export class ComputeStack extends cdk.Stack {
  public readonly filterLambda: lambda.IFunction;

  constructor(scope: Construct, id: string, props?: cdk.StackProps) {
    super(scope, id, props);

    const context = new Context();

    const photosBucketName = context.getFullName(CONSTANTS.phothosBucketName);

    this.filterLambda = new lambdanodejs.NodejsFunction(this, 'filter', {
      functionName: context.getFullName('filter'),
      entry: path.join(lambdasPath, 'filter', 'index.ts'),
      runtime: lambda.Runtime.NODEJS_18_X,
      timeout: cdk.Duration.seconds(3),
      initialPolicy: [
        new iam.PolicyStatement({
          effect: iam.Effect.ALLOW,
          actions: [
            's3:GetObject',
            's3:DeleteObject'
          ],
          resources: [
            `arn:aws:s3:::${photosBucketName}/input/*`
          ]
        }),
        new iam.PolicyStatement({
          effect: iam.Effect.ALLOW,
          actions: [
            's3:PutObject',
          ],
          resources: [
            `arn:aws:s3:::${photosBucketName}/output/*`
          ]
        })
      ],
      logRetention: logs.RetentionDays.ONE_DAY
    });
  }
}
```

Y finalmente un stack de integración “IntegrationStack” para que invoque la lambda en cada archivo **solo** sobre la carpeta input/*

```typescript
import * as cdk from "aws-cdk-lib";
import { Construct } from "constructs";
import * as s3 from "aws-cdk-lib/aws-s3";
import * as s3n from "aws-cdk-lib/aws-s3-notifications";
import * as lambda from "aws-cdk-lib/aws-lambda";

interface IntegrationStackProps extends cdk.StackProps {
  photosBucket: s3.IBucket;
  filterLambda: lambda.IFunction;
}

export class IntegrationStack extends cdk.Stack {
  constructor(scope: Construct, id: string, props: IntegrationStackProps) {
    super(scope, id, props);

    const { photosBucket, filterLambda } = props;

    photosBucket.addEventNotification(
      s3.EventType.OBJECT_CREATED,
      new s3n.LambdaDestination(filterLambda),
      { prefix: "input/" }
    );
  }
}
```

En app.ts se construyen los stacks, donde “IntegrationStack” depende de “ComputeStack” y “DatabaseStack”.

```typescript
import * as cdk from 'aws-cdk-lib';
import { ComputeStack, DatabaseStack, IntegrationStack } from '../lib/stacks';
import { Context } from '../lib/context';

const app = new cdk.App();
const context = new Context();

const computeStack = new ComputeStack(app, context.getFullName('ComputeStack'));
const databaseStack = new DatabaseStack(app, context.getFullName('DatabaseStack'));
new IntegrationStack(app, context.getFullName('IntegrationStack'), {
  photosBucket: databaseStack.photosBucket,
  filterLambda: computeStack.filterLambda
});
```

Código de la lambda

Esta función Lambda en TypeScript se encarga de leer un archivo de texto desde un bucket de Amazon S3, contar la cantidad de veces que aparece cada palabra y guardar el resultado en un nuevo archivo dentro del mismo bucket. Primero, utiliza el SDK de AWS para obtener el archivo desde la carpeta `input/`. Luego, convierte el contenido del archivo en una cadena de texto, elimina diferencias de mayúsculas y minúsculas y utiliza una expresión regular para extraer todas las palabras. Posteriormente, contabiliza la frecuencia de cada palabra y genera un archivo con los resultados en formato `palabra: cantidad`. Finalmente, la función sube este archivo procesado a la carpeta `output/` en el bucket de S3. Si ocurre algún error durante la ejecución, la función lo captura y lo devuelve en la respuesta.

```typescript
import * as AWS from "aws-sdk";

const s3 = new AWS.S3();

export const handler = async (event: any) => {
  const bucketName = process.env.BUCKET_NAME || "";
  try {
    const record = event.Records?.[0];
    const inputKey = record.s3.object.key;
    if (!inputKey.startsWith("input/")) {
      console.log("El archivo no está en la carpeta input/, se ignora.");
      return {
        statusCode: 200,
        body: "Archivo fuera de la carpeta input/, ignorado.",
      };
    }
    const data = await s3
      .getObject({ Bucket: bucketName, Key: inputKey })
      .promise();
    if (!data.Body) throw new Error("El archivo está vacío o no se pudo leer.");
    const text = data.Body.toString("utf-8");
    const words = text.toLowerCase().match(/\b\w+\b/g) || [];
    const wordCount: Record<string, number> = {};
    words.forEach((word) => {
      wordCount[word] = (wordCount[word] || 0) + 1;
    });
    const outputContent = Object.entries(wordCount)
      .map(([word, count]) => `${word}: ${count}`)
      .join("\n");
    const outputKey = inputKey
      .replace("input/", "output/")
      .replace(".txt", "_wordcount.txt");
    await s3
      .putObject({
        Bucket: bucketName,
        Key: outputKey,
        Body: outputContent,
        ContentType: "text/plain",
      })
      .promise();
    return {
      statusCode: 200,
      body: JSON.stringify({
        message: "Conteo de palabras generado",
        outputKey,
      }),
    };
  } catch (error: any) {
    console.error("Error:", error);
    return { statusCode: 500, body: JSON.stringify({ error: error.message }) };
  }
};
```

Para probar

Primero instalar dependencias, hacer bootstrap en la cuenta si es que no se ha realizado ya y desplegar

Luego subir un fichero de texto a la carpeta input/ del s3, en el código hay un archivo text.txt de prueba.