Informe

**Tema:** Desarrollo una API GetWay mediante el kit de herramientas AWSCDK

# Introducción:

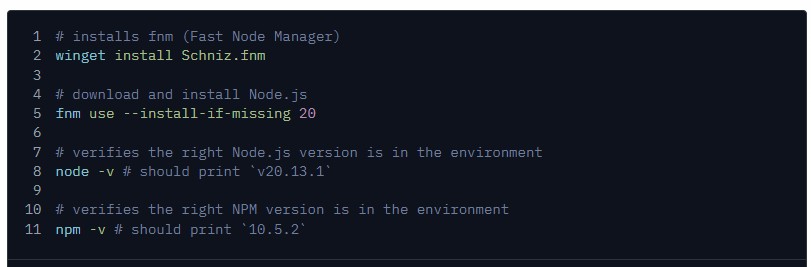
AWS CDK (Cloud Development Kit ) , es un marco de desarrollo de software de código abierto para definir la infraestructura de la nube en el código y aprovisionarla mediante ella AWS CloudFormation.

# Requisitos previos

AWS identity and Access Management, tenemos saber ¿Qué es IAM? Y ¿Qué es el centro de identidad de IAM?

# Prepare su entorno Local:

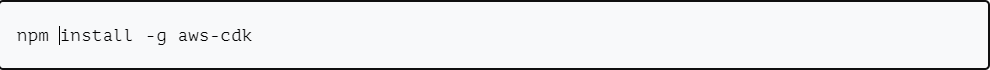
Instale en su maquina el Node.Js 14.15.0 o una versión posterior independientemente al lenguaje de programación que escoja.



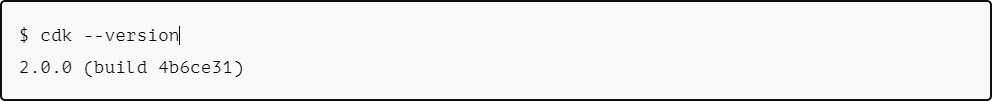
Nota: El Node.js ya incluye npm(10.5.2) que permitirá instalar los demás paquetes de AWS CDK.

A continuación, instalaremos el kit de herramientas AWS CDK. El kit de herramientas es una utilidad de línea de comandos que le permite trabajar con aplicaciones CDK.

Abra una sesión de terminal y ejecute el siguiente comando:



Windows: necesitarás ejecutar esto como administrador Puede consultar la versión del kit de herramientas:



El lenguaje TypeScript es un idioma de cliente totalmente compatible AWS CDK y se considera estable.

npm install -g typescript

Deberás contar también con un editor o IDE de código. Muchos AWS CDK desarrolladores utilizan Visual Studio Code ( código abierto)

Esta IDE deberá tener las librerías e instalar herramientas como “Codium” que hará más sencillo estructurar su código y construir la infraestructura basada en líneas de Código.

# Creación de un proyecto

mkdir my-project cd my-project

cdk init app --language typescript

Al crear un proyecto, también se instala el aws-cdk-libmódulo y sus dependencias. cdk init --language typescript

cdk init usa el nombre de la carpeta del proyecto para nombrar varios elementos del proyecto, incluidas las clases, las subcarpetas y los archivos. Los guiones del nombre de la carpeta se convierten en guiones bajos. Sin embargo, de lo contrario, el nombre debe tener la forma de un TypeScript identificador; por ejemplo, no debe empezar por un número ni contener espacios.

# Paso1: Crea una Cuenta de AWS

Si es la primera vez que lo usa AWS, debe registrarse Cuenta de AWS y crear un usuario administrativo.

Cuando interactúa con AWS, especifica sus credenciales de AWS seguridad para comprobar quién es y si tiene permiso para acceder a los recursos que solicita. AWS utiliza las credenciales de seguridad para autenticar y autorizar sus solicitudes.

Como crear perfiles de AWS locales para tener acceso programático los diferentes servicios de nuestra cuenta en AWS

# Paso2: Configurar el acceso mediante programación

Al desarrollar AWS CDK en su entorno local, confiará en ellos AWS CDK CLI para interactuar con sus AWS recursos Servicios de AWS y administrarlos. Para utilizar el AWS CDK CLI, debe configurar el acceso programático. Para obtener más información sobre las diferentes formas de configurar el acceso mediante programación, consulte Autenticación y acceso en la Guía de referencia de herramientas y AWS SDK.

Para los nuevos usuarios a los que su empresa no les haya proporcionado un método de autenticación, les recomendamos que lo utilicen. AWS IAM Identity Center Este método incluye instalar el AWS Command Line Interface (AWS CLI) y usarlo para configurar e iniciar sesión en el portal de AWS acceso. Para configurar el acceso mediante

programación mediante el Centro de identidad de IAM, consulte la autenticación del Centro de identidad de IAM:

Deberán ir al centro de IAM en AWS consola, ahí deberán crear un grupo de trabajo y crearán un usuario dándole permisos respectivos para el entorno de desarrollo, este nos proporcionara las Security Credentials:

Acces key xxxxxxx

Secret acces key xxxxxxxxxxxxxxxxxxxxx

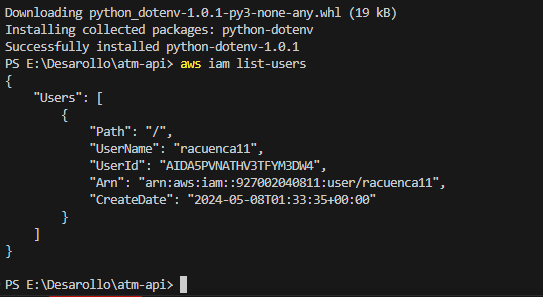
# Paso3: Instala el AWS CDKCLI

Primero debemos descargar e instalar la AWSCLI o la interfaz de comandos de AWS dependiendo su sistema operativo lo escogerán.

Una vez instalada verificaran vía consola en terminal línea de comandos:

Texto

Descripción generada automáticamente



# Paso4: Inicie su entorno

Se debe iniciar cada AWS entorno en el que planea implementar recursos. Para arrancar, ejecute lo siguiente:

cdk bootstrap aws://ACCOUNT-NUMBER/REGION

**Paso final:** Para levantar nuestra infraestructura basada en código

Liego de importar todas las librerías necesarias y descargar todas las dependencias necesarias para implementar una API en CDK estamos listos para desplegar e implementar toda nuestra infraestructura, mediante línea de comandos en consola:

cdk deploy

# Salidas de pila #

Cuando se complete la implementación, verá esta línea de comando:

CdkWorkshopStack.Endpoint8024A810 = https://xxxxxxxxxx.execute-api.us-east- 1.amazonaws.com/prod/

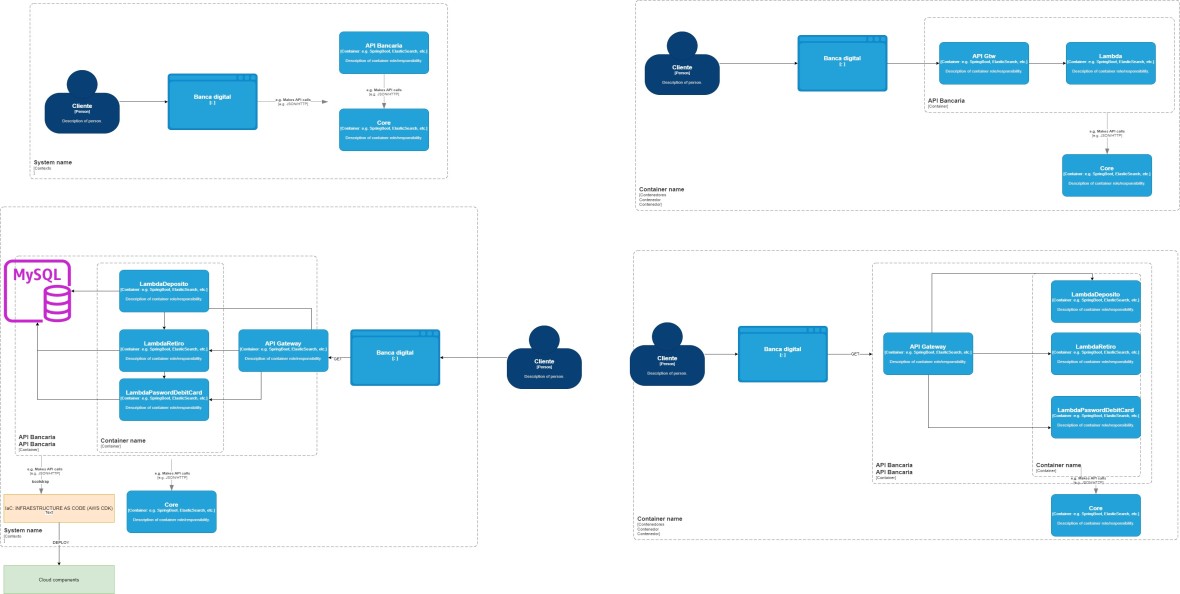
Esta es una salida de pila que la construcción de API Gateway agrega automáticamente e incluye la URL del punto final de API Gateway.

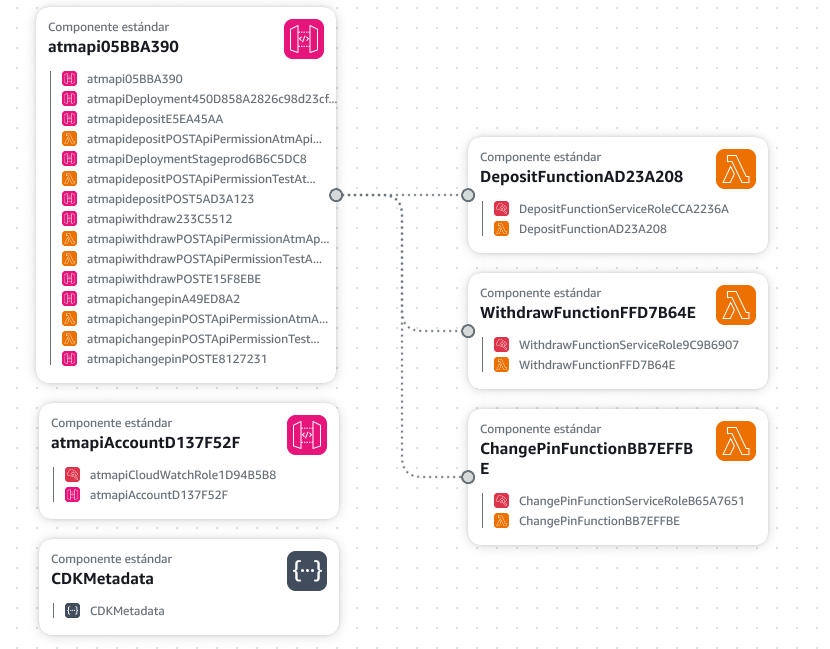
# Probando tu aplicación #

Intentemos llegar a este punto final con curl. Copie la URL y ejecútela (su prefijo y región probablemente serán diferentes).

Si no tiene curl instalado, siempre puede usar su navegador web favorito para acceder a esta URL.

https://xxxxxxxxxx.execute-api.us-east-1.amazonaws.com/prod/

Aquí el esquema de como debería desarrollase la solución mediante los servicios AWS:}



1Aplicacion con sus componentes

Texto

Descripción generada automáticamente

Modulo para encriptación de contraseña

Interfaz de usuario gráfica

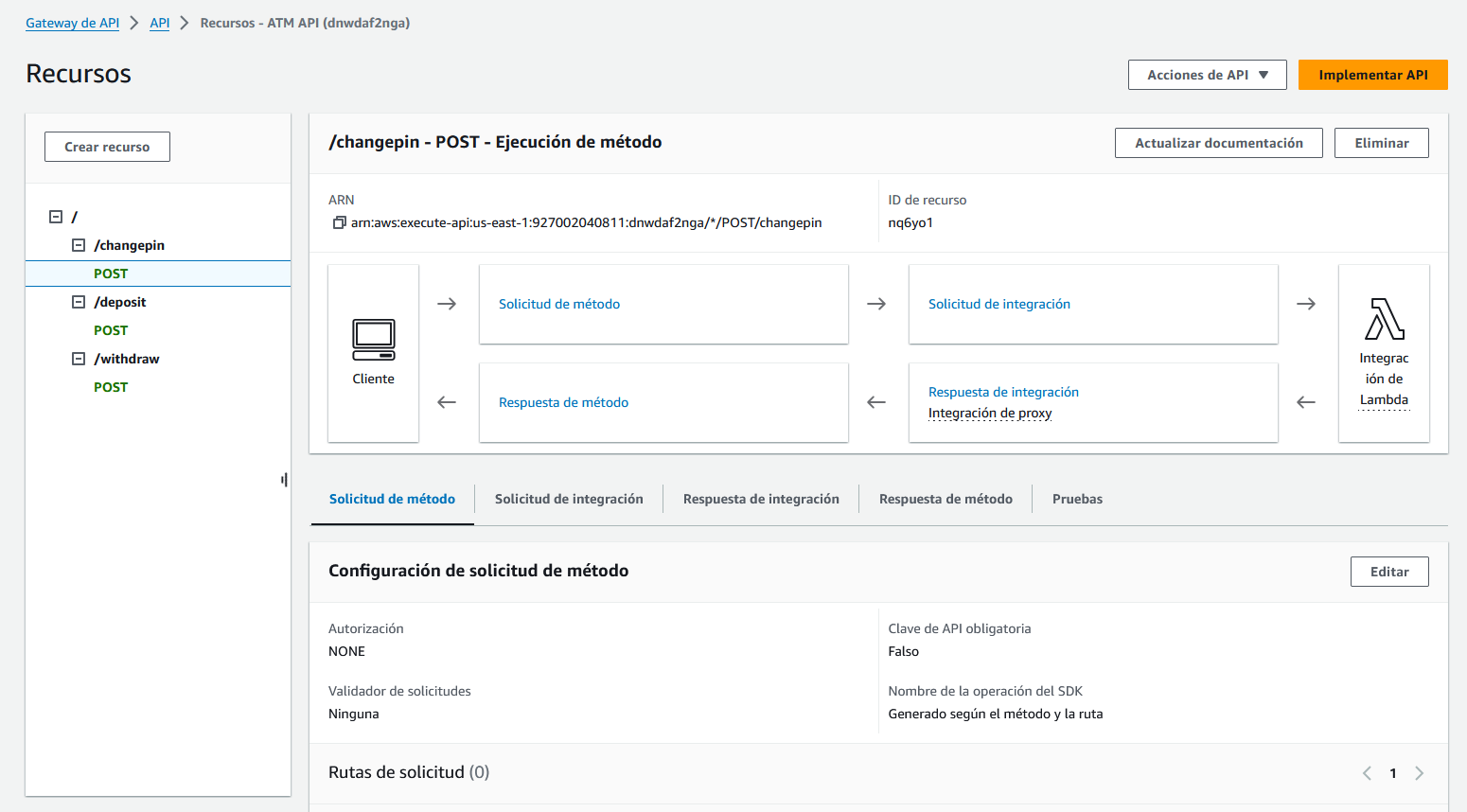
Descripción generada automáticamente con confianza baja

Despliegue exitoso

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Pilas



Api Gateway post para cada Lambda

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Desencadenadores

Código Changepin:

import pymysql

import json

import os

import bcrypt

from email\_utils import send\_email

def handler(event, context):

    body = json.loads(event['body'])

    numero\_cuenta = body['numeroCuenta']

    clave\_antigua = body['claveAntigua']

    clave\_nueva = body['claveNueva']

    connection = pymysql.connect(

        host=os.environ['DB\_HOST'],

        user=os.environ['DB\_USER'],

        password=os.environ['DB\_PASSWORD'],

        database=os.environ['DB\_NAME'],

        port=int(os.environ['DB\_PORT'])

    )

    try:

        with connection.cursor() as cursor:

            cursor.execute("SELECT claveTarjeta, correoElectronico FROM CuentaBancaria WHERE numeroCuenta = %s", (numero\_cuenta,))

            result = cursor.fetchone()

            if result is None:

                response = {

                    'statusCode': 404,

                    'body': json.dumps({'message': 'Cuenta no encontrada'})

                }

            else:

                clave\_hash = result[0]

                to\_email = result[1]

                if not bcrypt.checkpw(clave\_antigua.encode('utf-8'), clave\_hash.encode('utf-8')):

                    response = {

                        'statusCode': 400,

                        'body': json.dumps({'message': 'Clave antigua incorrecta'})

                    }

                else:

                    nuevo\_hash = bcrypt.hashpw(clave\_nueva.encode('utf-8'), bcrypt.gensalt())

                    cursor.execute("UPDATE CuentaBancaria SET claveTarjeta = %s WHERE numeroCuenta = %s", (nuevo\_hash.decode('utf-8'), numero\_cuenta))

                    connection.commit()

                    # Enviar correo electrónico

                    subject = 'Cambio de Clave Realizado'

                    email\_body = f'Se ha cambiado la clave de la tarjeta para la cuenta {numero\_cuenta}.'

                    send\_email(to\_email, subject, email\_body)

                    response = {

                        'statusCode': 200,

                        'body': json.dumps({'message': 'Cambio de clave realizado con éxito'})

                    }

    except Exception as e:

        response = {

            'statusCode': 500,

            'body': json.dumps({'message': str(e)})

        }

    finally:

        connection.close()

    return response

**Codigo deposit.py:**

import pymysql

import json

import os

import bcrypt

from email\_utils import send\_email

def handler(event, context):

    body = json.loads(event['body'])

    numero\_cuenta = body['numeroCuenta']

    monto = body['monto']

    connection = pymysql.connect(

        host=os.environ['DB\_HOST'],

        user=os.environ['DB\_USER'],

        password=os.environ['DB\_PASSWORD'],

        database=os.environ['DB\_NAME'],

        port=int(os.environ['DB\_PORT'])

    )

    try:

        with connection.cursor() as cursor:

            cursor.execute("UPDATE CuentaBancaria SET saldo = saldo + %s WHERE numeroCuenta = %s", (monto, numero\_cuenta))

            cursor.execute("SELECT correoElectronico FROM CuentaBancaria WHERE numeroCuenta = %s", (numero\_cuenta,))

            result = cursor.fetchone()

            cursor.execute("INSERT INTO Transaccion (tipo, monto, idCuenta) VALUES ('Deposito', %s, (SELECT id FROM CuentaBancaria WHERE numeroCuenta = %s))", (monto, numero\_cuenta))

        connection.commit()

        # Enviar correo electrónico

        to\_email = result[0]

        subject = 'Depósito Realizado'

        email\_body = f'Se ha realizado un depósito de {monto} en la cuenta {numero\_cuenta}.'

        send\_email(to\_email, subject, email\_body)

        response = {

            'statusCode': 200,

            'body': json.dumps({'message': 'Depósito realizado con éxito'})

        }

    except Exception as e:

        response = {

            'statusCode': 500,

            'body': json.dumps({'message': str(e)})

        }

    finally:

        connection.close()

    return response

**Código email\_utils.py**

import os

from email.message import EmailMessage

import ssl

import smtplib

def send\_email(to\_email, subject, body):

    email\_sender = os.environ['EMAIL\_SENDER']

    password = os.environ['EMAIL\_PASSWORD']

    em = EmailMessage()

    em['From'] = email\_sender

    em['To'] = to\_email

    em['Subject'] = subject

    em.set\_content(body)

    context = ssl.create\_default\_context()

    with smtplib.SMTP\_SSL('smtp.gmail.com', 465, context=context) as smtp:

        smtp.login(email\_sender, password)

        smtp.sendmail(email\_sender, to\_email, em.as\_string())

**Código withdraw.py**

import pymysql

import json

import os

import bcrypt

from email\_utils import send\_email

def handler(event, context):

    body = json.loads(event['body'])

    numero\_cuenta = body['numeroCuenta']

    monto = body['monto']

    connection = pymysql.connect(

        host=os.environ['DB\_HOST'],

        user=os.environ['DB\_USER'],

        password=os.environ['DB\_PASSWORD'],

        database=os.environ['DB\_NAME'],

        port=int(os.environ['DB\_PORT'])

    )

    try:

        with connection.cursor() as cursor:

            cursor.execute("SELECT saldo, correoElectronico FROM CuentaBancaria WHERE numeroCuenta = %s", (numero\_cuenta,))

            result = cursor.fetchone()

            if result is None:

                response = {

                    'statusCode': 404,

                    'body': json.dumps({'message': 'Cuenta no encontrada'})

                }

            else:

                saldo = result[0]

                to\_email = result[1]

                if saldo < monto:

                    response = {

                        'statusCode': 400,

                        'body': json.dumps({'message': 'Fondos insuficientes'})

                    }

                else:

                    cursor.execute("UPDATE CuentaBancaria SET saldo = saldo - %s WHERE numeroCuenta = %s", (monto, numero\_cuenta))

                    cursor.execute("INSERT INTO Transaccion (tipo, monto, idCuenta) VALUES ('Retiro', %s, (SELECT id FROM CuentaBancaria WHERE numeroCuenta = %s))", (monto, numero\_cuenta))

                    connection.commit()

                    # Enviar correo electrónico

                    subject = 'Retiro Realizado'

                    email\_body = f'Se ha realizado un retiro de {monto} de la cuenta {numero\_cuenta}.'

                    send\_email(to\_email, subject, email\_body)

                    response = {

                        'statusCode': 200,

                        'body': json.dumps({'message': 'Retiro realizado con éxito'})

                    }

    except Exception as e:

        response = {

            'statusCode': 500,

            'body': json.dumps({'message': str(e)})

        }

    finally:

        connection.close()

    return response

Código atm-api-stacks:

import \* as cdk from '@aws-cdk/core';

import \* as lambda from '@aws-cdk/aws-lambda';

import \* as apigateway from '@aws-cdk/aws-apigateway';

import \* as path from 'path';

export class AtmApiStack extends cdk.Stack {

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

        super(scope, id, props);

        // Dirección de correo electrónico verificada en SES

        const sesVerifiedEmail = 'racuenca11@utpl.edu.ec';

        // Credenciales de correo electrónico

        const emailSender = 'kristylanistek@gmail.com';

        const emailPassword = 'qrnl ldum kpjs plmx';

        // Definir las funciones Lambda

        const depositLambda = new lambda.Function(this, 'DepositFunction', {

            runtime: lambda.Runtime.PYTHON\_3\_8,

            code: lambda.Code.fromAsset(path.join(\_\_dirname, '../lambda-deployment-package.zip')),

            handler: 'deposit.handler',

            environment: {

                DB\_HOST: 'sql3.freemysqlhosting.net',

                DB\_NAME: 'sql3710437',

                DB\_USER: 'sql3710437',

                DB\_PASSWORD: 'M4ZS1SNcx4',

                DB\_PORT: '3306',

                SES\_VERIFIED\_EMAIL: sesVerifiedEmail,

                EMAIL\_SENDER: emailSender,

                EMAIL\_PASSWORD: emailPassword

            }

        });

        const withdrawLambda = new lambda.Function(this, 'WithdrawFunction', {

            runtime: lambda.Runtime.PYTHON\_3\_8,

            code: lambda.Code.fromAsset(path.join(\_\_dirname, '../lambda-deployment-package.zip')),

            handler: 'withdraw.handler',

            environment: {

                DB\_HOST: 'sql3.freemysqlhosting.net',

                DB\_NAME: 'sql3710437',

                DB\_USER: 'sql3710437',

                DB\_PASSWORD: 'M4ZS1SNcx4',

                DB\_PORT: '3306',

                SES\_VERIFIED\_EMAIL: sesVerifiedEmail,

                EMAIL\_SENDER: emailSender,

                EMAIL\_PASSWORD: emailPassword

            }

        });

        const changePinLambda = new lambda.Function(this, 'ChangePinFunction', {

            runtime: lambda.Runtime.PYTHON\_3\_8,

            code: lambda.Code.fromAsset(path.join(\_\_dirname, '../lambda-deployment-package.zip')),

            handler: 'changepin.handler',

            environment: {

                DB\_HOST: 'sql3.freemysqlhosting.net',

                DB\_NAME: 'sql3710437',

                DB\_USER: 'sql3710437',

                DB\_PASSWORD: 'M4ZS1SNcx4',

                DB\_PORT: '3306',

                SES\_VERIFIED\_EMAIL: sesVerifiedEmail,

                EMAIL\_SENDER: emailSender,

                EMAIL\_PASSWORD: emailPassword

            }

        });

        // Definir API Gateway

        const api = new apigateway.RestApi(this, 'atm-api', {

            restApiName: 'ATM Service',

            description: 'This service handles ATM transactions.'

        });

        const depositIntegration = new apigateway.LambdaIntegration(depositLambda);

        const withdrawIntegration = new apigateway.LambdaIntegration(withdrawLambda);

        const changePinIntegration = new apigateway.LambdaIntegration(changePinLambda);

        api.root.addResource('deposit').addMethod('POST', depositIntegration);

        api.root.addResource('withdraw').addMethod('POST', withdrawIntegration);

        api.root.addResource('changepin').addMethod('POST', changePinIntegration);

    }

}

<https://github.com/racuenca11/atm-aws>