

1. Infraestructura

- **Usuario (Action)** Representa a los usuarios finales que interactúan con la aplicación a través de un dispositivo móvil o una computadora.
El canal es el frontend que se comunica con el backend a través de una API, representada por Amazon API Gateway.

- **Amazon API Gateway** Actúa como punto de entrada para las solicitudes que provienen del frontend.
Recibe las solicitudes del usuario y las envía a las funciones AWS Lambda para su procesamiento.

Es una buena práctica usar API Gateway para gestionar las solicitudes, ya que permite manejar autenticaciones, limitaciones de solicitudes, entre otros.

- **AWS Lambda** Aquí hay dos funciones Lambda:

Lambda seguros Voluntarios: Maneja procesos relacionados con seguros voluntarios.

Lambda Rango Plazo tasas: Maneja la lógica de negocio relacionada con plazos y tasas.

Las lambdas son funciones serverless que ejecutan código en respuesta a eventos y escalan automáticamente según la carga.

- **Amazon SQS** Amazon SQS (Simple Queue Service): Sirve para manejar procesamiento en segundo plano.

Las lambdas pueden enviar mensajes a la cola SQS cuando hay tareas que requieren procesamiento asíncrono.

Una vez que los mensajes están en la cola, otras funciones Lambda u otros servicios pueden leer esos mensajes y procesarlos.

- **Amazon Cognito** Se utiliza Amazon Cognito para la autenticación y autorización de los usuarios.

Cognito permite gestionar la identidad de los usuarios, tanto autenticaciones directas como integraciones con proveedores de identidad externos (Google, Facebook, etc.). El frontend se comunica con Cognito para autenticar a los usuarios antes de enviar las solicitudes a la API Gateway.

- **Amazon S3** (Simple Storage Service) es utilizado para almacenar archivos, como imágenes, documentos u otros recursos estáticos que pueden ser accesibles desde el frontend o backend.
- **Amazon EC2** está correctamente configurado como un respaldo o procesamiento para trabajos pesados.
- **Amazon SQS** correctamente actúa como una cola asíncrona para conectar **Lambda** y **EC2**.
- **PostgreSQL** está bien ubicado dentro de la VPC y conectado a **EC2**, pero podrías considerar usar **Amazon RDS** o **Aurora** para optimización adicional.
- **Amazon CloudWatch** para monitorear el desempeño de los componentes, lo cual sería una mejora importante para monitoreo en tiempo real y alertas.

2. Descripción de la Infraestructura en la Nube.

Simulación Local con LocalStack

Para simular este entorno localmente utilizando LocalStack, sigue estos pasos:

I. Instalar LocalStack:

- LocalStack proporciona un entorno local para AWS que simula servicios como S3, DynamoDB, Lambda, API Gateway, etc.
- Instalar LocalStack mediante Docker o pip:

pip install localstack

II. Configurar LocalStack:

- Inicia LocalStack:

localstack start

III. Configurar Servicios Locales:

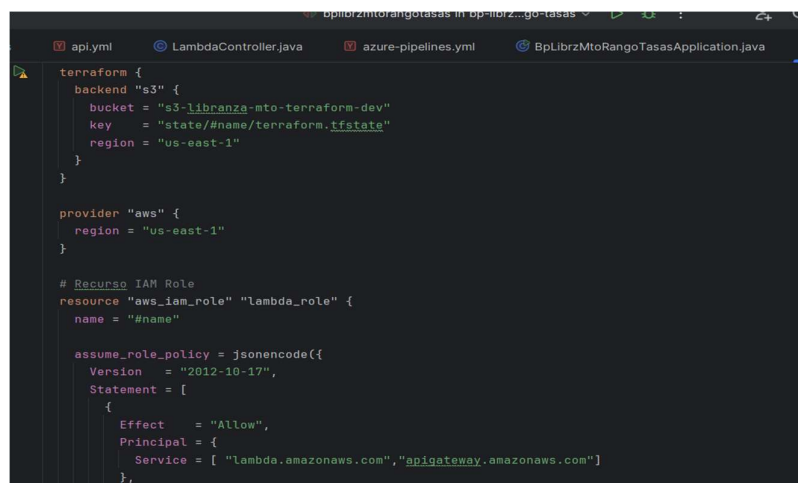
- Frontend: Ejecutar la aplicación web localmente como se haría en un entorno de desarrollo normal.
- Backend:
 - Lambda: Crear funciones Lambda en LocalStack. Instalar AWS CLI y Configurar LocalStack ejecute el comando: `aws configure` Ingresa las siguientes configuraciones:
 - **AWS Access Key ID:** test
 - **AWS Secret Access Key:** test

- **Default region name:** us-east-1 (o tu región preferida)
- **Default output format:** json
- Para usar LocalStack, configura la AWS CLI para apuntar a LocalStack, ejecute: export AWS_ENDPOINT_URL=http://localhost:4566
- **Crear una Función Lambda:** Para crear una función Lambda, primero necesitas definir el código de la función y empaquetarlo en un archivo zip.
- **Crear la Función Lambda en LocalStack:** Usa la AWS CLI para crear la función Lambda en LocalStack:

```
aws lambda create-function \ --function-name bp-librz-mto-rango-
tasas \ --runtime java17 \ --role
arn:aws:iam::000000000000:role/irrelevant \ --handler
com.example.HelloWorldHandler::handleRequest \ --zip-file
fileb://target/lambda-java-example-1.0-SNAPSHOT.jar \ --endpoint-
url $AWS_ENDPOINT_UR
```

- API Gateway: Configurar APIs locales que se enruten a las funciones Lambda.
- Base de Datos:
 - RDS: LocalStack no simula RDS completamente, pero puedes usar una base de datos local PostgreSQL para simular.
- Autenticación: Configurar servicios como Cognito si es necesario, o simular autenticación de manera alternativa.
- Pruebas y Desarrollo:
 - Usar las herramientas de AWS CLI o SDKs configurados para apuntar a LocalStack.
 - Realizar pruebas de integración y asegurarte de que los servicios interactúan correctamente en tu entorno local.

3. Prácticas DevOps:



```

terraform {
  backend "s3" {
    bucket = "s3-libranza-mto-terraform-dev"
    key    = "state/#name/terraform.tfstate"
    region = "us-east-1"
  }
}

provider "aws" {
  region = "us-east-1"
}

# Recurso IAM Role
resource "aws_iam_role" "lambda_role" {
  name = "#name"

  assume_role_policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Effect = "Allow",
        Principal = {
          Service = [ "lambda.amazonaws.com", "apigateway.amazonaws.com" ]
        }
      }
    ]
  })
}

```

En los repositorios de los micros encontraran un archivo que es el main.ts utilizando **Terraform**, un ejemplo de **canalización (pipeline)** en términos de la infraestructura de despliegue. Aprovisionamiento automatizado de recursos en la nube (como servicios de AWS) a través de código, facilitando la entrega continua y la administración de infraestructuras.

Desglose del ejemplo:

1. Backend en S3:

- La sección backend "s3" define el almacenamiento remoto del estado de Terraform en un bucket de **Amazon S3**. Este estado mantiene la información sobre los recursos que Terraform está gestionando.

2. Proveedor AWS:

- La configuración de provider "aws" indica que los recursos de Terraform se gestionarán en AWS en la región us-east-1.

3. Recursos IAM:

- Se crea un **IAM Role** (aws_iam_role) para que **Lambda** y **API Gateway** puedan asumir ciertos permisos.
- Luego, se adjuntan varias políticas (aws_iam_role_policy_attachment) al rol IAM, dándole acceso a diferentes servicios de AWS como **EC2**, **CloudWatch Logs**, **Secrets Manager**, y **SQS**.

4. Políticas Personalizadas:

- Definición de políticas IAM personalizadas como la que permite acceso a **Secrets Manager** (aws_iam_policy).

5. Lambda Function:

- La función Lambda (aws_lambda_function) está configurada para ejecutarse en un entorno de **Java 17**, con acceso a una **VPC** utilizando subnets y grupos de seguridad.
- Se establecen variables de entorno y un tiempo de ejecución de 60 segundos.

6. API Gateway:

- Se define un API Gateway (aws_api_gateway_rest_api) que será utilizado para exponer la funcionalidad de la Lambda.