1. Configuración de la base de datos en AWS

    - Se creó la base de datos utilizando la opción de "creación estándar" en Amazon RDS.
    - Se seleccionó el motor MySQL y se asignó el nombre "efrouting" al servidor de la base de datos.
    - Se generó una contraseña para el usuario maestro.
    - La base de datos se asoció a una VPC para establecer reglas de entrada y salida por el puerto 3306.
    - Se configuró la base de datos para ser accesible públicamente desde el localhost con fines de prueba.

2. Creación de la estructura de la base de datos

    - Se accedió a la RDS a través de linea de comando:

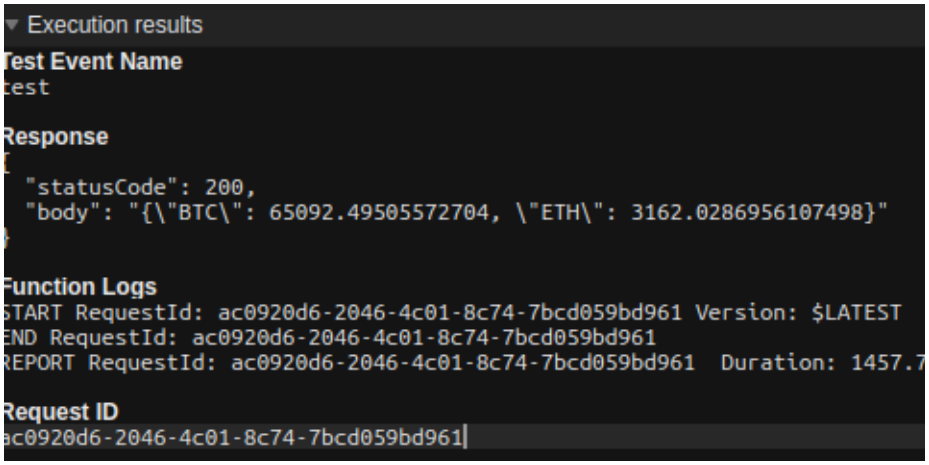
    ```
    mysql -h efrouting.cuuo9scbn0zr.us-east-2.rds.amazonaws.com -u admin -p
    ```

    - Se creó una nueva base de datos denominada "Test".
    - Dentro de la base de datos "Test", se creó una tabla llamada "precios" utilizando los siguientes comandos SQL:

    ```
    CREATE TABLE precios (
        id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
        symbol VARCHAR(10) NOT NULL,
        precio DECIMAL(18, 8) NOT NULL,
        timestamp INT NOT NULL
    );
    ```

3. Integración con la API CoinMarketCAP
    - Se realizó el registro en la página web de CoinMarketCap para obtener acceso a la API.
    - Se desarrolló un script en Python para interactuar con la API y obtener los precios actuales de Bitcoin (BTC) y Ethereum (ETH).
    - El código de esta implementación fue:

    ```python
    import os
    import pymysql
    import requests
    import time
    import json

    def obtener_precio(symbol):
        url = f"https://pro-api.coinmarketcap.com/v1/cryptocurrency/quotes/latest"
        parametros = {
            'symbol': symbol,
            'convert': 'USD'
        }
        headers = {
            'X-CMC_PRO_API_KEY': os.getenv('API')
        }
        respuesta = requests.get(url, headers=headers, params=parametros)
        data = respuesta.json()
        return data['data'][symbol]['quote']['USD']['price']
    ```



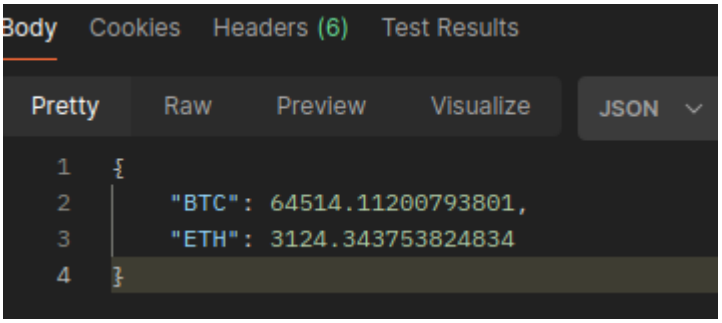
4. Inserción de datos en DB
    - Se hizo uso de la biblioteca PyMySQL
    - Se hizo uso de variables de entorno para insertar los valores
    - La inserción de datos se hizo del siguiente modo:

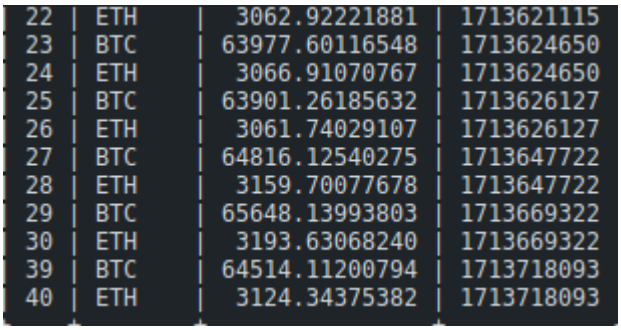
```python
def almacenar_precio(symbol, precio):
    conn = pymysql.connect(
        host=os.getenv('MySQL_H'),
        user=os.getenv('MySQL_U'),
        password=os.getenv('MySQL_P'),
        database=os.getenv('MySQL_DB')
    )
    cursor = conn.cursor()

    timestamp = int(time.time())
    cursor.execute("INSERT INTO precios (symbol, precio, timestamp) VALUES (%s, %s, %s)", (symbol, precio, timestamp))

    conn.commit()
    conn.close()
```

De esta forma se dio creación a la infraestructura básica para una gestión de datos de criptomonedas en AWS

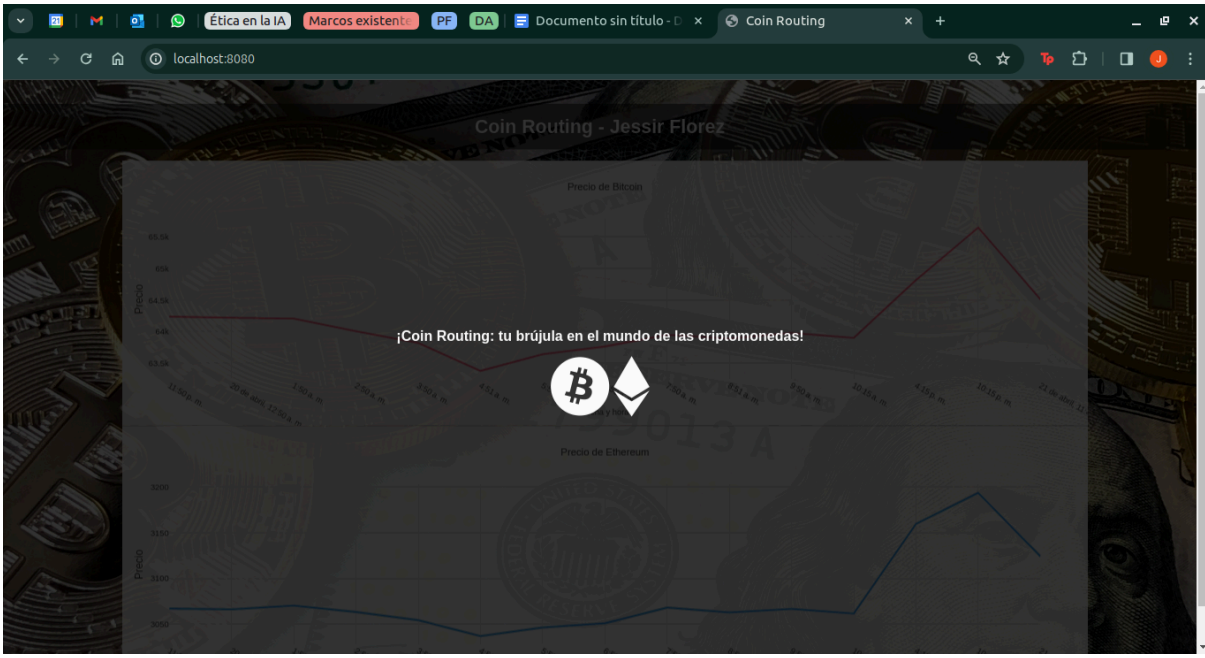
5. Creación y configuración de la Lambda
    - Se desarrolló una función Lambda desde cero con nombre “test_efrouting” usando Python 3.12
    - Se dio creación a un nuevo rol con permisos para acceder a la RDS
    - Se cargaron las librerias “request” y “PyMySQL”
    - Se montó sobre un archivo .zip y se hizo deploy a la Lambda
    - Seguidamente se configuraron las variables de entorno

6. Prueba de la función Lambda
    - Se realizó la prueba de si se estaban devolviendo los datos en una salida .JSON
    - Se hizo uso de la herramienta POSTMAN para verificar este funcionamiento

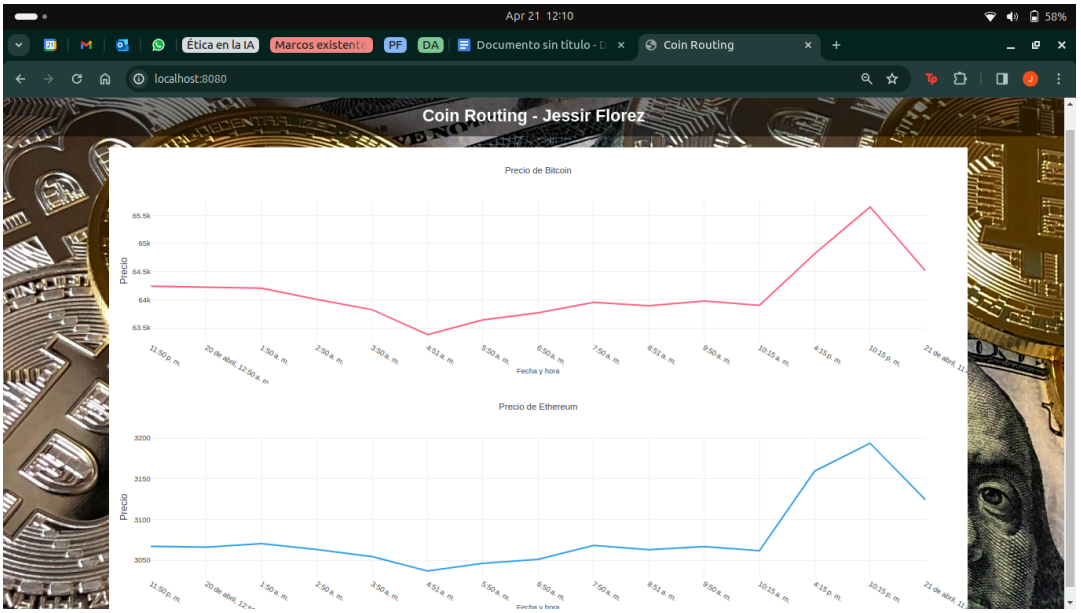


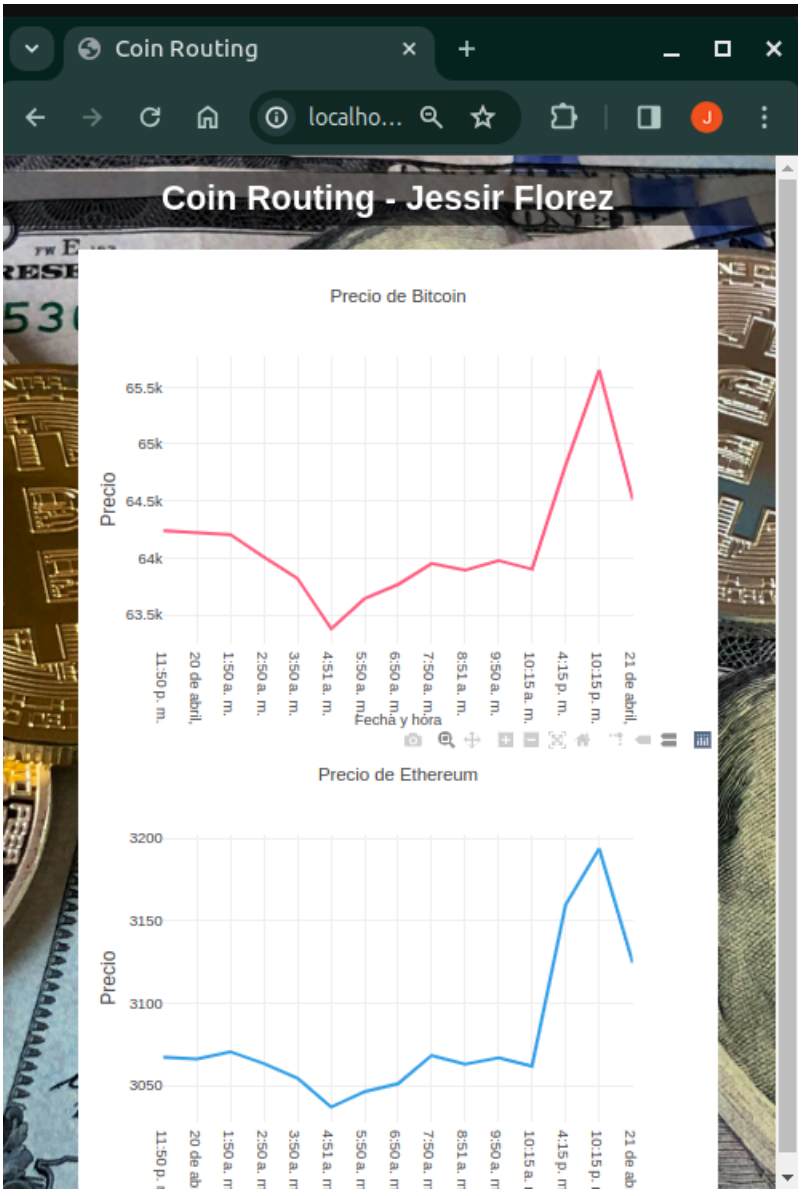


7. Construcción del contenedor Docker
    - Se creó una página web utilizando Node.js, que consta de archivos HTML, CSS(web responsive) y JavaScript.



Jessir Daniel Florez Hamburger - Prueba de desarrollador Full Stack





- Se creó un archivo Dockerfile para configurar el contenedor

```
# Usamos una imagen base de Node.js
FROM node:14

# Creamos un directorio para la aplicación
WORKDIR /usr/src/app

# Copiamos el archivo de dependencias
COPY package*.json ./

# Instalamos las dependencias
RUN npm install

# Copiamos el resto del código de la aplicación
COPY . .

# Exponemos el puerto que usa nuestra aplicación
EXPOSE 80

# Comando para iniciar la aplicación
CMD [ "node", "server.js" ]
```

  - Se construyó la imagen de Docker con el siguiente comando: ***docker build -t coin_routing .***
  - Se ejecutó la imagen del contenedor localmente con el siguiente comando: ***docker run -p 80:8080 coin_routing***
  - Seguidamente se cargó la imagen en el ECR (Elastic Container Registry) de la siguiente forma: ***aws ecr create-repository --repository-name jf***

    aws ecr get-login-password --region us-east-2 | docker login --username AWS --password-stdin 791976247251.dkr.ecr.us-east-2.amazonaws.com

    docker tag coin_routing:latest ***791976247251.dkr.ecr.us-east-2.amazonaws.com/jf:latest***

    docker push ***791976247251.dkr.ecr.us-east-2.amazonaws.com/jf:latest***

8. Configuración en AWS
   - Se configuró un nuevo usuario en AWS con los permisos necesarios para acceder a los recursos creados previamente

- Se realiza al configuración de las claves de acceso de aws configure
- Se definió el repositorio y se subio la imagen
- Se procedió a crear la Task Definition
- Seguidamente se creó el Cluster
- Finalmente se vinculó el servicio entré el cluster y la task definition

Es en este último ítem donde se presentan inconvenientes, el servicio nunca termina de ejecutarse y el link DNS brindado por el Load Balancer no redirecciona a ninguna página, se intenta solucionar otorgando permisos de administrador a la ECS, se realiza un nuevo repositorio en DockerHub para probar el push de la image desde otro medio, se cambia el puerto en el archivo sever.JS, se aplica redireccionamiento de puertos y demás pruebas para verificar el funcionamiento, sin embargo el error persiste, se verifica en los logs de la ECS que la conexión con la RDS se establece, por lo cual se estima que la image está bien diseñada, sin embargo, no se ha logrado conseguir solución al problema, se seguirá trabajando y se enviará un nuevo correo con la corrección de este problema en dado caso sea encontrada la falla