DevOpsProyecto Final

Línea corta

Carlos Rodriguez

029900574

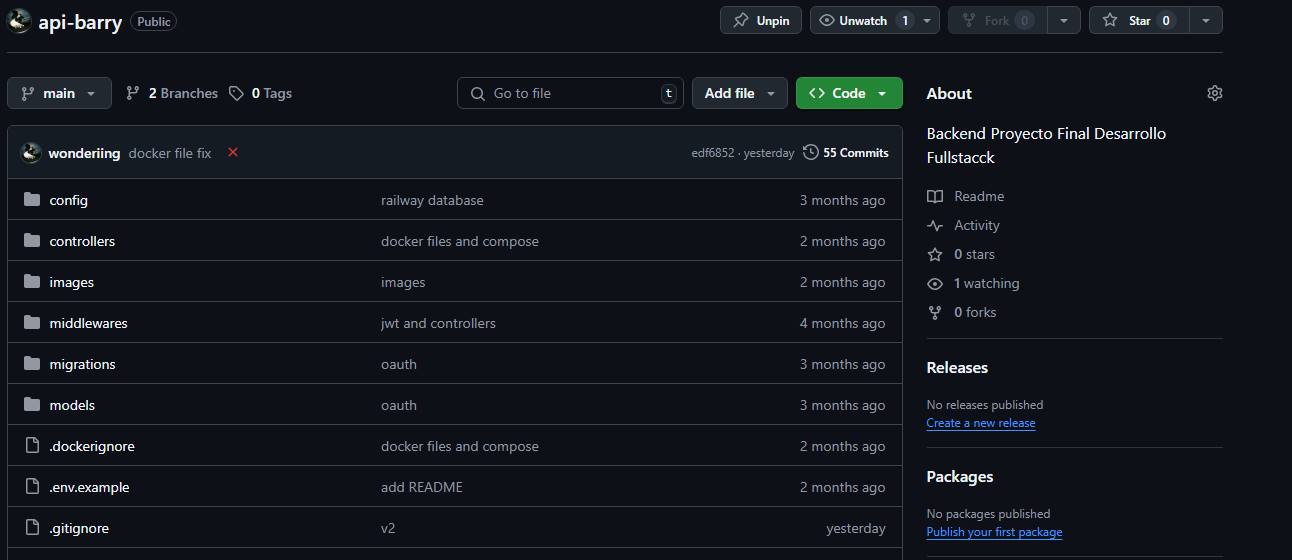
**Introduccion**

Este proyecto fue un reto interesante debido a que las instrucciones simplemente parecian “cosas que podrías hacer en DevOps”, en lugar de un proyecto cohesivo, lo que me hacia pensar que solo era crear servicios al azar sin tener nada conectado entre si. En lugar de simplemente seguir las instrucciones, decidí tratar de “simular” un entorno funcional que reflejara un despliegue real de software, integrando servicios de AWS como lo es Lambda, EC2 CloudFormation etc y aplicando principios de DevOps.

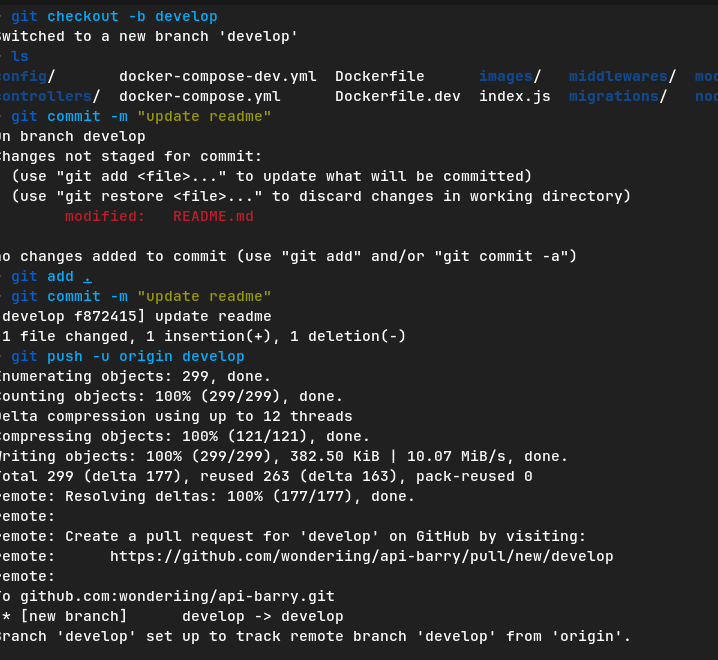
A pesar de que algunos puntos del proyecto no tenían sentido en el contexto de lo que queria hacer , como el uso de DynamoDB, que fue reemplazado por RDS y el hecho de que no segui las instrucciones al pie de la letra traté de cumplir con todos los puntos del proyecto y, sobre todo, aprender a integrar estos servicios en un flujo de trabajo.

**1 - Repositorios de Github:**

1. mkdir api-barry
2. git init - inicializa el repositorio
3. git add . - añade todos los archivos a mi repo local
4. git commit -m - Tiramos el primer commit
5. git branch -b - es para crear una rama nueva llamada develop
6. git push -u origin main - Sube todo al github

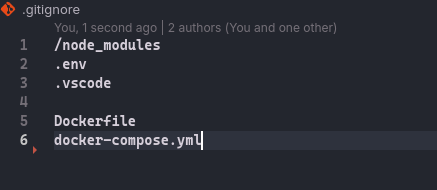
<https://github.com/wonderiing/api-barry>

Pull requests y merges de las 2 ramas:

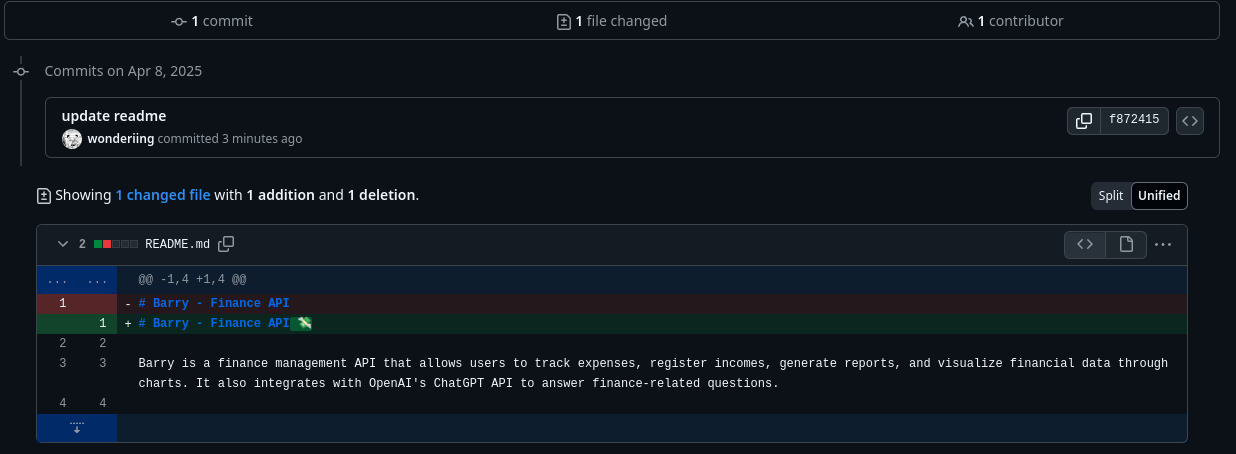


**Configuracion de .gitignore**

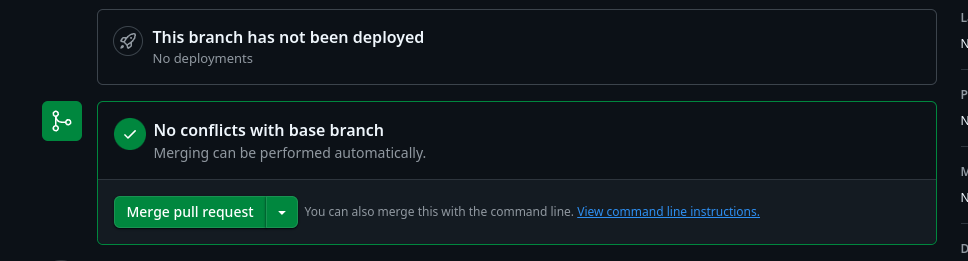
En este archivo se declaran los archivos/directorios que no quieres que sean subidos al repositorio



Este es el cambio de la pull request:



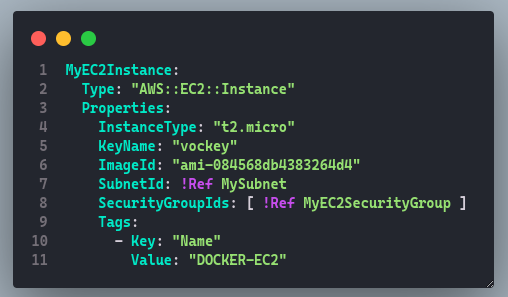
Merge:

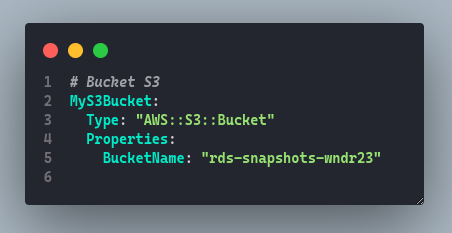


**2 - Creacion del stack con CloudFormation:**

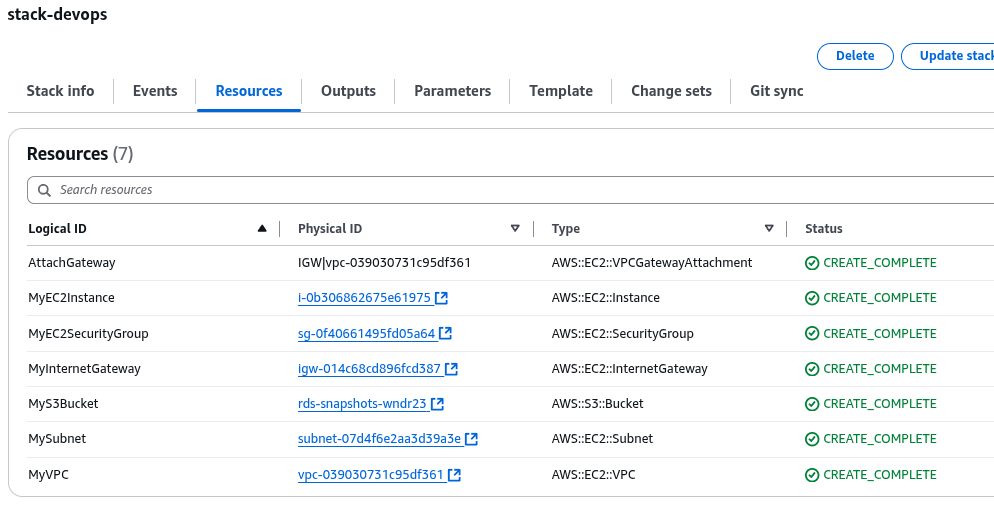
La plantilla es bastante grande, incluye una VPC con sus subredes, gateway , tablas de rutas, un EC2 Ubuntu con su security group y un bucket S3. Algunas Partes de la plantilla:







Recursos completo del Stack:



# Mapa de recursos de la VPC



# 

# **3 - Configuración del Entorno Linux**

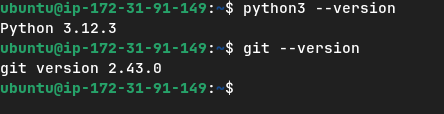
Conexion:

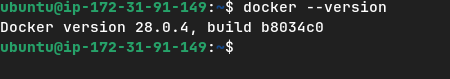


Actualizando paquetes



Paquetes esenciales instalados y actualizados

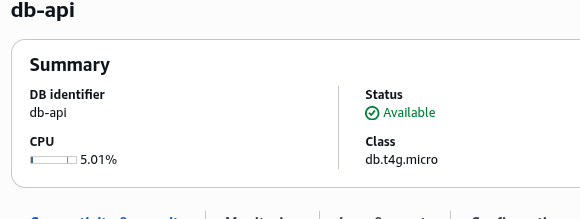




**4 - Dockerizacion, Despliegue de la app y base de datos**

Aqui despliegue todos los servicios necesarios para hacer funcionar la app completa, usando EC2, Docker, Nginx, AWS RDS etc..

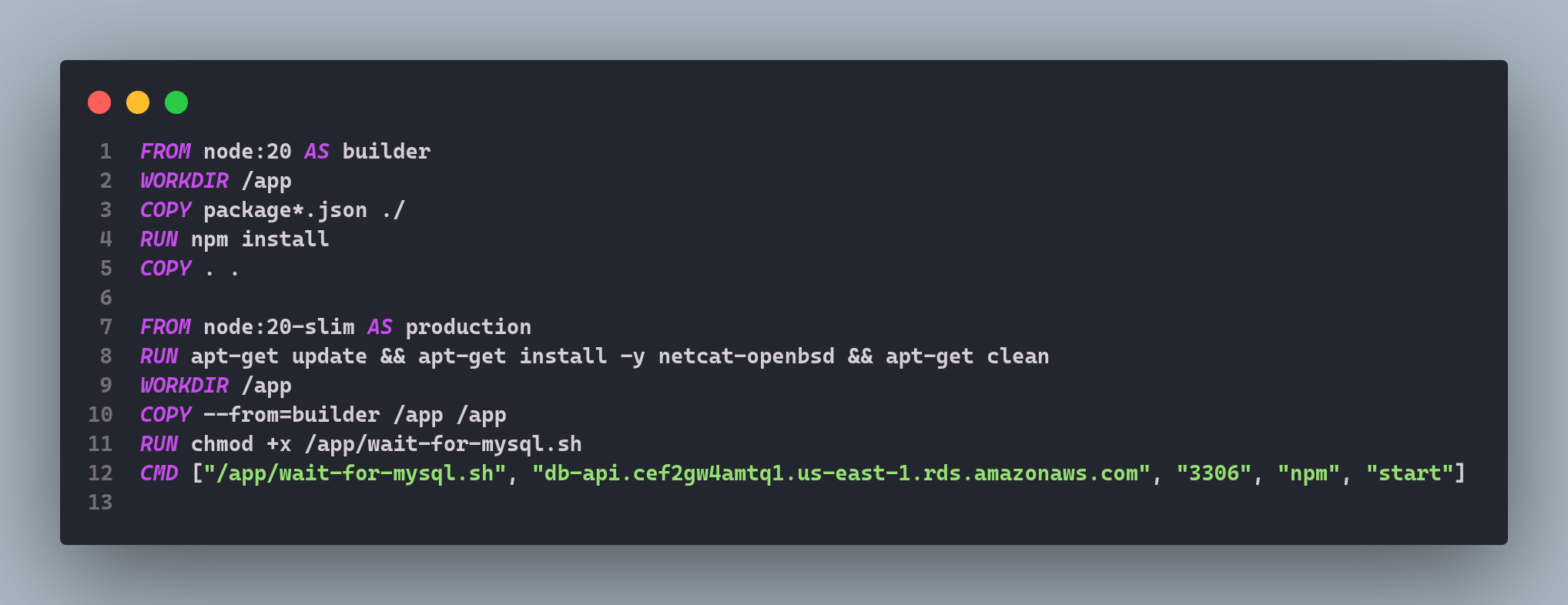
En este caso en lugar de usar DynamoDB utilize una AWS RDS por que no tenia mucho sentido y requiria usar más de una sola tabla para la app que queria desplegar. La base de datos es relacional y MySQL



Abrimos el puerto 3306 para permitir conexiones la base de datos



Dockerizar la app con multi-stage. En este caso dockerize una API hecha en express



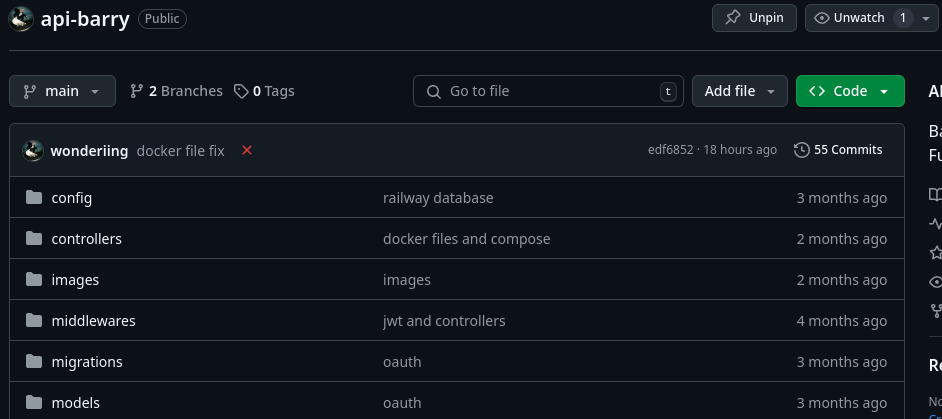
Despues en el EC2 que creamos con CloudFormation nos conectamos y instalamos docker





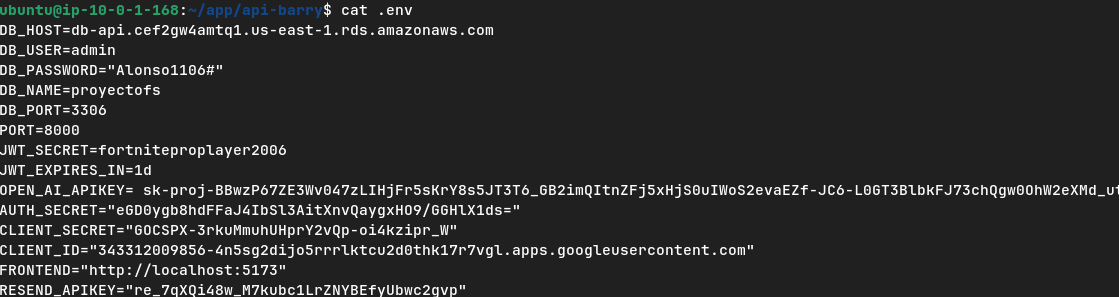
Clonamos el repositorio anteriormente subida a Github





Configurar las variables de entorno. En las variables de entorno colocamos las credenciales de la base de datos de AWS RDS entre otras cosas



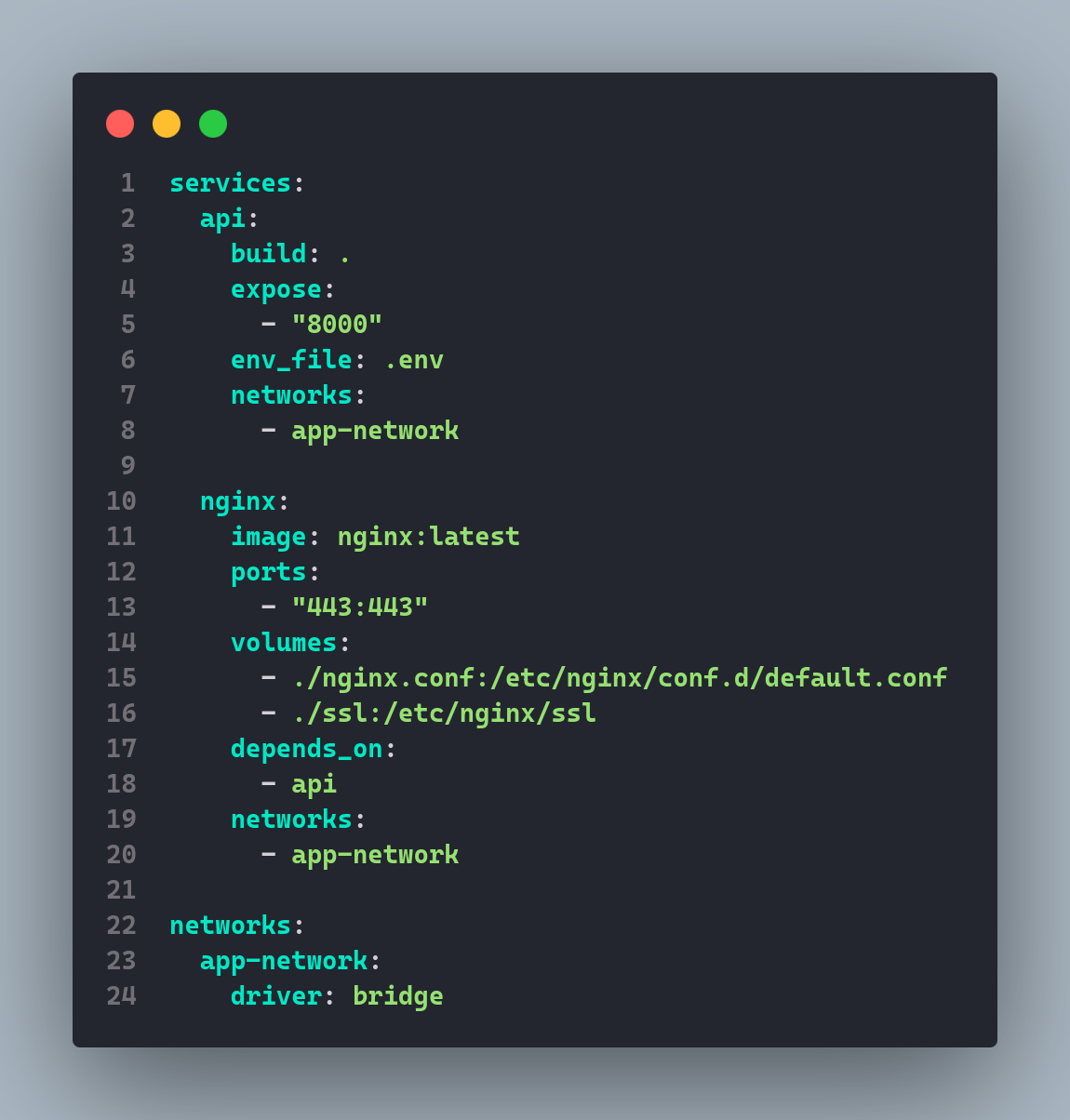


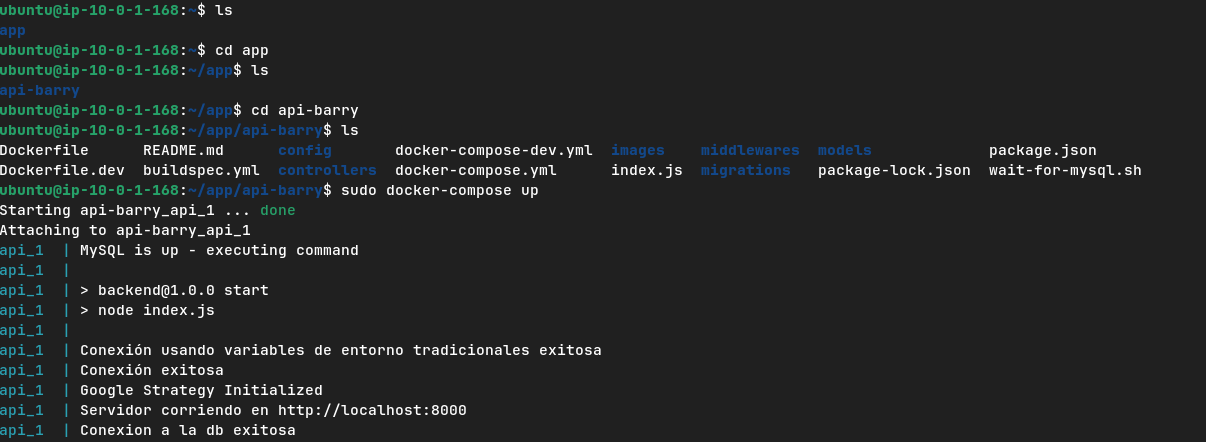
Configuracion del HTTPS para más seguridad:

Nginx escuchara todo el trafico del puerto 443(HTTPS). El puerto 8000 de mi api no es accesible desde fuera, es gracias al Docker Compose que nginx se comunica internamente con mi api y los redirige a ella

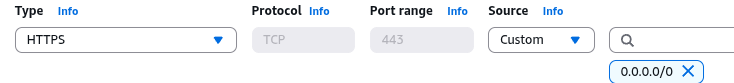


DockerCompose

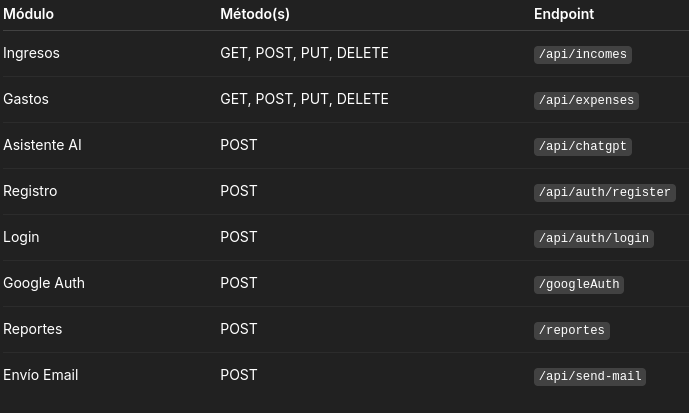


Ejecucion del DockerFIle 

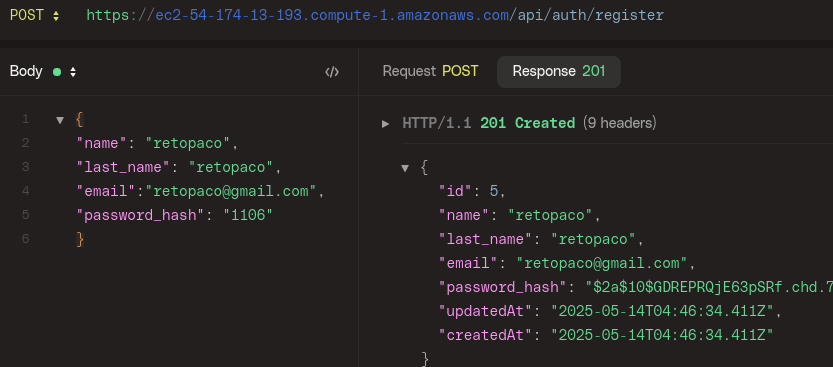
Permitir el trafico desde todos lados al EC2:

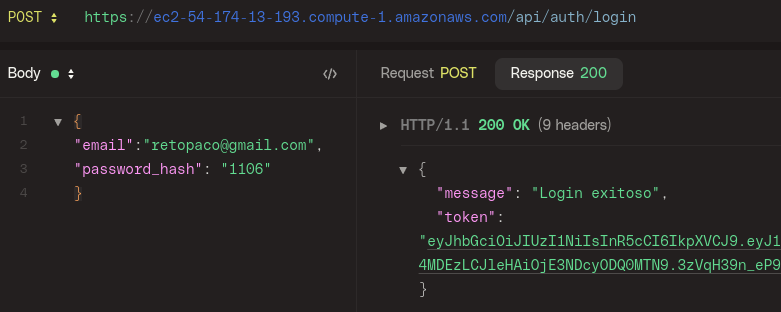


Endpoints y pruebas:



Link: <https://ec2-54-174-13-193.compute-1.amazonaws.com/>

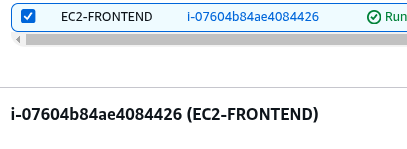




**5 - Despliegue del FrontEnd para consumir la API**

Desplegue un FrontEnd para consumir la API que desplegue anteriormente.

Cree un EC2 Ubuntu

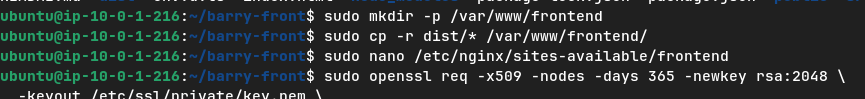


En el EC2 instale nodejs y clone el repositorio del FrontEnd

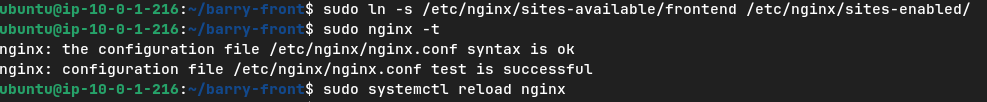




Configure Nginx para los certificados SSL

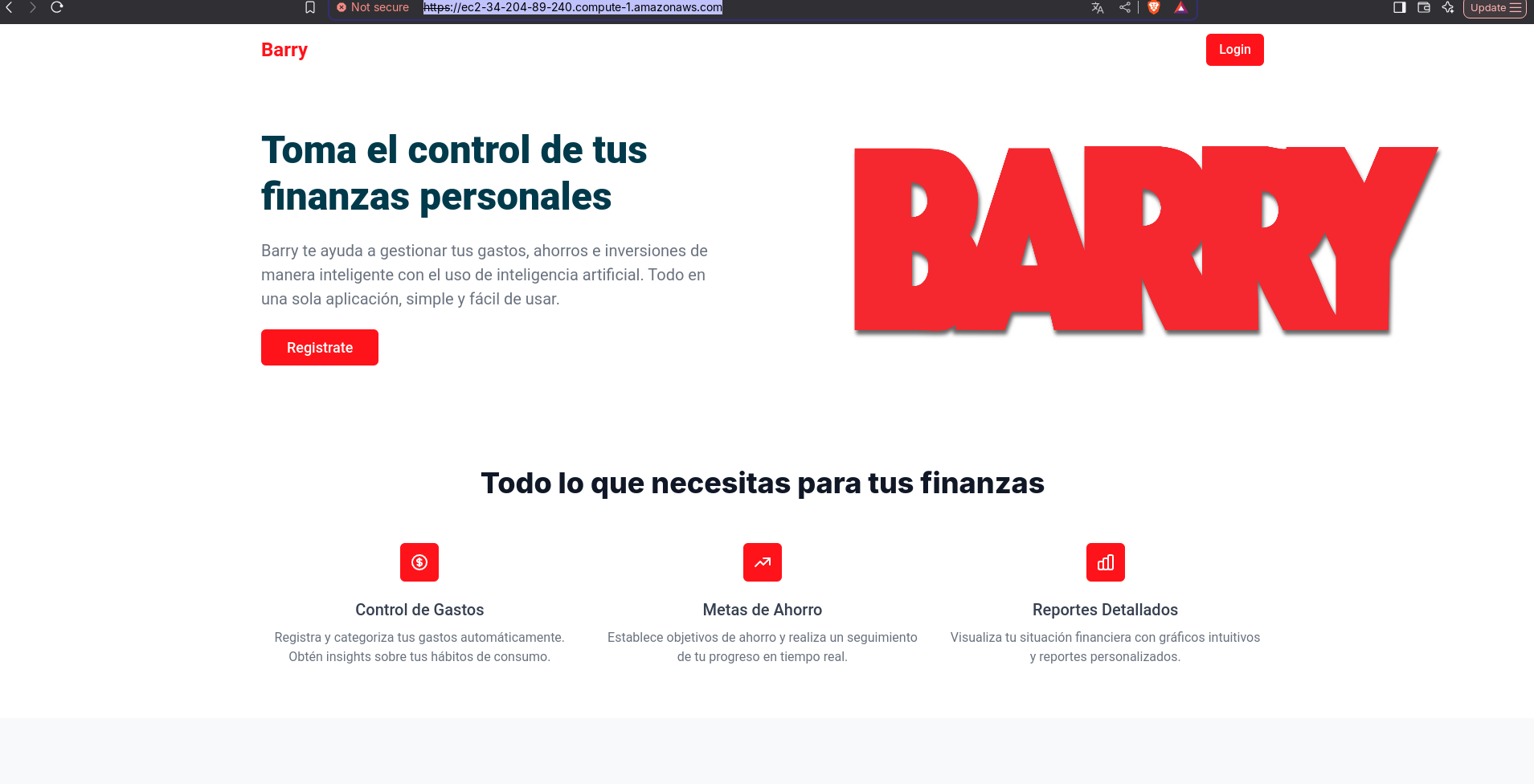


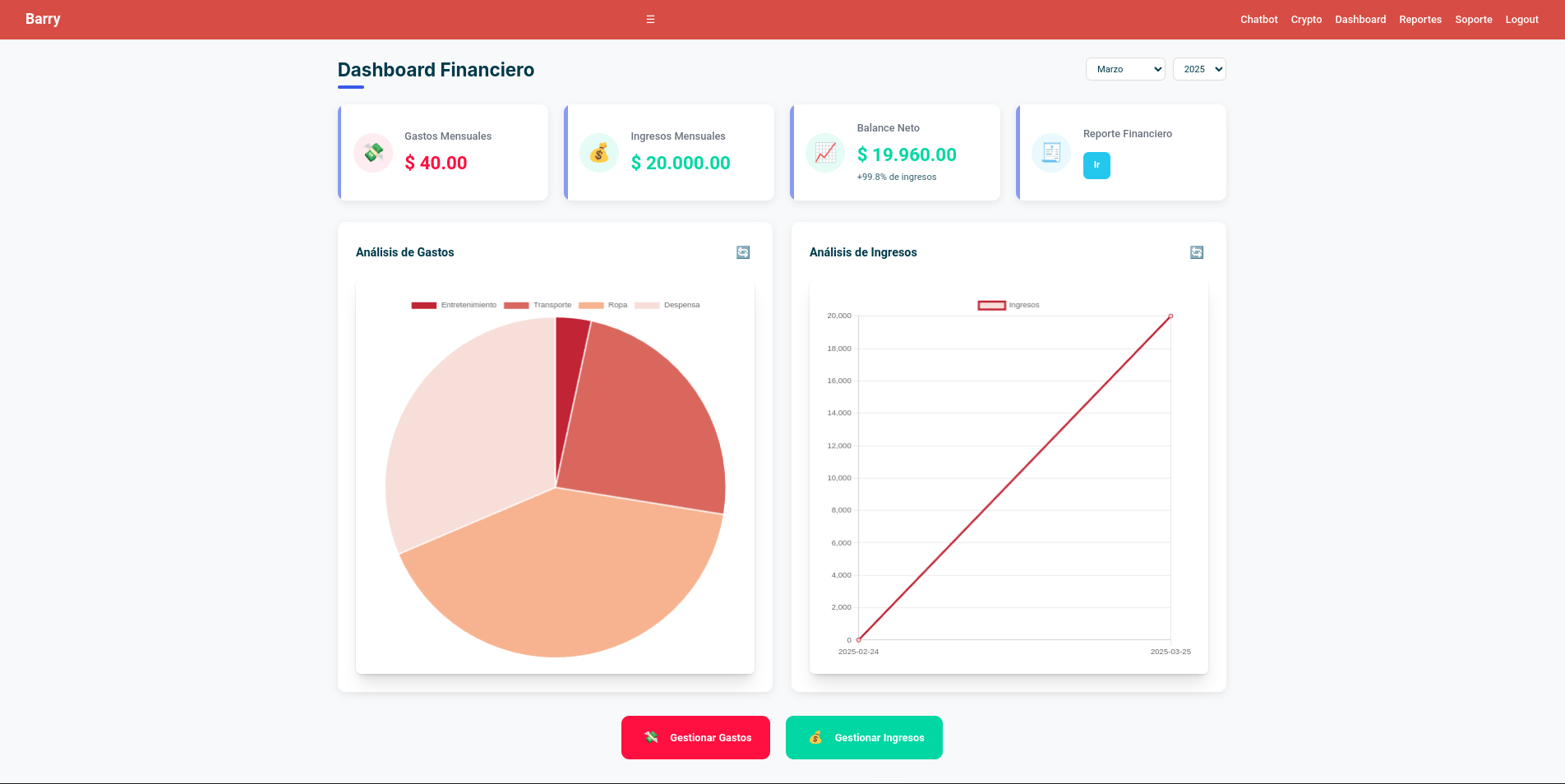
Corri el FrontEnd



Link y Pruebas:

<https://ec2-34-204-89-240.compute-1.amazonaws.com/>

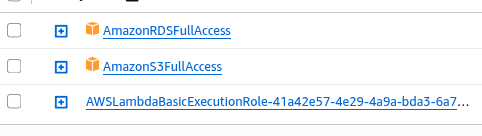
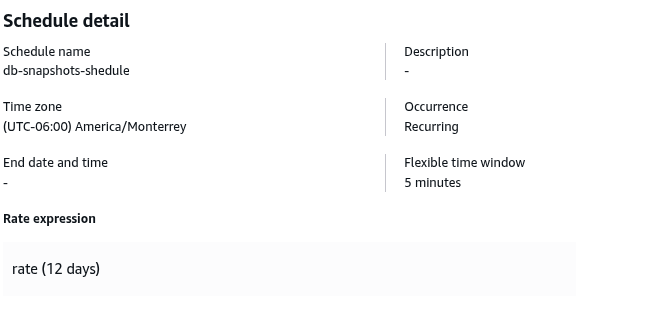
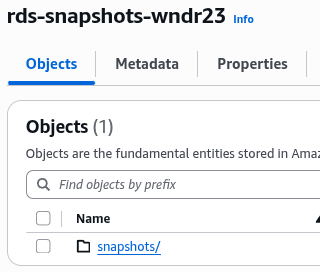
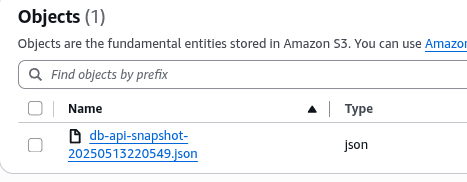




**6 - Arquitectura de microservicios Lambda con EventBridge**

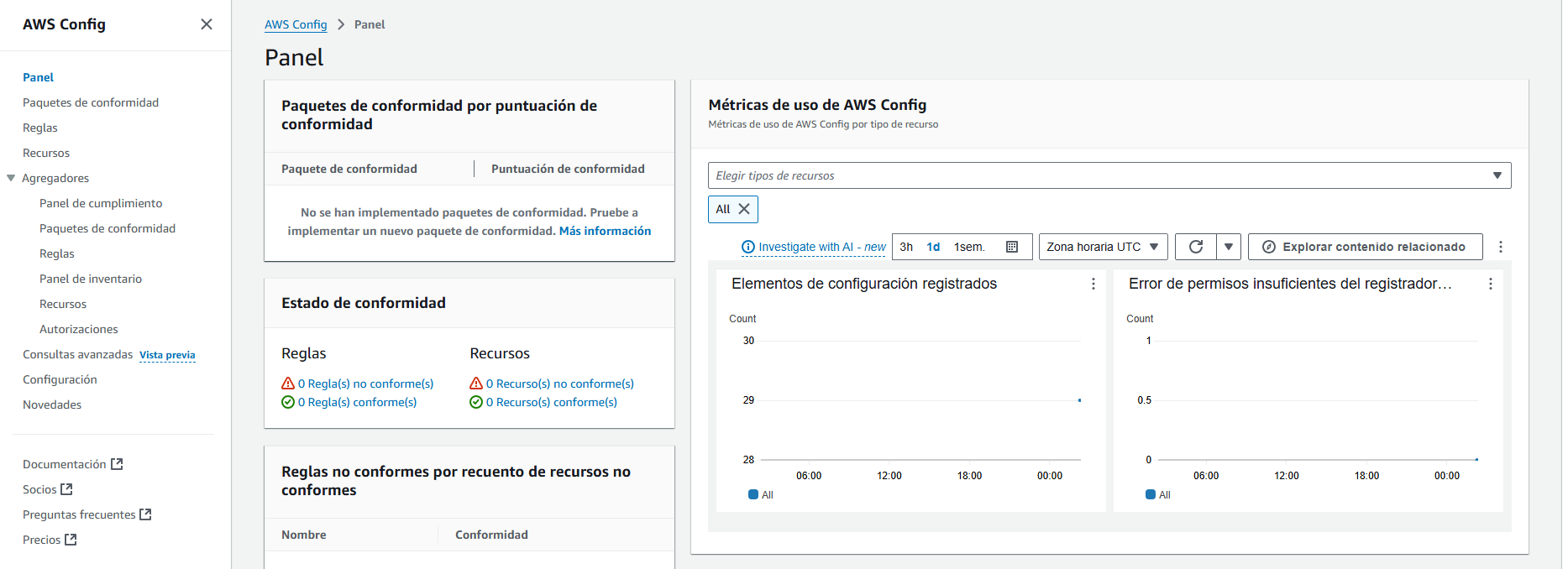
La instruccion pedia crear una Lambda y retornar un Hello World, al no tener mucho sentido con la app que yo despliegue decidi cambiarlo

Creamos una Lambda que hace snapshots a la base de datos que usamos en la API y guarda las snapshots en un bucket S3. Con EventBridge la configuramos para que se ejecute cada cierto tiempo

* Creamos la Lambda, no use ningun comando para subirla ya que la cree directamente desde la consola, este es el codigo:
* 
* Permisos para la Lambda
* 
* EventBridge para la ejectucion de la Lambda. Se ejecutara cada 12 dias pero para probarlo lo coloque con 1 minuto
* 
* Prueba de la snapshot guardad en el bucket:
* 
* 

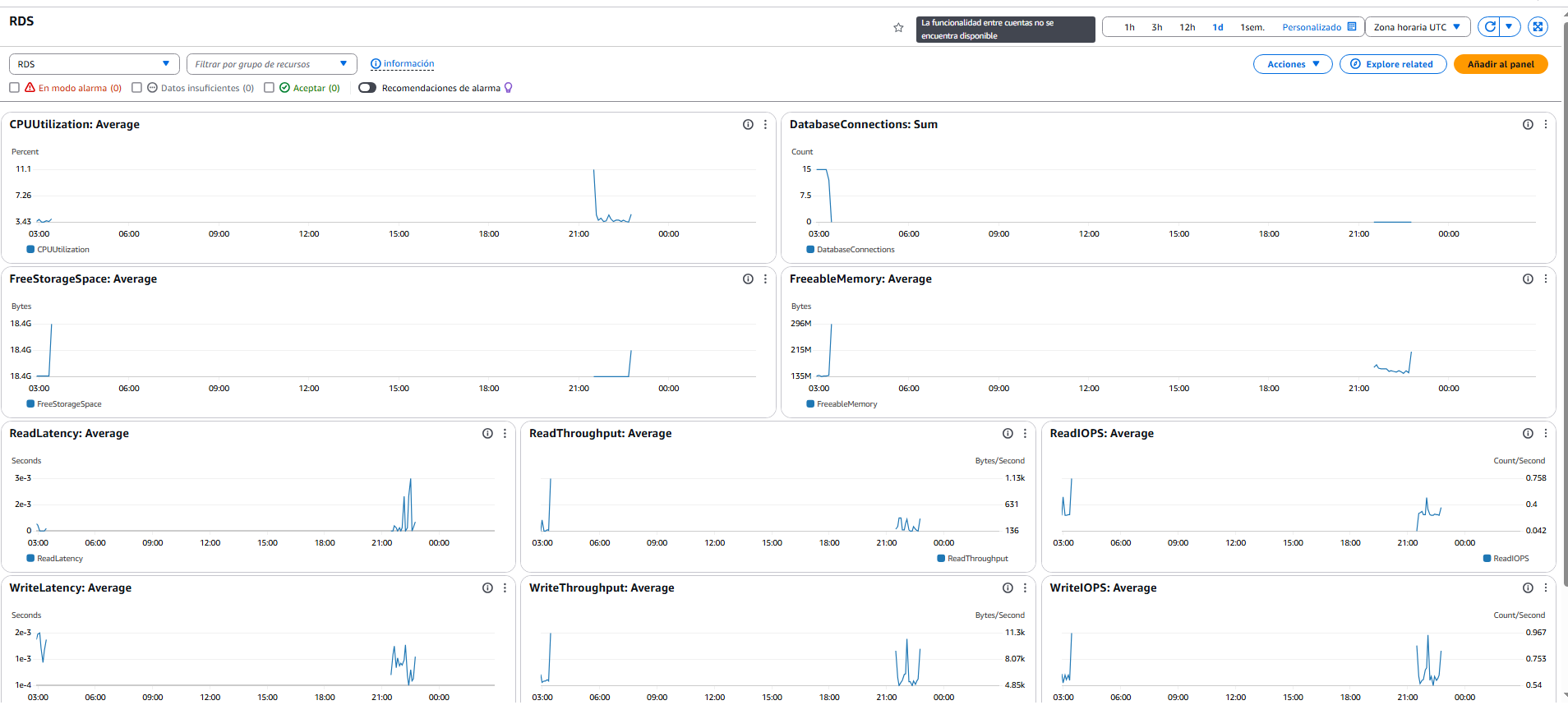
**7 - Monitoreo y registro de eventos:**

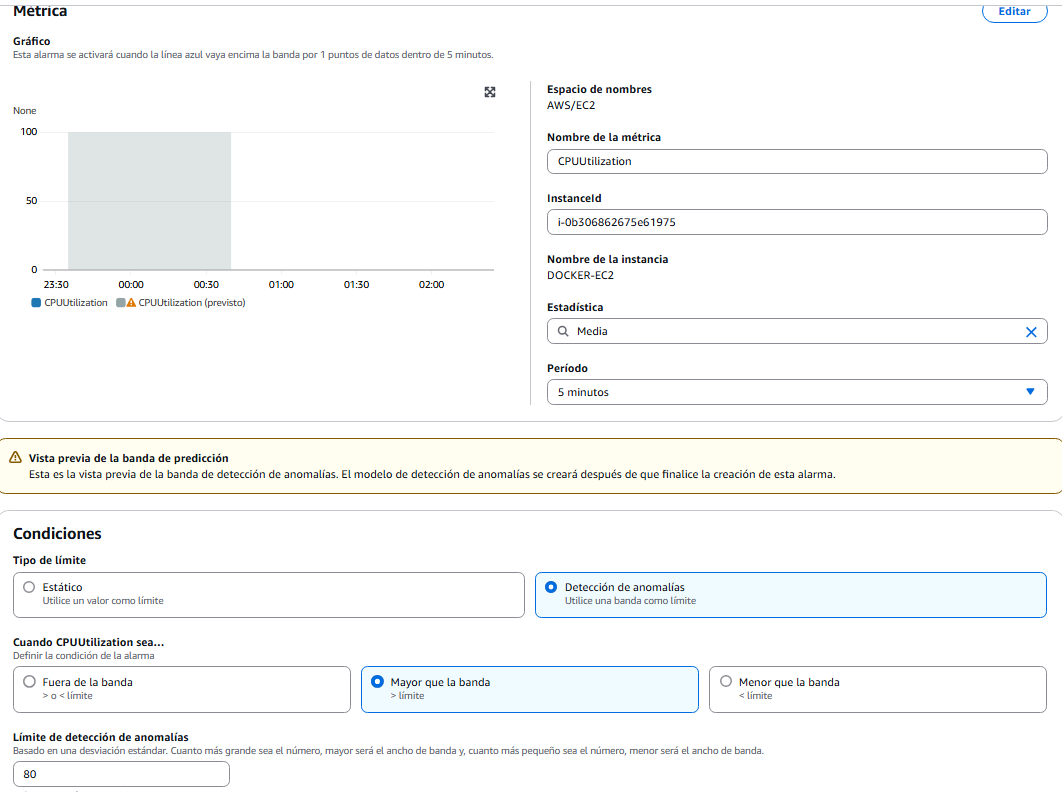
AWS Config:

****

Metricas y alarma de CloudWatch para el uso excesivo del CPU en mi instancia EC2

Metricas de la base de datos



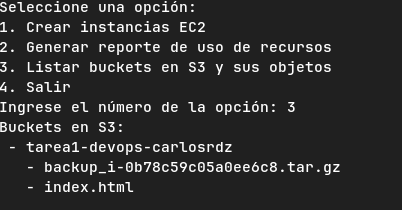


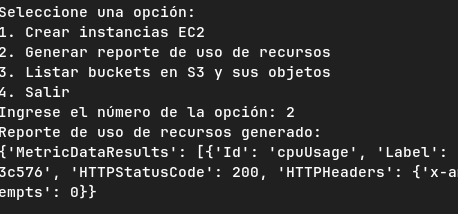
**8 - Script en Python para Automatizar recursos**

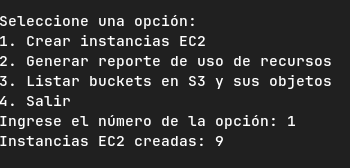
Script para crear y gestionar recursos aws:



Pruebas:





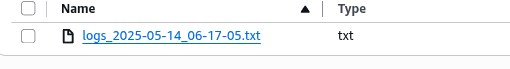


**9 - Logs**

Cree un script de python que recibe los logs de mi API y los sube a un S3



Prueba de log en s3:



Tambien cree un grupo de Logs usando CloudWatch Logs que registra los logs de la funcion lambda que genera snapshots a la base de datos

