TRABAJO PRÁCTICO PROGRAMACIÓN AVANZADA PARA GRANDES VOLÚMENES DE DATOS

Integrantes: Martina De Angelis y Santiago Lares Harbin

Como primer paso, configuramos IAM en AWS para asegurar que los servicios puedan interactuar adecuadamente entre sí. Posteriormente, preparamos los datos crudos, los cuales fueron necesarios para nuestro DAG, y los almacenamos en un bucket de S3.

Luego procedimos a instalar Airflow en nuestra consola, dentro de un entorno virtual específico. Para verificar el correcto funcionamiento del DAG, realizamos pruebas locales antes de su implementación en una instancia EC2. Durante este proceso, enfrentamos varios retos relacionados con la instalación y configuración de Airflow, así como con la ejecución eficiente de los dags en el entorno de Airflow. Tras resolver estos problemas y confirmar su funcionamiento local, migramos el sistema a la instancia EC2. Esto implicó cambiar la versión preinstalada de Python por una versión anterior compatible con Airflow, copiar la carpeta de dags y reinstalar y reconfigurar Airflow en EC2. Además, tuvimos que ajustar la configuración de EC2 para permitir el tráfico a través del puerto 8080, utilizado por Airflow.

Finalmente, configuramos una API para visualizar los resultados. En comparación con Airflow, la configuración local de la API fue relativamente más sencilla, ya que no requería múltiples comandos. Además, el desarrollo del código para la API fue más directo, dado que las consultas SQL empleadas no requerían una codificación extensa.

Borrador simple para pasar por Chat GPT: La API fue diseñada con una arquitectura en modulos para asemejarse lo mas posible al mundo real en un entorno de produccion, lo que creemos que es uno de los objetivos de la materia. Luego de desarrollada la aplicación y cerciorarse de que corra correctamente localmente, se instalo Docker Desktop, se instalaron los paquetes necesarios. Se creo el archivo Dockerfile con los comandos necesarios, listando todas las dependencias del ambiente virtual en un archivo app\_requirements.txt . Seguido de esto, se creo la imagen de la aplicación en un container Docker. Hubo un problema con la dependencias ya que se uso originalmente una copia del venv usado en un proyecto de trabajo de uno de los integrantes, por lo que se tuvo que desinstalar todos los paquetes del venv y pasar a instalar los necesarios uno por uno. Superado esto, se corrio la imagen localemnte y se verifico despues de algunos intentos de que esta corria correctamente. Se creo una instancia de ECS y un ECR. Configurar estas instancias tampoco fue sencilla dada las configuraciones de claves que los usuarios IAM debian tener, las politicas de accesos ya determinadas no funcionaban correctamente y se tuvo que especificar manualmente en un JSON los accesos necesarios de la politica del usuario. Luego se pudo crear un ECR, conseguir el token de acceso temporario y poder pushear la imagen al ECR para que corra dentro de App Runner. Finalmente se pudo levantar la instancia de App Runner.

Otro problema: la imagen se habia construido por default en una arquitectura que no permitia que uvicorn funcionara. Hasta que se descubrio que era este el problema por el cual el servicio de App Runner no se lograba construir, pasaron varios intentos.   
  
Pasos: construir imagen de docker. Crear repositorio de ECR. Pushear Imagen a ECR. Levantar imagen en App Runner.