MLOps
Universidad Torcuato Di Tella

Trabajo Práctico Final

2024

Alumna: Eugenia Tellechea, Juana Masanet,
Micaela Beller, Florencia Messidoro Saavedra

Pasos de implementación

Comenzamos configurando los servicios de AWS. Creamos un bucket en S3 para almacenar datos, configuramos una instancia EC2 para instalar y ejecutar Apache Airflow, y configuramos una base de datos PostgreSQL utilizando Amazon RDS. Este paso incluyó la obtención del endpoint de RDS para conectarlo con los demás servicios y la configuración de las políticas de seguridad para garantizar que los servicios se comunicaran correctamente entre sí.

En EC2 instalamos Apache Airflow dentro de un entorno virtual. Configuramos el DAG necesario para el pipeline, integrándose con el bucket S3 para leer datos y con RDS para procesarlos. Dentro del pipeline definimos tareas para realizar los cálculos pedidos. Testeamos las funciones de a una para asegurarnos que su implementación sea correcta y corregir errores encontrados en cada paso.

Configuramos el repositorio en GitHub con la estructura necesaria para alojar las database, main, models, un archivo para los modelos (TopProduct, TopCTR, y un modelo para devolver las recomendaciones) y el Dockerfile.

Una vez finalizados, diseñamos la API utilizando FastAPI. Esta API fue empaquetada en un contenedor Docker, subimos la imagen Docker construida a Amazon ECR.Configuramos AWS App Runner, creando el servicio y conectándolo con el repositorio ECR creado. Luego, testeamos la API para asegurarnos que nos devuelva datos. Finalmente, seteamos el scheduler para que implemente las funciones por las noches.

Dificultades encontradas para desarrollar el sistema

Las mayores dificultades con las que nos encontramos fueron durante el primer paso del trabajo práctico.

El principio del trabajo práctico fue desafiante. Nos encontramos con un entorno completamente nuevo y fuimos aprendiendo mucho al andar. El primer paso desafiante fue conectar RDS con EC2. Tuvimos varios problemas al configurar los security groups y aprendimos acerca de la importancia de configurar correctamente los **Security Groups**, las credenciales y las reglas de red. También aprendimos a usar herramientas como `psql` y revisar logs para diagnosticar problemas de conectividad y configuración. Una vez que pasamos este obstáculo, nos llevó mucho tiempo re-configurar la conexión de RDS con EC2 después de que se borraran las instancias de RDS de todos los grupos.

Una vez que logramos implementar correctamente la base de datos, la instancia y Airflow, el mayor desafío apareció al configurar el DAG. Esto incluyó, primero, entender y adaptarnos al formato del DAG, y luego implementar las funciones específicas, como las de filtrado, cálculo del top CTR, top product, y escritura en la base de datos. Algunos de los errores con los que nos encontramos:

- A la hora de filtrar no estábamos seleccionando los campos correctamente. Perdimos mucho tiempo dado que creíamos que el error estaba en los comandos, sin embargo era simplemente el campo de `advertiser_ids`.

  ```
  advertisers_df = pd.read_csv(s3.get_object(Bucket=bucket_name, Key='advertiser_ids.csv')['Body'])
  ```

- En la implementación de top product, el uso de `sort_values` no devolvía los productos de mayores CTR hasta que logramos identificar e implementarlo aclarando un ordenamiento a partir de la columna CTR. Lo mismo nos ocurrió con Top Product. Luego de finalizar el trabajo tuvimos que volver a este punto dado que buscamos también que nos den el top por día.

  ```
  top_ctr_df = ctr_df.groupby(['advertiser_id', 'date']).head(20).reset_index(drop=True)
  ```

  ```
  top_products_df = top_products_df.groupby(['advertiser_id', 'date']).head(20).reset_index(drop=True)
  ```

- Si bien logramos realizar los cálculos sobre los datos y observamos que daban resultados, nos costó implementar la función `escribir_en_bd.` Lo solucionamos con `create_table_queries` que crea un diccionario que crea dos tablas en la base de datos de PostgreSQL, top_ctr_df y top_products_df, que almacenan los datos calculados previamente. A su vez, originalmente, `SQLAlchemy` se usaba para insertar datos en PostgreSQL, específicamente con `pandas.to_sql()`. Aunque esta solución es sencilla, no proporciona un control granular sobre la creación de tablas ni sobre las operaciones de inserción. Tuvimos muchos problemas usando esta librería y decidimos cambiar a la librería de `psycopg2, la cual nos` permitió incluir lógica para crear las tablas automáticamente si no existían. A su vez, nos otorgó **mayor control,** permitiéndonos gestionar la creación de tablas, inserciones personalizadas y manejo detallado de errores.
- Al implementar la función **`top_products,`** la tabla **`top_product_df`** no existía. Esto causaba errores al consultar desde la API. La solución encontrada fue modificar la tarea de escritura para crear la tabla si no existía.

Durante la implementación tuvimos varios errores similares a los anteriores, los cuales aprendimos a depurar. Para ello, probamos cada tarea de forma individual y añadimos mensajes de advertencia y manejo de excepciones en las funciones, lo que nos permitió mejorar la gestión de errores y garantizar un comportamiento más robusto del sistema.

Otro gran aprendizaje de esta etapa fue la importancia de mantener un historial de versiones del código. Inicialmente, subíamos todo de forma local directamente a nuestra carpeta de

DAGs desde la terminal. En una de las modificaciones, reemplazamos un DAG funcional por otro que no lo era, lo que nos hizo perder avances en nuestro trabajo. Posteriormente, al avanzar con los puntos 2 y 3, adquirimos mayor familiaridad en GitHub. Aunque fue un desafío comprender cómo funcionaban las versiones y cómo interactuar con el repositorio, esta herramienta solucionó el problema del versionado y facilitó el trabajo colaborativo.

Al configurar el entorno de Docker nos encontramos con problemas al incluir todas las dependencias necesarias y que el contenedor se construyera sin errores. Adicionalmente, se encontraron varios errores relacionados con la instalación de bibliotecas. También tuvimos un exceso de capacidad lo cual resolvimos eliminando paquetes y librerías. Durante la construcción de la imagen Docker, se encontraron errores relacionados con la instalación de `psycopg2`, como la falta de las herramientas de compilación necesarias.