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.

ChatGPT:  
  
**Informe Mejorado**

**TRABAJO PRÁCTICO PROGRAMACIÓN AVANZADA PARA GRANDES VOLÚMENES DE DATOS**

**Integrantes:**

Martina De Angelis y Santiago Lares Harbin

**Introducción**

En el presente trabajo práctico se desarrolló un sistema de recomendación de productos como solución de un producto AdTech, convirtiéndolo en un servicio accesible mediante una API. Este informe detalla los pasos más importantes del proceso y las dificultades encontradas durante el desarrollo del sistema, conforme a lo solicitado en el enunciado del trabajo práctico.

**Descripción del Problema**

El problema a resolver consiste en la recomendación de artículos en publicidades por internet, considerando la historia de navegación del usuario. Para ello, se desarrollaron dos procesos clave: la generación de recomendaciones y el servicio de recomendaciones a través de una API.

**Implementación**

**1. Pipeline de Procesamiento de Datos**

**Configuración Inicial:**

* **Configuración de IAM en AWS:** Se configuraron las políticas de acceso necesarias para que los servicios pudieran interactuar entre sí de manera adecuada.
* **Preparación de Datos Crudos:** Los datos crudos se almacenaron en un bucket de S3, necesarios para nuestro DAG.

**Instalación y Configuración de Airflow:**

* **Entorno Virtual:** Se instaló Airflow en un entorno virtual específico.
* **Pruebas Locales:** Se realizaron pruebas locales para verificar el correcto funcionamiento del DAG antes de su implementación en una instancia EC2.
* **Migración a EC2:** Se migró el sistema a una instancia EC2, cambiando la versión preinstalada de Python por una versión anterior compatible con Airflow. Esto incluyó copiar la carpeta de DAGs y reinstalar y reconfigurar Airflow en EC2. También se ajustó la configuración de EC2 para permitir el tráfico a través del puerto 8080 utilizado por Airflow.

**Desafíos y Soluciones:**

* **Configuración de Airflow:** Enfrentamos varios retos relacionados con la instalación y configuración de Airflow, así como con la ejecución eficiente de los DAGs. Tras resolver estos problemas y confirmar su funcionamiento local, migramos el sistema a EC2.
* **Permisos en EC2:** Fue necesario ajustar los permisos en EC2 para permitir el tráfico a través del puerto 8080.

**Desarrollo del Pipeline:**

* **Filtrado de Datos:** Se filtraron los logs crudos para excluir datos de advertisers inactivos.
* **Modelos de Recomendación:**
  + **TopCTR:** Se computó el modelo TopCTR, que devuelve los 20 productos con mejor click-through-rate para cada advertiser activo.
  + **TopProduct:** Se computó el modelo TopProduct, que devuelve los 20 productos más vistos en la web del cliente.
* **Escritura en Base de Datos:** Los resultados de los modelos se escribieron en una base de datos PostgreSQL disponible en AWS RDS.

**Configuración del Executor:**

* **Cambios en airflow.cfg:**
  + Se configuró el executor como LocalExecutor.
  + Se estableció el parallelism en 2 para no saturar los recursos del servidor.
  + Se configuró la conexión a la base de datos PostgreSQL en AWS RDS.

**2. Implementación de la API**

**Desarrollo de la API:**

* **Framework:** La API fue desarrollada utilizando FastAPI.
* **Arquitectura en Módulos:** La API fue diseñada con una arquitectura modular para asemejarse a un entorno de producción real.
* **Pruebas Locales:** Se realizaron pruebas locales para verificar su correcto funcionamiento antes de proceder con la dockerización.

**Dockerización y Deploy:**

* **Configuración de Docker:** Se instaló Docker Desktop y se crearon los archivos necesarios (Dockerfile y app\_requirements.txt) para construir la imagen de la aplicación.
* **Resolución de Dependencias:** Inicialmente, hubo problemas con las dependencias debido a la reutilización de un venv de un proyecto anterior. Se resolvió desinstalando los paquetes del venv y reinstalando solo los necesarios.
* **Creación de la Imagen Docker:** Una vez resuelto el problema de dependencias, se creó la imagen Docker y se verificó su correcto funcionamiento local.
* **Implementación en AWS:** Se configuraron instancias de ECS y ECR. Configurar estas instancias presentó desafíos relacionados con las políticas de acceso, que se resolvieron especificando manualmente los accesos necesarios en un archivo JSON.

**Despliegue Final:**

* **App Runner:** Finalmente, se desplegó la API en App Runner, asegurando su disponibilidad y correcto funcionamiento.

**Conclusión**

El desarrollo de este sistema de recomendación de productos y su implementación en un entorno de producción en AWS presentaron múltiples desafíos técnicos que fueron superados a través de pruebas, ajustes de configuración y la correcta gestión de dependencias. La configuración modular y el uso de Docker garantizaron un entorno robusto y escalable para la API, mientras que el pipeline de procesamiento de datos, implementado en Airflow, permitió la generación eficiente de recomendaciones basadas en datos de navegación.

**Nota:** Se recomienda revisar periódicamente el resumen de costos en AWS para evitar gastos inesperados, considerando que todos los servicios utilizados entran en el free tier o tienen un costo muy bajo por hora de uso.