Analytics-Driven Decisions.

Prueba técnica

Documento de descripción de prueba técnica para que puedas demostrar tus capacidades en el ámbito de la ciencia de datos manejando conceptos básicos del desarrollo de soluciones de ciencia de datos, su ciclo de vida y toda la ingeniería que pueda girar en torno a poner en producción una solución de analítica avanzada.

Objetivo

El **objetivo de la prueba técnica** es que puedas demostrar tus capacidades en el ámbito de la ciencia de datos manejando conceptos básicos del desarrollo de soluciones de ciencia de datos, su ciclo de vida y toda la ingeniería que pueda girar en torno a poner en producción una solución de analítica avanzada. Somos conscientes de que algunos de los componentes tecnológicos que proponemos para esta prueba técnica quizás no los conozcas y supongan un reto para ti, pero pensamos que por tus capacidades podrás adquirir un conocimiento básico en estos días para sacar adelante la prueba.

Configurando el entorno de trabajo: Docker + Airflow

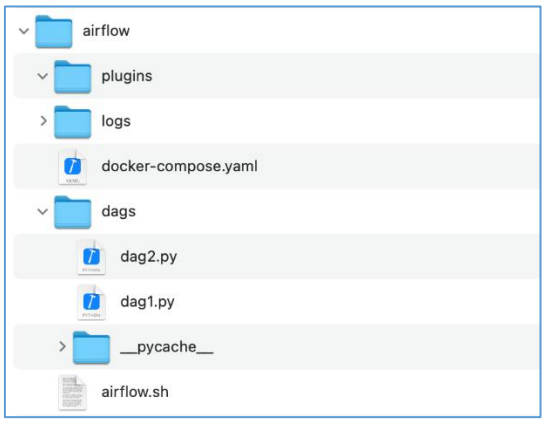
1. El primer punto que deberás tratar en la prueba técnica es el uso de **Docker** (https://www.docker.com/) para desplegar **Apache Airflow** (https://airflow.apache.org/). Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. Este punto es importante porque te servirá para montar el entorno que proponemos a continuación.

   > **HOW-TO:** Docker por lo general, se maneja mejor en entornos Linux, pero es perfectamente viable utilizarlo en Windows siempre que tengas Windows 10. Si estás en Windows, te recomendamos utilizar WLS2. Tienes más información sobre como instalar Docker en el siguiente enlace: https://www.docker.com/get-started

2. Una vez te hayas familiarizado con Docker, deberás usarlo para desplegar **Apache Airflow**. Apache Airflow es un orquestador de procesos que, a diferencia de otros, se basa en la especificación de los flujos de ejecución a través de *scripts* de Python. El **orquestador** es una pieza importante dentro de cualquier plataforma de analítica avanzada, ya que es la pieza que permite integrar servicios provenientes de diversas fuentes de datos, y proveer información de forma síncrona o asíncrona a otros servicios conectado orígenes de datos con destinos mediante la definición de *tareas* o *actividades*.
   En el caso de Airflow, la orquestación entre las tareas se realiza definiendo lo que se conoce como DAG (Directed Acyclic Graph). Un DAG es una colección de tareas organizadas mediante dependencias entre ellas y estableciendo relaciones para decir en que orden se deben ejecutar dichas tareas (https://airflow.apache.org/docs/apache-airflow/stable/concepts/dags.html).

   Para facilitar la labor de cómo usar Apache Airflow desplegado en un contenedor de Docker te vamos a dar las siguientes pistas:

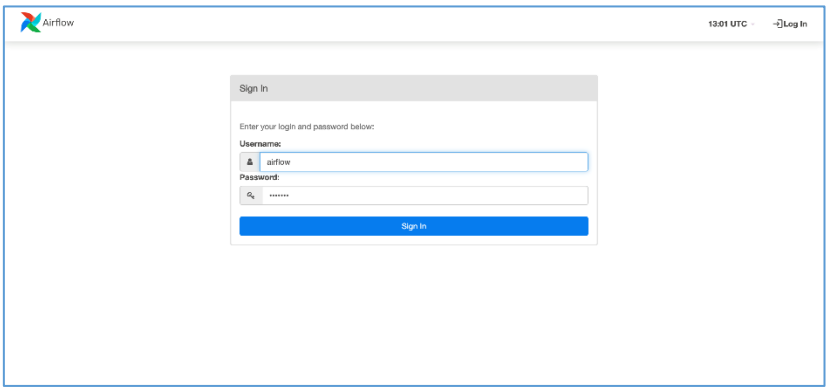
   > **HOW-TO:** Sigue las instrucciones que encontrarás en el siguiente enlace para desplegar Apache Airflow en Docker usando un archivo de configuración *".yaml"* llamado *Docker Compose*: https://airflow.apache.org/docs/apache-airflow/stable/start/docker.html.
   >
   > Básicamente, los pasos que tendrás que seguir son los siguientes:
   >
   > - Descargarte el archivo de configuración y ponerlo en una carpeta de proyecto nueva acompañado de otras tres subcarpetas (\dags, \logs y \plugins). La carpeta de \dags es donde tendrás que guardar los flujos de las tareas que posteriormente tú vayas implementando para resolver esta prueba técnica. La carpeta de proyecto tendrá entonces este formato inicial:



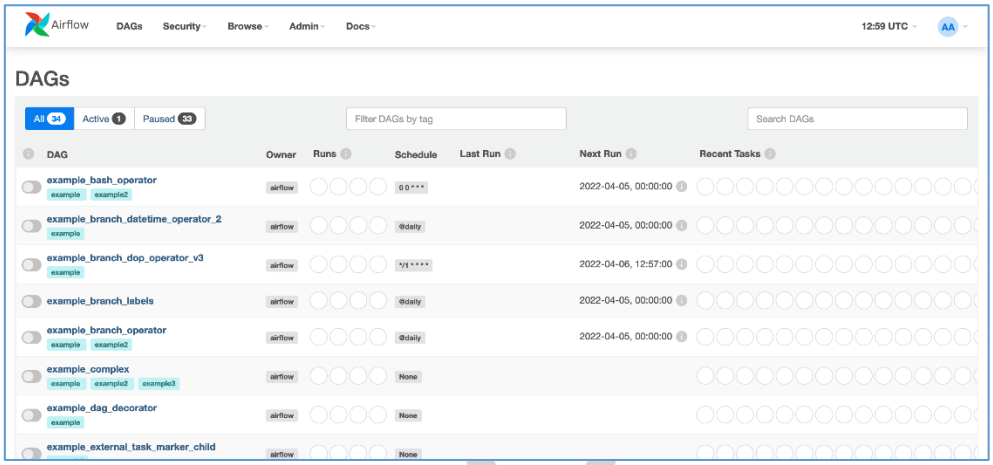
- Como segundo paso principal tendrás que inicializar la base de datos que da soporte a los datos que maneja el orquestador mediante el comando **docker-compose up airflow-init**. Este comando se ejecuta por Terminal en la carpeta donde tengas almacenado el *docker-compose.yaml* y se encarga de inicializar todos los servicios necesarios para levantar Airflow en Docker. Así:

```
angel@MacBook-Pro-de-Angel airflow % docker-compose up airflow-init
Creating network "airflow_default" with the default driver
Creating airflow_redis_1    ... done
Creating airflow_postgres_1 ... done
Creating airflow_airflow-init_1 ... done
Attaching to airflow_airflow-init_1
airflow-init_1       | The container is run as root user. For security, consider using a regular user account.
airflow-init_1       |
airflow-init_1       | /home/airflow/.local/lib/python3.7/site-packages/sqlalchemy/dialects/postgresql/base.py:357
3 SAWarning: Predicate of partial index idx_dag_run_queued_dags ignored during reflection
airflow-init_1       | /home/airflow/.local/lib/python3.7/site-packages/sqlalchemy/dialects/postgresql/base.py:357
3 SAWarning: Predicate of partial index idx_dag_run_running_dags ignored during reflection
airflow-init_1       | DB: postgresql+psycopg2://airflow:***@postgres/airflow
airflow-init_1       | [2022-04-06 13:16:16,415] {db.py:919} INFO - Creating tables
airflow-init_1       | INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
airflow-init_1       | INFO  [alembic.runtime.migration] Will assume transactional DDL.
airflow-init_1       | Upgrades done
airflow-init_1       | [2022-04-06 13:16:27,578] {manager.py:512} WARNING - Refused to delete permission view, ass
oc with role exists DAG Runs.can_create User
airflow-init_1       | airflow already exist in the db
airflow-init_1       | 2.2.5
airflow_airflow-init_1 exited with code 0
angel@MacBook-Pro-de-Angel airflow %
```

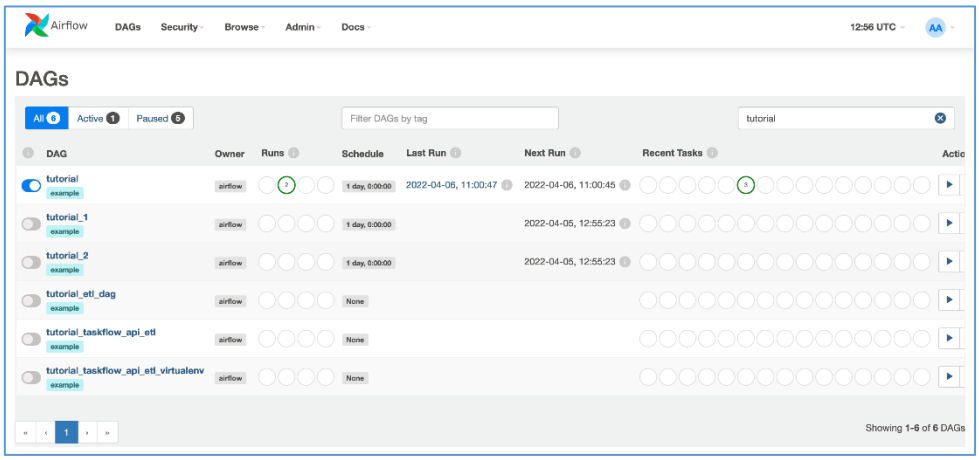
- Finalmente, tendrás que dejar correr (run) Airflow usando el comando **docker-compose up**. Este comando hará que Airflow sea accesible desde un navegador a través del enlace http://localhost:8080/. Las instrucciones facilitadas te crearán un usuario con permisos de administrador para loggearte en la interfaz web de Airflow con usuario ***airflow*** y password ***airflow***. Sabrás que ha funcionado si al entrar en el navegador a través de la dirección de localhost ves una pantalla al loggearte de este tipo:



- Una vez loggeado, verás una lista completa de DAGs de ejemplo de la siguiente forma:



- Además, verás aquellos DAGs que hayas ido dejando en la carpeta \dags del proyecto. Por ejemplo, en nuestro caso hemos dejado dos dags con nombre “tutorial_1” y “tutorial_2”.



Es muy normal atascarse durante el despliegue de Airflow en Docker, pero para evitar estos problemas es muy importante seguir todas las instrucciones indicadas en los enlaces facilitados una a una. A partir de aquí, familiarízate con la interfaz de Airflow, el concepto de DAG (puedes ver el código entrando en cada uno de ellos) y cómo ejecutarlos.

Tarea que deberás realizar

Una vez que tengas Apache Airflow desplegado en local usando Docker y te hayas familiarizado con su interfaz, la forma de construir DAGs y la forma de ejecutarlos, te proponemos que construyas un DAG que represente el ciclo de vida típico de construcción y aplicación de un modelo de analítica avanzada. Para ello deberás usar como origen de datos el CSV que se te da anexo a esta prueba (*dataset.csv*), para entender que contiene también tienes disponible el CSV *data_descriptions.csv* donde se describen las columnas que contiene el conjunto de datos facilitado aportando su significado.

> De manera resumida, *el dataset facilitado contiene una lista de clientes* (identificados mediante la columna *Customer_Id*) pertenecientes a una empresa del sector de las Telecomunicaciones. Esta empresa nos ha pedido ayuda acerca de cómo mejorar la gestión de su cartera de clientes. En concreto, tienen interés en la aplicación de técnicas analíticas para conocer los factores asociados a los motivos que lleva a un cliente a abandonar la compañía, con el fin de implementar mejoras en sus procedimientos. Para ello, verás que el resto de las columnas del *dataset* contienen información sobre, por ejemplo, cuantas llamadas o SMSs ha recibido el cliente.

Tu labor aquí será, a partir del conjunto de datos facilitado, construir un DAG que tenga un paso de preparación de datos y que cargue el tablón de entrada para el modelo de analítica avanzada de tu elección, otro paso que entrene el modelo y guarde el modelo empaquetado en un *pickle*, y un último paso que cargue el modelo previamente entrenado y guardado y, con un conjunto de datos de *hold-out*, evalúe el modelo en términos de la métrica que consideres más adecuada y muestre esa métrica de alguna manera (escribiéndola en el log de ejecución, por ejemplo).

Para dar solución a tu problema, tendrás que desarrollar un modelo predictivo acorde con las etapas que pueda implicar (análisis de datos, preparación, modelado, medición…). Piensa que los usuarios de negocio de la compañía querrán conocer cuál es el efecto individual de cada variable sobre el evento de abandono de cada cliente. Además, se desea entender qué efecto genera cada variable sobre el modelo (*feature importance*), y cuando se puntúa a un cliente, qué variables están favoreciendo o penalizando el riesgo de abandono de la compañía.

Finalmente, para dar orden a tu trabajo, te pedimos que prepares una pequeña presentación que contenga las decisiones que has ido tomando y que contenga una pequeña descripción del problema a solventar, la arquitectura montada para dar solución a este problema y la descripción del modelo analítico implementado para dar respuesta a las cuestiones planteadas. Además, deberás entregar el código realizado con el fin de valorarlo desde un punto de vista de buenas prácticas de programación.

Posibles mejoras

La prueba técnica es bastante libre de interpretación, por lo que te damos pie a pensar por ti mismo/a.

- Así, te animamos a enriquecer la solución provista y, si quieres, investigar un poco sobre como desplegar una base de datos (PostgreSQL, por ejemplo) que te permita guardar todos los datos del proceso (los de inicio, los intermedios y los finales) en un sistema que no sea el almacenamiento local de Apache Airflow o el log de ejecución. Esto implicará modificar el proceso para conectar con la base de datos para leer/escribir.
- También podrías desplegar una instancia de *MLflow* (https://mlflow.org/) que te permita registrar los parámetros y métricas del modelo generado, lo cual es bastante habitual a la hora de aplicar una metodología *MLOps*.
- También se valorará de forma positiva toda integración con herramientas de monitorización que quieras plantear (Prometheus y Grafana, por ejemplo).
- Finalmente, cualquier cosa que se te ocurra para enriquecer la solución en términos de ingeniería, ya sea integrando nuevas piezas en la arquitectura, u optimizando los procesos tanto en rendimiento como en calidad del código, será igualmente bien valorada.

Por otro lado, independientemente de hasta donde hayas podido llegar, también se valorará lo que hayas investigado, la exposición que hagas del problema a solventar y todo lo que se te ocurra a ti aportar a la prueba más allá de lo que nosotros proponemos (por ejemplo, si te ves fuerte en la visualización de datos y crees que hay algo en este punto que puedas integrar, adelante).

Si en algún momento tienes algo que comentar con nosotros, o necesitas que movamos la fecha o la hora de presentación por algún tema personal, no dudes en contactarnos a través de las siguientes direcciones de correo:

Jesús Vicente García Hernández (jesusvicente.garcia@sdggroup.com)
Andrés Padrones García (andres.padrones@sdggroup.com)

¡Muchas gracias y suerte!