Práctica 0: Despliegue del entorno y primeros pasos

Procesamiento Masivo de Datos y Visualización (PMDV)

Curso 2025-2026

1. Introducción

Con el objetivo de poner en práctica los conceptos aprendidos en teoría, se dispone de un entorno que incluye algunas de las principales herramientas.

- **Almacenamiento distribuido:** MinIO
- **Procesamiento Distribuido:** Apache Spark, para su uso con Python (PySpark). Además, se incluyen las librerías necesarias para usar los formatos Delta e Iceberg.
- **Orquestación:** Apache Airflow
- **Calidad de Datos:** Great Expectations
- **Exploración y Visualización:** Jupyter Lab. Además, se incluyen las librerías Plotly para Python.

Las anteriores herramientas conforman una arquitectura Big Data moderna, de código abierto y ampliamente utilizadas en los proyectos de datos de las empresas y organizaciones.

Para facilitar su despliegue, el entorno se ha definido con la herramienta Docker, la cual permite empaquetar y distribuir las distintas herramientas que conforman la arquitectura. Las herramientas ya se encuentran pre-configuradas para uso conjunto.

El aprendizaje de Docker queda fuera del alcance de esta asignatura. No obstante, se proporcionan los comandos básicos para su uso, incluyendo el arranque y parada del entorno.

En cuanto al entorno de desarrollo, se usará principalmente de Visual Studio Code/PyCharm. No obstante, también se usará la herramienta de Notebooks, Jupyter Lab. Esta herramienta permitirá ejecutar el código Spark (PySpark) de forma interactiva, facilitando el aprendizaje de Spark, la exploración interactiva de datos y la generación de visualizaciones sobre los datos procesados y almacenados en nuestro Data Lake.

Tanto Docker Desktop como Visual Studio Code, son herramientas gratuitas y se encuentran disponibles en los ordenadores de los laboratorios. Se usará el SO Windows para las prácticas y solo se garantiza el funcionamiento en este SO.

No obstante, el entorno y herramientas son compatibles con Linux (ej. Ubuntu), aunque podrían ser necesarios algunos ajustes en los scripts de Docker que definen el entorno.

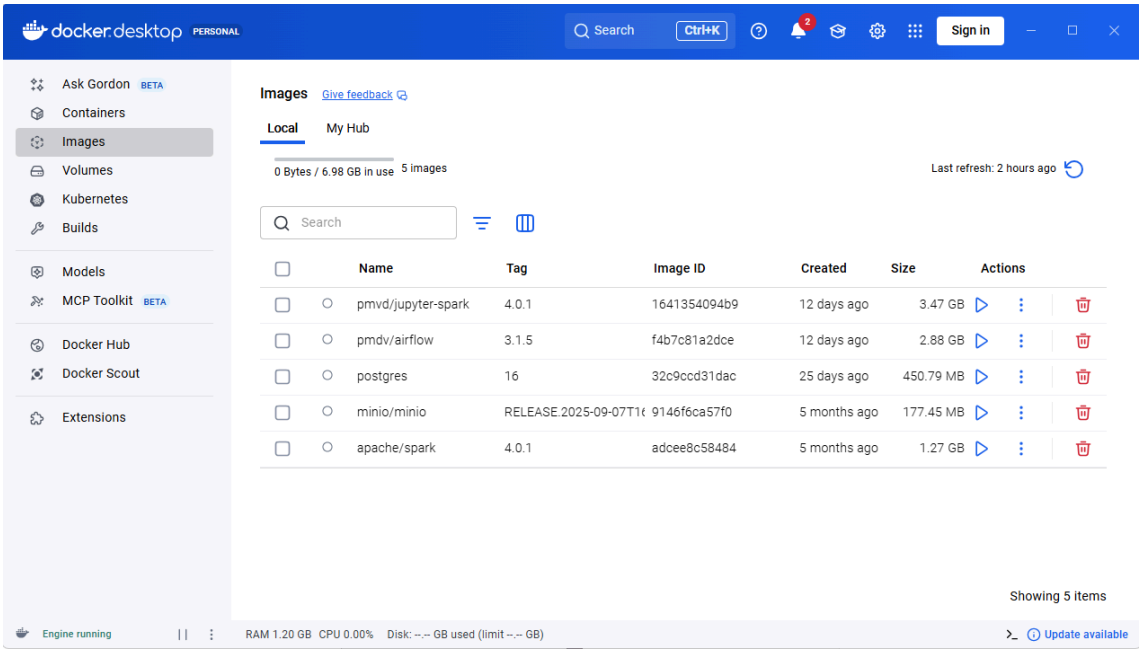
2. Descripción de la práctica

El objetivo principal de esta primera práctica es aprender a arrancar el entorno y probar el acceso a las distintas herramientas, como punto de partida fundamental para poder llevar a cabo las siguientes prácticas de la asignatura.

En los siguientes apartados se detalla cada una las tareas a llevar a cabo, así como la salida de cada una de ellas que forma parte de los archivos a entregar para la evaluación de esta práctica. En el apartado “3. Entrega de la Práctica”, se enumeran los archivos a entregar y las condiciones para dicha entrega.

2.1 Arranque del entorno

En primer lugar, arrancaremos Docker Desktop desde el menú de inicio. Será necesario para poder ejecutar los comandos de Docker más adelante, en la terminal que abriremos desde Visual Studio Code.



En los laboratorios, dado que se solicitó la creación previa del entorno con Docker en cada puesto, debería aparecer el listado de imágenes anteriores. En otro caso se han revisar los anexos para alternativas de uso mediante USB o instalación en tu equipo personal.

Tras esto, se ha de localizar o descargar la carpeta “infra”. En los ordenadores de los laboratorios debería encontrarse en “*C:\sw\infra*”. En otro caso puede descargarse desde el Moodle de la asignatura “Software de la asignatura sobre Docker”.

Dentro de esta carpeta, tendremos a su vez dos carpetas:

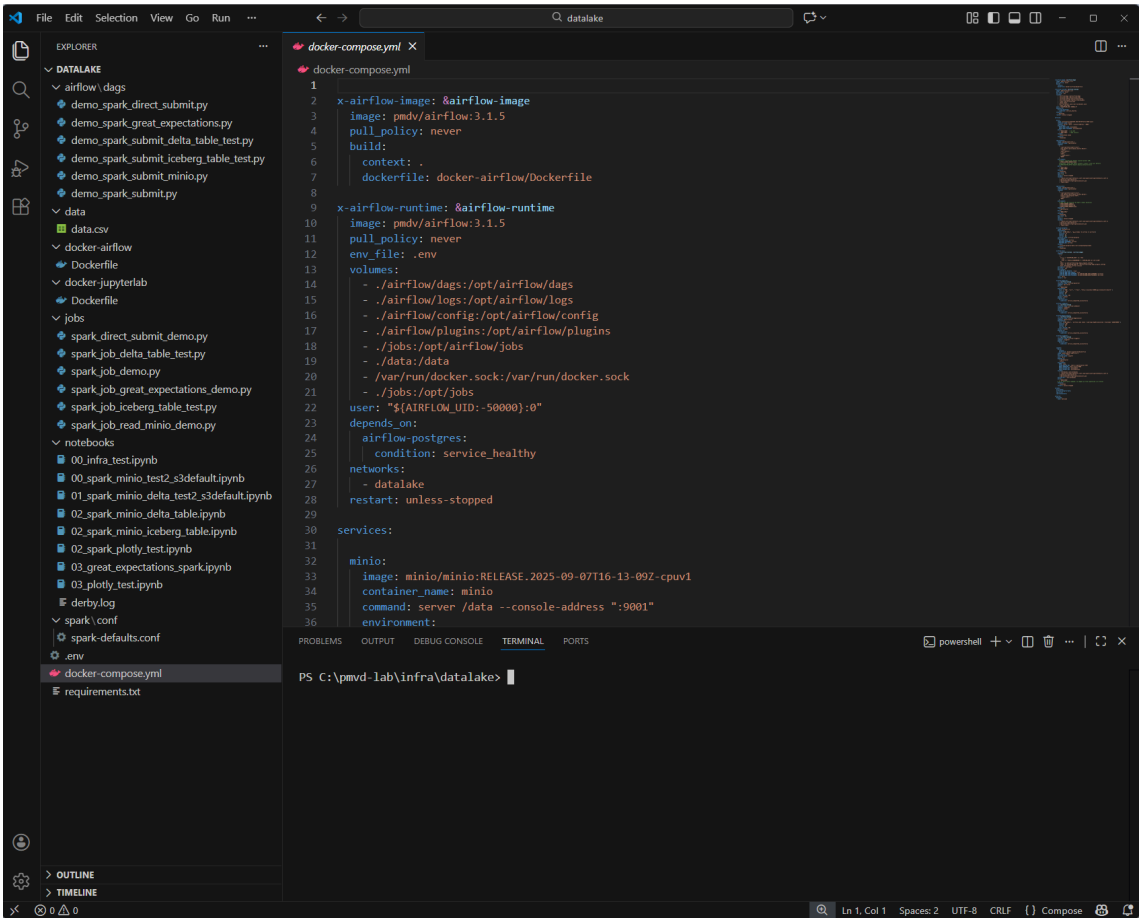
- ***datalake***: Contiene los archivos imprescindibles para desplegar el entorno con Docker.
    - docker-compose.yml
    - requirements.txt
    - .env
    - spark
    - docker-airflow
    - docker-jupyterlab
- ***tests***: Contiene los ejemplos de código a ejecutar como parte de esta Práctica 0.
    - airflow
    - data
    - jobs
    - notebooks

Se ha de mover o copiar las 4 carpetas dentro de *tests* en la carpeta *datalake*.

En este momento, se iniciará Visual Studio Code, que será nuestro entorno de desarrollo principal. En Visual Studio, hacemos clic en *File->Open Folder*. Se debe buscar la carpeta datalake y seleccionarla para abrirla.

No es necesario abrir ningún archivo de la carpeta datalake para arrancar el entorno. No obstante, el archivo raíz con la definición del entorno se encuentra en *datalake/docker-compose.yml*. Este archivo es el que usará por defecto el comando de arranque de Docker que se ejecutará a continuación.

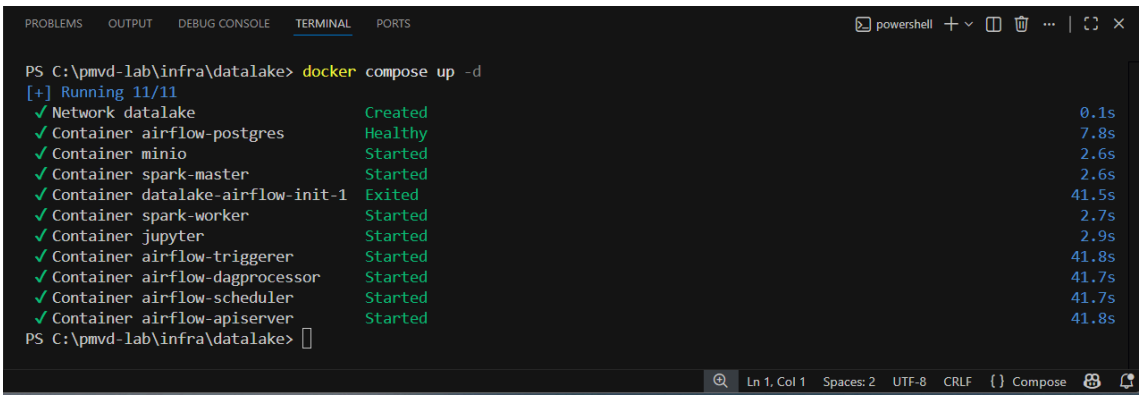
Para ejecutar ese comando y cualquier otro comando Docker, se usará una terminal en Visual Studio (zona de abajo a la derecha en la captura anterior). Si no apareciese una terminal, es posible abrir una nuevo haciendo clic en el menú superior, *Terminal->New Terminal.*



Una vez en el terminal se ejecutará el siguiente comando para arrancar el entorno:

- **docker** compose up -d

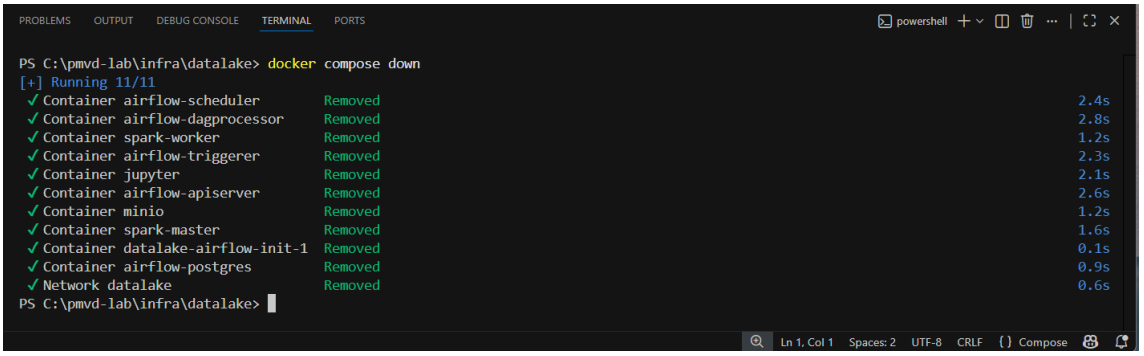
Si todo va bien, se mostrarán y listarán los distintos contenedores que contienen los servicios que conforman nuestro entorno (minio, spark-master, spark-worker,..), informado sobre el estado del arranque. Si el arranque concluye con éxito obtendremos el siguiente resultado.



En este momento ya se puede avanzar al siguiente apartado “2.2 Acceso a MinIO y carga manual de datos”.

Aunque **no lo ejecutes aún si vas a continuar con la práctica**, a continuación, se indica el comando para detener el entorno:

- **docker** compose down



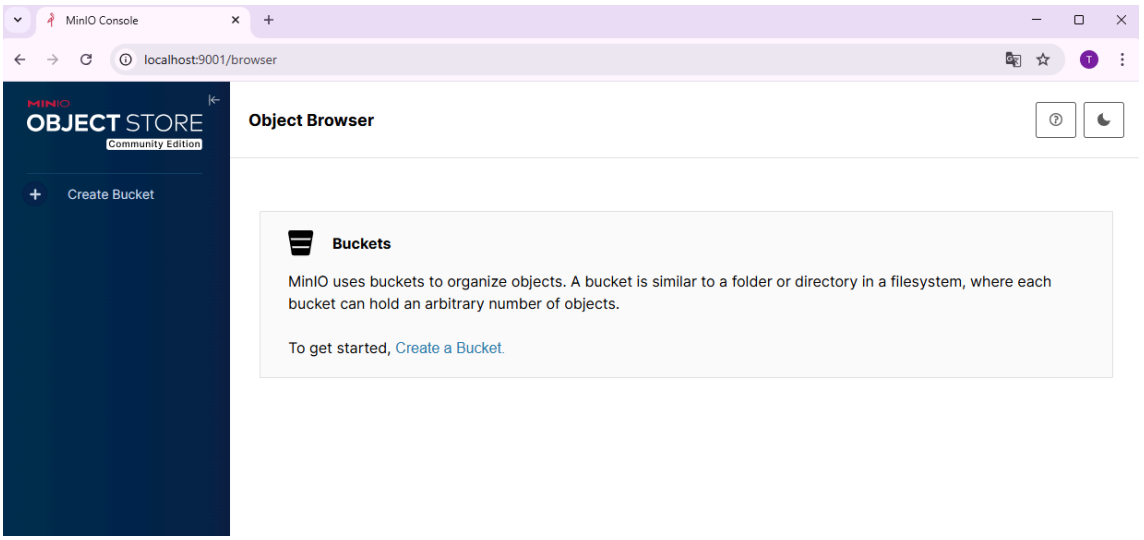
2.2 Acceso a MinIO y carga manual de datos

MinIO es un sistema de almacenamiento distribuido de código abierto. Está basado la tecnología de almacenamiento de objetos y se ha diseñado para su uso en la nube, siendo su interfaz compatible con la de AWS S3, servicio de almacenamiento cloud ampliamente usado y compatible con las herramientas de Big Data.

Una vez arrancado el entorno con Docker, es posible acceder a MinIO a través de tu navegador:

- **url:** http://localhost:9001
- **usuario:** minioadmin
- **pass:** minioadmin123

Si no hemos usado el entorno previamente el almacenamiento en MinIO estará vacío, como se muestra en la siguiente captura.

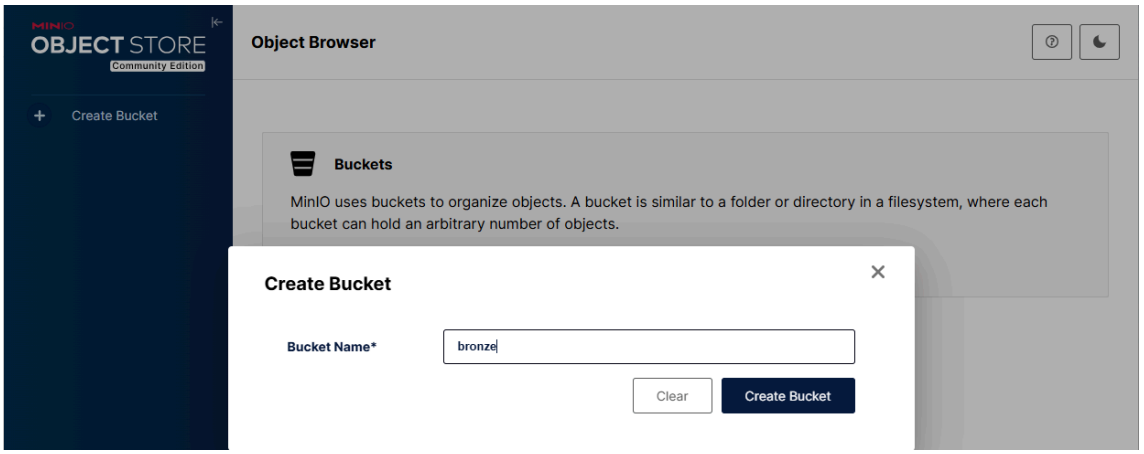


En MinIO y, en general, en sistemas basados en almacenamiento de objetos, las distintas zonas de datos se aíslan en buckets. Un bucket es un contenedor de almacenamiento donde se pueden subir archivos y organizarlos en carpetas.

En nuestro caso vamos a crear **cuatro buckets**:

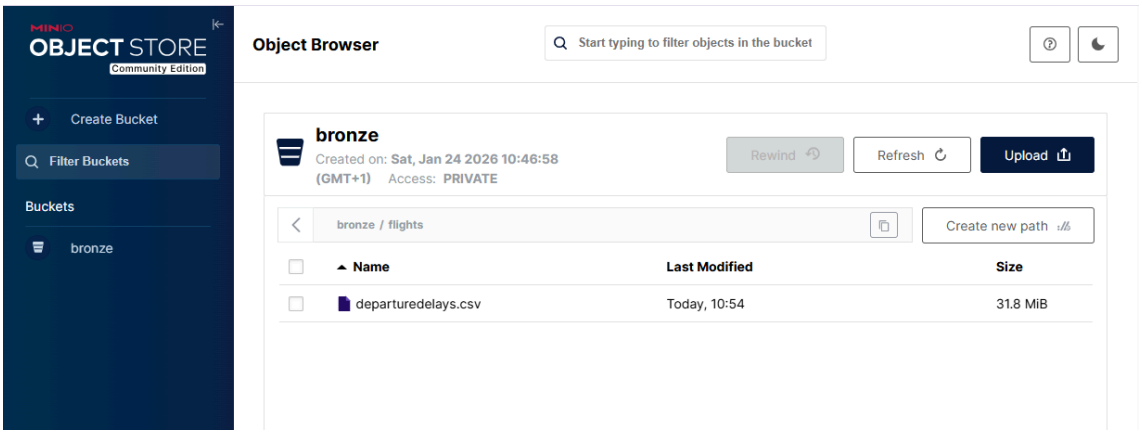
- **bronze:** Contendrá los conjuntos de datos que subamos manualmente, en bruto y sin transformar, y que se usarán como fuentes de datos de las prácticas. Los enlaces para descargar los conjuntos de datos necesarios para cada práctica se irán publicando en el Moodle de la asignatura (sección "*Datasets necesarios para las prácticas*").
- **silver:** Se usará como destino de los conjuntos de datos procesados a partir de bronze, tras aplicar operaciones básicas de limpieza y estructuración a nivel de conjunto de datos, en formatos optimizados (por ejemplo, Parquet) para su uso con herramientas como Spark.
- **lakehouse:** Representa la capa gold, que almacenará los datos procesados tras aplicar uniones, agregaciones y modelado, de forma que queden listos para su consulta, visualización (por ejemplo, cuadros de mando) o IA / Machine Learning.

La forma más sencilla de crear Buckets en MinIO es usar la interfaz web mediante la opción "+ Create Bucket" (menú de la izquierda).



Una vez creado el primer bucket *bronze* se pueden empezar a subir archivos dentro del mismo. A continuación, se debe cargar el archivo "departuredelays.csv" dentro de una carpeta /flights en dicho bucket. Dentro del bucket *bronze* se hará clic en "Create new path", se introduce el nombre flights y hacemos clic en "Create". Para subir archivos en la nueva carpeta se hará clic en "Upload->Upload File" y se selecciona en nuestro equipo local el archivo "departuredelays.csv".

Si se han seguido correctamente los pasos se visualizará el archivo subido a la carpeta, que a su vez está dentro del bucket *bronze*.



Para esta práctica 0 solo es necesario usar este conjunto de datos en el bucket *bronze* y, además, usaremos el bucket *silver* para la salida de las pruebas. No obstante, se deben crear los otros dos buckets (gold y lakehouse) para dejar el entorno preparado para el resto de las prácticas.

2.2 Acceso a Jupyter Lab y prueba de un Notebook con código Spark

JupyterLab es una herramienta que permite la creación y ejecución de *notebooks*, combinando código, texto y resultados de forma interactiva.

En las prácticas de la asignatura se utilizará JupyterLab como apoyo para:

- La exploración inicial de los conjuntos de datos.
- El desarrollo de una primera versión del código de los procesos de datos con Spark + Python, antes de su orquestación.
- Facilitar el aprendizaje y la depuración del código Spark, gracias a su ejecución interactiva y a la visualización inmediata de resultados.

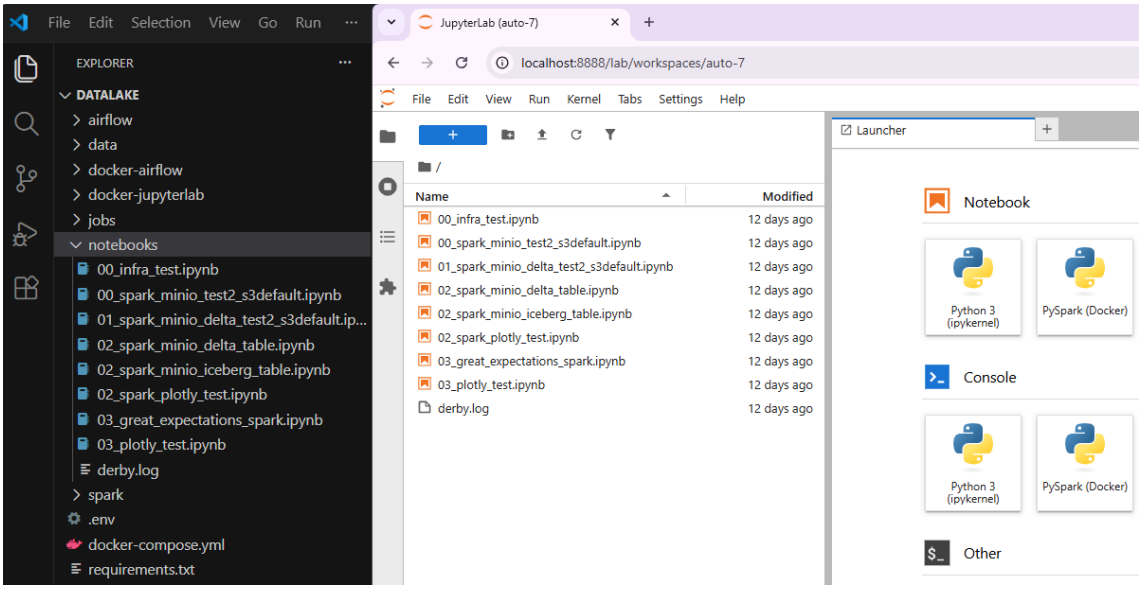
Además, en la Práctica 4, JupyterLab se empleará para la generación de visualizaciones a partir de los datos ya procesados y almacenados en el Data Lake.

Una vez arrancado el entorno con Docker, es posible acceder a Jupyter Lab a través de tu navegador:

- **url:** http://localhost:8888/

En la siguiente captura se muestra el entorno de Jupyter a la derecha y, a la izquierda, la estructura de carpetas desde Visual Studio Code. Se puede observar que los archivos (notebooks) que están disponibles en Jupyter se almacenan en la carpeta local "*/infra/notebooks*", por tanto, los podemos

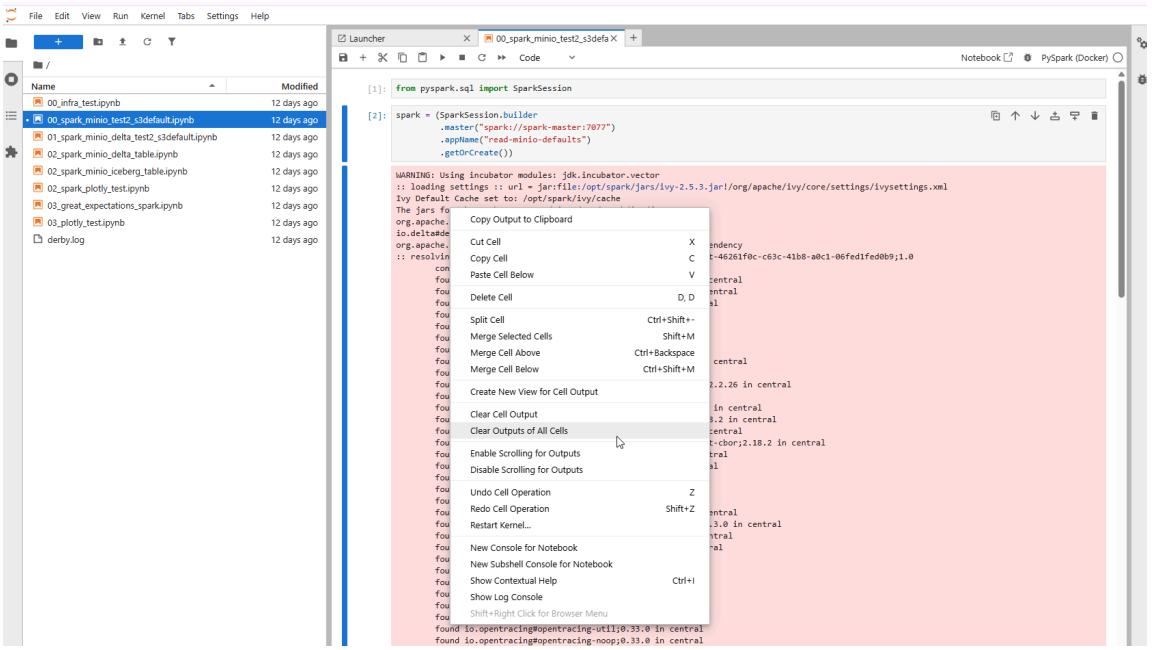
exportar para futuras ejecuciones y evitar que los borren (especialmente si usamos un equipo del laboratorio).

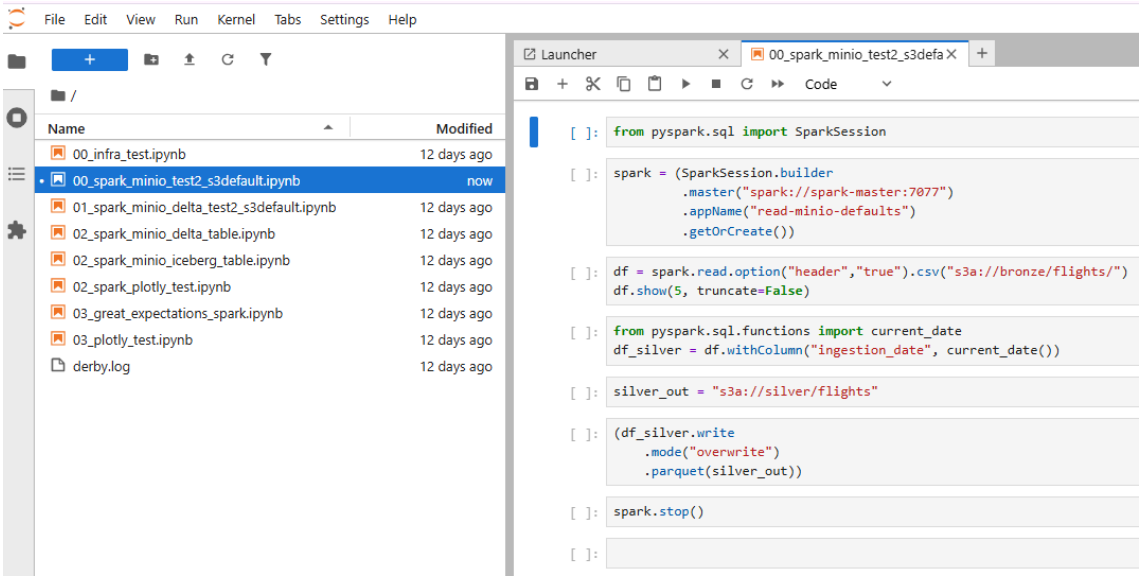


Es posible crear un nuevo notebook seleccionando el icono de “PySpark (Docker)”. Esto permitirá que se use el clúster de Spark desplegado con Docker y no una nueva instancia de Spark dentro del mismo contenedor de Jupyter.

No obstante, en esta práctica 0 el objetivo es abrir el notebook de test “*00_spark_minio_test2_s3default.ipynb*” y ejecutarlo paso a paso. Para abrirlo se hará clic en dicho archivo en el menú de la izquierda en Jupyter y como resultado se verá el código de este a la derecha.

Además, para facilitar una ejecución limpia, se hará clic en con el botón derecho del ratón en cualquier zona del notebook abierto y se hará clic en “Clear Output of All Cells”.





Ahora, se ejecutará el notebook bloque a bloque (también llamados celdas). Para ello, colocamos el cursor para garantizar la selección de la primera celda y ejecutamos pulsando Shift+Enter (o haciendo clic en el icono ▶). Esto ejecutará el código de la celda y moverá el foco a la siguiente, que podrá ser ejecutada de la misma forma. Repetiremos hasta llegar al final del Notebook.

Si alguna celda genera algún error al ejecutar el código Spark, debes revisarla para que se ejecute con éxito. Si se genera algún warning se debe ignorar para esta práctica.

A continuación, se muestra la salida correcta.

[7]:
```
df = spark.read.option("header","true").csv("s3a://bronze/flights/")
df.show(5, truncate=False)
```

```
26/01/24 10:30:24 WARN MetricsConfig: Cannot locate configuration: tried hadoop-metrics2-s3a-file-system.properties,hadoop-metrics2.properties
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

```
+--------+-----+--------+------+-----------+
|date    |delay|distance|origin|destination|
+--------+-----+--------+------+-----------+
|01011245|6    |602     |ABE   |ATL        |
|01020600|-8   |369     |ABE   |DTW        |
|01021245|-2   |602     |ABE   |ATL        |
|01020605|-4   |602     |ABE   |ATL        |
|01031245|-4   |602     |ABE   |ATL        |
+--------+-----+--------+------+-----------+
only showing top 5 rows
```

[8]:
```
from pyspark.sql.functions import current_date
df_silver = df.withColumn("ingestion_date", current_date())
```

[9]:
```
silver_out = "s3a://silver/flights"
```

[11]:
```
(df_silver.write
    .mode("overwrite")
    .parquet(silver_out))
```

[12]:
```
spark.stop()
```

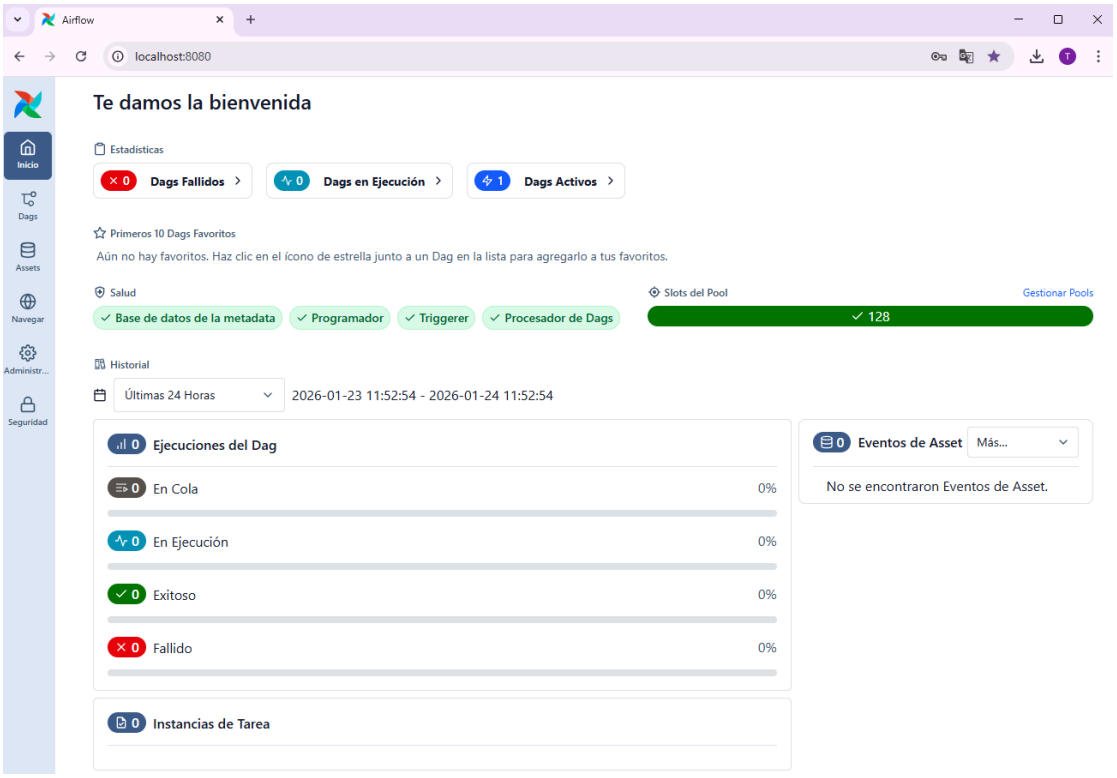
Para guardar el notebook ejecutado junto con la salida de sus celdas (no las borres como hicimos al inicio), puedes pulsar Ctrl+S o hacer clic en 💾. **Como parte de la entrega de la práctica se ha de incluir una copia del notebook ejecutado**, incluyendo el resultado de cada celda. De esta forma podremos conocer la fecha de ejecución y verificar que es una ejecución única.

2.3 Acceso a Airflow y prueba de un proceso de orquestación

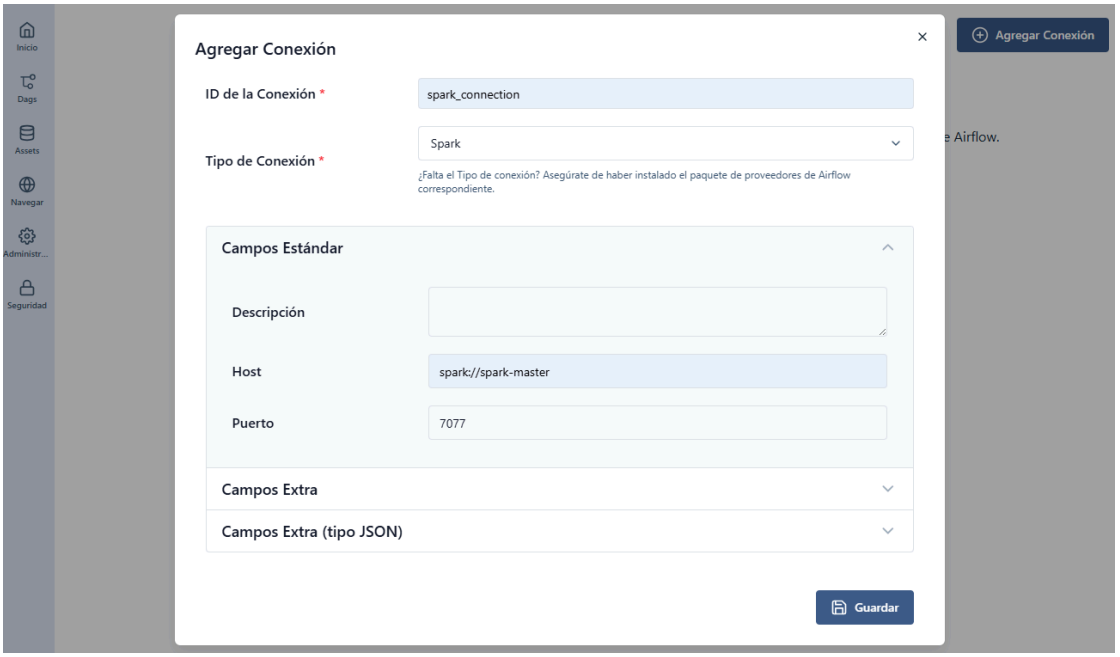
Apache Airflow es la herramienta que usaremos para la orquestación o automatización de la ejecución de los procesos de Spark desarrollados.

Una vez arrancado el entorno con Docker, es posible acceder a Airflow a través de tu navegador:

- **url:** http://localhost:8080/
- **usuario:** airflow
- **pass:** airflow

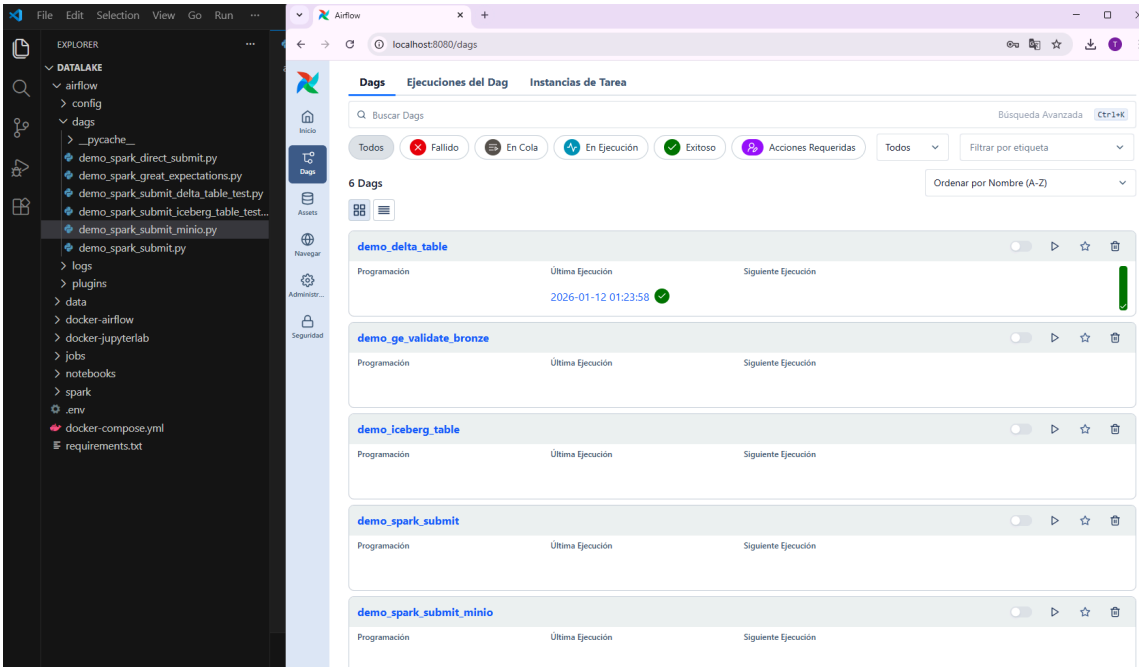


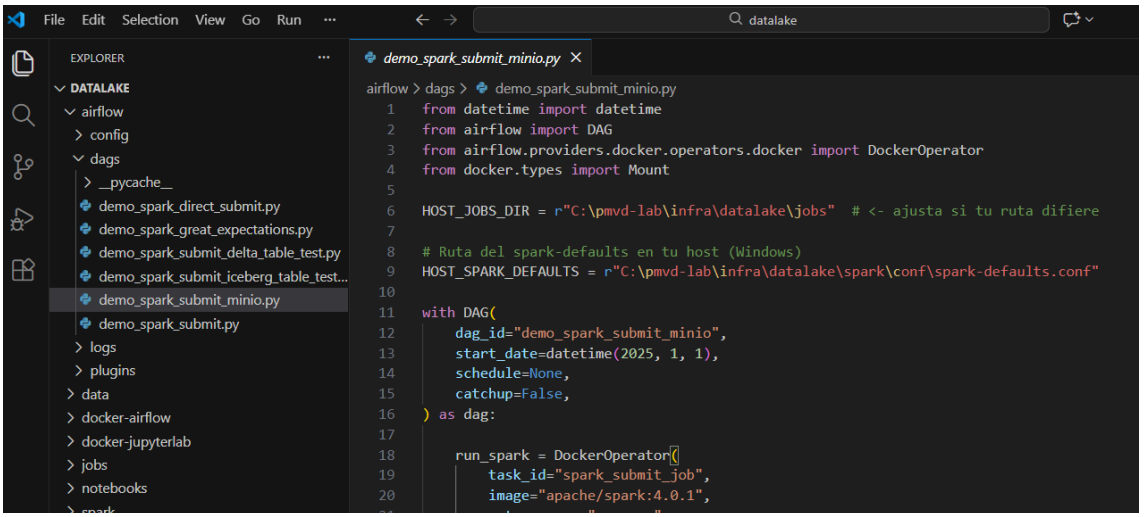
En primer lugar, se creará una conexión con clúster de Spark desplegado en otros contenedores de Docker como parte del entorno. Para ello, el menú lateral hacemos clic en “Administración” y dentro, arriba a la derecha, en “Agregar Conexión”. Se hará clic en Tipo de Conexión (puede tardar unos segundos en listarlas la primera vez) y seleccionaremos Spark. En la siguiente captura se muestran el resto de los parámetros a rellenar y los valores que se deben introducir.



Para terminar de crearla se hará clic en Guardar, tras lo que se debe mostrar la nueva conexión añadida en el listado de conexiones.

A continuación, se revisará la sección Dags (clic en el menú lateral izquierdo). En esta sección se muestran los procesos de orquestación disponibles y listos para ejecutar, de forma manual programada. Estos procesos se denominan DAG y se crean mediante código Python. Lo archivos que contienen el código fuente se almacenan en la carpeta *infra/airflow/dags* y se pueden editar para su desarrollo con Visual Studio Code. Al estar en esa carpeta se sincronizan automáticamente con el servidor de Airflow, donde se ejecutan y monitorizan de forma visual.





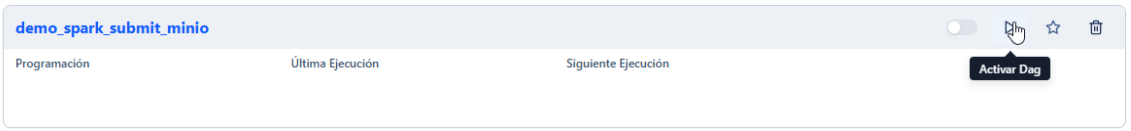
En esta práctica 0 se ha de ejecutar el DAG ***demo_spark_submit_minio***. Si se explora el código de este en el correspondiente archivo .py , podemos observar como hace referencia a un proceso de Spark desarrollado con Python "*jobs/spark_job_read_minio_demo.py*".

Este proceso es equivalente al código del Notebook que ejecutamos con Jupyter en el apartado anterior. En las siguientes prácticas veremos como convertir un Notebook de Jupyter a un script de Python .py, siendo necesario para su orquestación desde Airflow.

Estos archivos que definen los proceso de Spark con Python se almacenarán en la carpeta *infra/jobs*. Además, en los procesos de Airflow podrá ser necesario ajustar la siguiente variable en función de dónde tengamos ubicada la carpeta *infra* en nuestro equipo local.

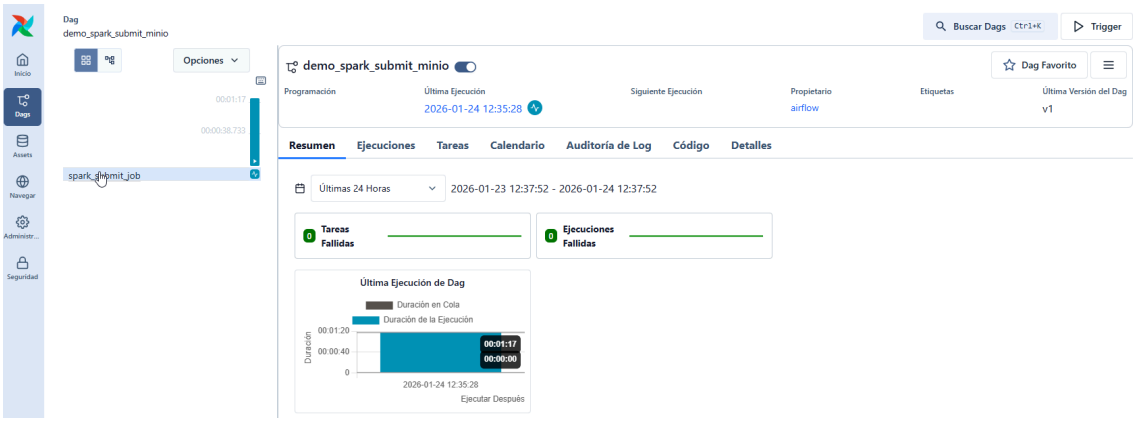
```
# Ajusta si tu ruta difiere
HOST_JOBS_DIR = r"C:\pmvd-lab\infra\datalake\jobs"
```

Tras revisar lo anterior (y ajustar la ruta en caso necesario), desde la interfaz web de Airflow se ejecutará el DAG "demo_spark_submit_minio". Para ello, se hará clic en el botón Play (Activar Dag).



Tras esto seleccionaremos "ejecución única" y haremos clic en "Trigger".

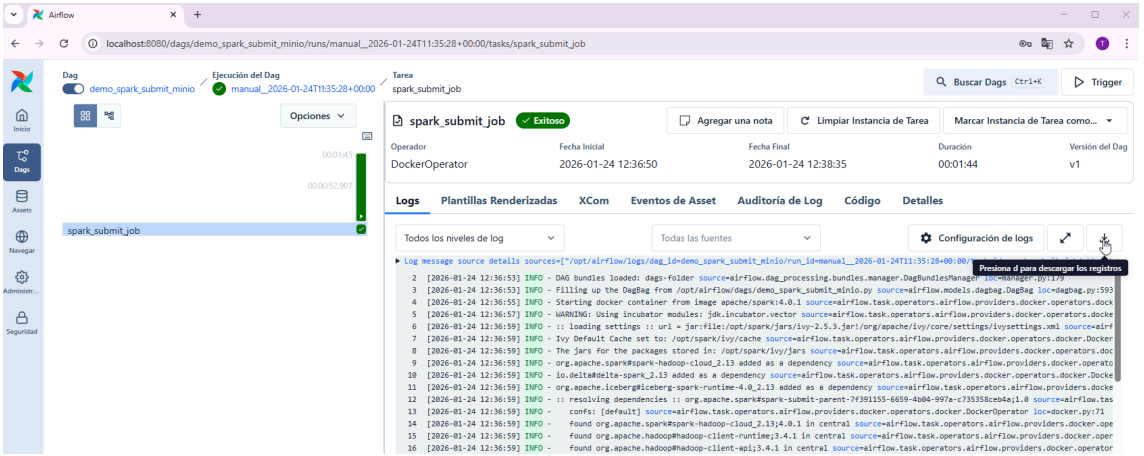
En este momento, comenzará la ejecución del DAG. Se hará clic en el nombre de este para entrar al detalle las ejecuciones, actuales e histórico. Cada DAG se compone de 1 o más Tasks, que se ejecutarán por separado. Podremos acceder al logs las task para revisar en detalla su ejecución en caso necesario.



Para acceder a los logs de una ejecución, dentro del detalle de ese DAG, hemos de:

- Clic en Ejecuciones para ver el listado de ejecuciones. Haremos clic en la ejecución que nos interese para acceder al detalle de monitorización y logs, en este caso la ejecución en curso o terminada.
- Una vez en la ejecución, seleccionaremos la Task que nos interese. En nuestro caso se llama “spark_submit_job”. Si hacemos clic en la misma, se nos mostrará su log, el cual se actualizará en vivo durante su ejecución.

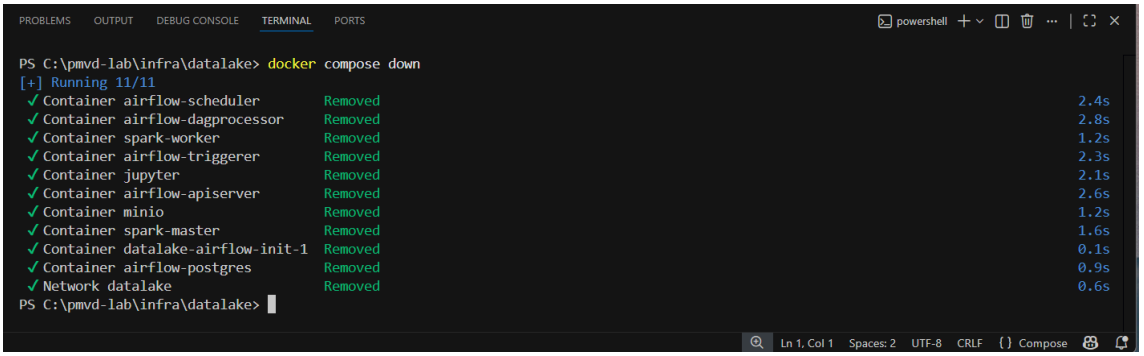
Como parte de la entrega de la práctica se ha de incluir una copia del log de ejecución generado al término de una ejecución exitosa. Se descargará en formato .txt al hacer clic en .



2.4 Apagado del entorno

Para terminar la sesión de forma segura, es necesario detener el entorno con el siguiente comando de Docker:

- **docker** compose down



3. Entrega de la Práctica

La fecha límite de entrega para esta práctica es el **viernes 13 de febrero**, hasta las **23:59.**

Debes entregar un fichero llamado **practica0.zip** que incluya:

- ***00_spark_minio_test2_s3default.ipynb*** : Copia ejecutada con éxito por el alumno, incluyendo obligatoriamente la salida de las celdas (incluyen log).
- **[logs_airflow].txt:** Archivo de logs descargado de Airflow. El nombre logs_airflow se sustituirá por el autogenerado por Airflow, por ejemplo, *logs_demo_spark_submit_minio_manual__2026-01-24T11_35_28+00_0 0_ spark_submit_job_-1_1.txt.*
- **alumno.txt:** Archivo de texto incluyendo:
  - Nombre y apellidos
  - DNI
  - Grupo de prácticas

La entrega se realizará **exclusivamente a través de la página de Moodle de la asignatura.**

4. Evaluación

Dado que esta práctica es guiada, se evaluará de la siguiente forma:

- 0 puntos por no entregar en fecha
- 4 puntos por entrega a la que le faltan partes
- 6 puntos por entrega conforme a lo especificado, pero con errores
- 10 puntos por práctica entregada conforme a lo especificado (sin errores)

Anexo 1: Instalación en tu equipo personal

Es posible la instalación del entorno en tu equipo personal, siempre y cumpla los siguientes requisitos:

- Docker Desktop instalado en la máquina
- Mínimo estimado de 12GB de RAM
- Aunque no es obligatorio, se recomienda también instalar Visual Studio Code como entorno de desarrollo.

Para ello, se descargará el archivo “Software de la asignatura sobre Docker” desde Moodle. A continuación, descomprimir el archivo zip descargado *infra-PMDV-UA.zip* en tu equipo, obteniendo de esta forma las carpetas ***datalake*** y ***tests***.

A partir de este punto se deben seguir los pasos indicados en el apartado **2.1 Arranque del entorno** y siguientes:

- Mover o copiar las 4 carpetas dentro de *tests* en la carpeta *datalake*.
- Abrir una terminal (ej. en Visual Studio), acceder a la ruta donde se encuentre *datalake* y ejecutar el comando ***docker*** *compose up -d*

Cuando se ejecuta el comando anterior la primera vez, se van a descargar y compilar algunas imágenes en Docker, pudiendo llevar hasta 10-20 minutos esa instalación, según las características de tu equipo. Esto solo ocurrirá en la primera ejecución, aunque es posible evitarlo usando la copia de las imágenes Docker ya generadas que se proporcionan para en versión portable, usando el comando docker load -i pmvd-images.tar para cargarlas antes ejecutar el comando compose.

Anexo 2: Versión portable USB

Alternativa para casos en los que el PC de laboratorio no tenga instalado correctamente el entorno o se vaya a usar otro equipo cualquiera de forma puntual. Las diferencias principales con la instalación completa previamente descrita en el Anexo 1 son:

- Se cargarán las imágenes de Docker previamente compiladas desde un USB o disco duro externo (pmvd-images.zip – 3,48 GB comprimido)
- Se usará una carpeta project, que contiene una versión especial de la definición de Docker para que sea más fácil copiarla desde el USB al equipo y viceversa. Ya incluye los ejemplos de test copiados en las correspondientes carpetas (project.zip)
- En el zip anterior se incluye unas instrucciones (.txt) detalladas para su uso.