1. Verificamos que la base de datos tripdate este creada y nos posicionamos en la misma.

```
hive> show databases;
OK
default
tripdata
Time taken: 0.022 seconds, Fetched: 2 row(s)
hive> select current_database();
OK
default
Time taken: 0.071 seconds, Fetched: 1 row(s)
hive> use tripdata;
OK
Time taken: 0.033 seconds
```

Creo la tabla

```
hive> CREATE EXTERNAL TABLE IF NOT EXISTS tripdata.airport_trips (
    >     tpep_pickup_datetime    timestamp
    >     airport_fee             double
    >     payment_type            bigint
    >     tolls_amount            double
    >     total_amount            double
    > )
    > STORED AS PARQUET
    > LOCATION '/user/hive/external/tripdata/airport_trips';
OK
Time taken: 0.093 seconds
```

2. Muestro el contenido

```
hive> DESCRIBE airport_trips;
OK
tpep_pickup_datetime    timestamp
airport_fee             double
payment_type            bigint
tolls_amount            double
total_amount            double
Time taken: -0.663 seconds, Fetched: 5 row(s)
```

3. Verifico que la carpeta de destino existe:

```
hadoop@ab87a6c54b92:~$ hdfs dfs -ls /user/hive/warehouse
Found 3 items
drwxrwxr-x   - hadoop supergroup          0 2022-05-02 14:06 /user/hive/warehouse/emp.db
drwxrwxr-x   - hadoop supergroup          0 2022-05-09 18:00 /user/hive/warehouse/tables
drwxrwxr-x   - hadoop supergroup          0 2022-05-02 13:39 /user/hive/warehouse/tripdata.db
```

Creo el script que va a descargar los archivos:

```
hadoop@ab87a6c54b92:~$ nano ingest_yellow_data.sh
```

```
  GNU nano 4.8                                                ingest_yellow_data.sh
#!/bin/bash

# ================================================
# 1. VARIABLES DE CONFIGURACION
# ================================================

# URL de la fuente de los archivos Parquet
BASE_URL="https://data-engineer-edvai-public.s3.amazonaws.com"

# Lista de archivos a descargar
FILES=(
    "yellow_tripdata_2021-01.parquet"
    "yellow_tripdata_2021-02.parquet"
)

# Directorio de destino en HDFS
HDFS_TARGET_DIR="/tmp/ingest_raw"
```

```
  GNU nano 4.8                                                ingest_yellow_data.sh
# ================================================
# 2. PROCESO DE INGESTA
# ================================================

echo "--- Verificando y creando directorio de HDFS: ${HDFS_TARGET_DIR} ---"

hdfs dfs -mkdir -p ${HDFS_TARGET_DIR}

for FILE_NAME in "${FILES[@]}"; do
    FILE_URL="${BASE_URL}/${FILE_NAME}"

    echo "--- Procesando archivo: ${FILE_NAME} ---"

    # Descargo el archivo de la URL
    echo "Descargando desde: ${FILE_URL}..."
    wget --no-check-certificate "${FILE_URL}" -O "${FILE_NAME}"

    # Verifico si la descarga fue exitosa
    if [ $? -ne 0 ]; then
        echo "ERROR: Fallo la descarga de ${FILE_NAME}. Saliendo."
        exit 1
    fi

    # Cargo el archivo en la ruta de HDFS temporal
    echo "Ingestando ${FILE_NAME} a HDFS..."
    hdfs dfs -put -f "${FILE_NAME}" "${HDFS_TARGET_DIR}"

    # Elimino el archivo local despues de cargarlo a HDFS para ahorrar espacio
    echo "Limpiando archivo local..."
    rm "${FILE_NAME}"
done

echo "--- Ingesta de todos los archivos completada con exito. ---"

# ================================================
# 3. VERIFICACION
# ================================================

echo "--- Verificando archivos finales en HDFS (${HDFS_TARGET_DIR}) ---"
hdfs dfs -ls ${HDFS_TARGET_DIR}
```

Le asigno permisos

```
hadoop@ab87a6c54b92:~$ chmod +x ingest_yellow_data.sh
```

Ingesto los archivos:

```
hadoop@ab87a6c54b92:~$ ./ingest_yellow_data.sh
--- Verificando y creando directorio de HDFS: /tmp/ingest_raw ---
--- Procesando archivo: yellow_tripdata_2021-01.parquet ---
Descargando desde: https://data-engineer-edvai-public.s3.amazonaws.com/yellow_tripdata_2021-01.parquet...
--2025-10-15 23:18:58--  https://data-engineer-edvai-public.s3.amazonaws.com/yellow_tripdata_2021-01.parquet
Resolving data-engineer-edvai-public.s3.amazonaws.com (data-engineer-edvai-public.s3.amazonaws.com)... 52.216.24.12, 52.216.114.27, 52.217.140.177, ...
Connecting to data-engineer-edvai-public.s3.amazonaws.com (data-engineer-edvai-public.s3.amazonaws.com)|52.216.24.12|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 21686067 (21M) [application/vnd.apache.parquet]
Saving to: 'yellow_tripdata_2021-01.parquet'

yellow_tripdata_2021-01.parquet      100%[=====================================================================>]  20.68M  2.75MB/s    in 7.9s

2025-10-15 23:19:07 (2.61 MB/s) - 'yellow_tripdata_2021-01.parquet' saved [21686067/21686067]

Ingestando yellow_tripdata_2021-01.parquet a HDFS...
Limpiando archivo local...
--- Procesando archivo: yellow_tripdata_2021-02.parquet ---
Descargando desde: https://data-engineer-edvai-public.s3.amazonaws.com/yellow_tripdata_2021-02.parquet...
--2025-10-15 23:19:08--  https://data-engineer-edvai-public.s3.amazonaws.com/yellow_tripdata_2021-02.parquet
Resolving data-engineer-edvai-public.s3.amazonaws.com (data-engineer-edvai-public.s3.amazonaws.com)... 52.217.89.156, 16.15.181.131, 16.15.182.152, ...
Connecting to data-engineer-edvai-public.s3.amazonaws.com (data-engineer-edvai-public.s3.amazonaws.com)|52.217.89.156|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 21777258 (21M) [application/vnd.apache.parquet]
Saving to: 'yellow_tripdata_2021-02.parquet'

yellow_tripdata_2021-02.parquet      100%[=====================================================================>]  20.77M  1.13MB/s    in 13s

2025-10-15 23:19:23 (1.59 MB/s) - 'yellow_tripdata_2021-02.parquet' saved [21777258/21777258]

Ingestando yellow_tripdata_2021-02.parquet a HDFS...
Limpiando archivo local...
--- Ingesta de todos los archivos completada con exito. ---
--- Verificando archivos finales en HDFS (/tmp/ingest_raw) ---
Found 2 items
-rw-r--r--   1 hadoop supergroup   21686067 2025-10-15 23:19 /tmp/ingest_raw/yellow_tripdata_2021-01.parquet
-rw-r--r--   1 hadoop supergroup   21777258 2025-10-15 23:19 /tmp/ingest_raw/yellow_tripdata_2021-02.parquet
```

4. Creo el script:

```
  GNU nano 4.8                                                    airport.py
# -*- coding: utf-8 -*-
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
import subprocess
import sys
# ================================================
# 1. Inicializar la sesion de Spark
# ================================================
spark = SparkSession.builder \
      .appName("IngestaAirportTrips") \
      .config("spark.sql.legacy.timeParserPolicy", "LEGACY") \
      .enableHiveSupport() \
      .getOrCreate()
# ================================================
# 2. DEFINICION DE RUTAS Y CONFIGURACION
# ================================================
BASE_PATH = "/tmp/ingest_raw/*"
HIVE_TABLE = "tripdata.airport_trips"
AIRPORT_LOCATION_IDS = [1, 132, 138]
# ================================================
# 3. EXTRACCION (E) Y UNION DE DATOS
# ================================================

print("-> Leyendo y uniendo archivos Parquet...")

# Cargar los archivos Parquet
df_trip = spark.read.parquet(BASE_PATH)
```

En BASE_PATH al usar * al final, ya hdfs hace unión de las tablas.

```
  GNU nano 4.8                                                    airport.py
# ================================================
# 4. TRANSFORMACION (T)
# ================================================

print("-> Aplicando filtros (Aeropuerto y Efectivo)...")

df_filtrado = df_trip.filter(
    (
        (col("PULocationID").isin(AIRPORT_LOCATION_IDS)) |
        (col("DOLocationID").isin(AIRPORT_LOCATION_IDS))
    )&
    (col("payment_type") == 2)
)

# --- Seleccion de Columnas Finales ---
print("-> Seleccionando columnas finales...")

df_final = df_filtrado.select(
    col("tpep_pickup_datetime"),
    col("airport_fee"),
    col("payment_type"),
    col("tolls_amount"),
    col("total_amount")
)

df_final.printSchema()
print(f"Total de registros a insertar: {df_final.count()}")
```

```
# ================================================
# 5. CARGA (L) - Insertar en Hive
# ================================================
try:
        print(f"-> Insertando datos en la tabla Hive: {HIVE_TABLE}...")
        df_final.write \
            .mode("overwrite") \
            .insertInto(HIVE_TABLE)

        print("--- Insercion completada con exito. ---")
except Exception as e:
        print(f"ERROR: Fallo la insercion en Hive. No se intentara la limpieza. Error: {e}")
        spark.stop()
        sys.exit(1)
```

Luego busque hacer un cambio para que, si yo ejecutaba de nuevo el archivo.py hive no siga duplicando datos, y solo sobrescriba los que ya tenía. Ayuda IA:

```
# ================================================
# 5. CARGA (L) - Insertar en Hive
# ================================================
HIVE_PATH = "/user/hive/warehouse/tripdata.db/airport_trips"
try:
        print(f"-> Insertando datos en la tabla Hive: {HIVE_TABLE}...")
        df_final.write \
            .mode("overwrite") \
            .format("parquet") \
            .option("path", HIVE_PATH) \
            .saveAsTable(HIVE_TABLE)
        print("--- Insercion completada con exito. ---")
except Exception as e:
        print(f"ERROR: Fallo la insercion en Hive. No se intentara la limpieza. Error: {e}")
        spark.stop()
        sys.exit(1)
```

La IA me recomendó agregar esta Limpieza, ya que anteriormente tuve problemas con la acumulación de datos en la carpeta.

```
# ==========================================
# 6. LIMPIEZA: Eliminar datos temporales de HDFS
# ==========================================
CLEAN_PATH = BASE_PATH.replace("/*", "")
print(f"-> Limpiando directorio temporal HDFS: {CLEAN_PATH}")
try:
        command = ["hdfs", "dfs", "-rm", "-r", "-skipTrash",CLEAN_PATH]

        result = subprocess.run(
        command,
        check=True,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        text=True
        )

        print(f"--- Limpieza completada con exito. Salida: {result.stdout.strip()}")

except subprocess.CalledProcessError as e:
        print(f"ADVERTENCIA: Fallo la limpieza de HDFS (esto puede ser normal si el directorio ya estaba vacio). Error: {e.stderr.strip()}")

except FileNotFoundError:
        print("ADVERTENCIA: El comando 'hdfs' no se encontro. No se pudo limpiar el directorio temporal.")
finally:
        spark.stop()
```

Otra opción de filtrado:

```
df_filtrado = df_trip.filter(
    (
        col("RatecodeID").isin(2, 3) |
        (col("airport_fee").isNotNull() & (col("airport_fee") > 0))
        ) &
    (col("payment_type") == 2)
)
```

Lo ejecuto de esta forma porque con spark-submit process_spark.py al ejecutar se quedaba cargando y nunca terminaba, de esta forma le doy mas poder.

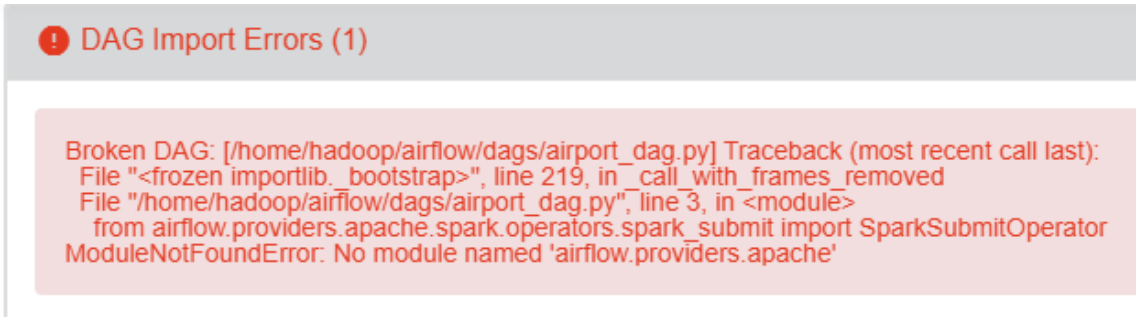
```
hadoop@ab87a6c54b92:~$ spark-submit \
>      --master local[2] \
>      --driver-memory 1g \
>      --executor-memory 2g \
>      --conf spark.sql.shuffle.partitions=100 \
>      process_spark.py
```

Y luego corroboramos en HIVE:

```
hive> SELECT * FROM tripdata.airport_trips LIMIT 10;
OK
2021-01-31 21:33:38     NULL    2       0.0     22.3
2021-01-31 21:24:54     NULL    2       0.0     32.3
2021-01-31 21:39:37     NULL    2       0.0     25.8
2021-01-31 21:31:34     NULL    2       0.0     23.3
2021-01-31 20:56:33     NULL    2       0.0     16.8
2021-01-31 21:33:02     NULL    2       0.0     23.3
2021-01-31 21:31:01     NULL    2       0.0     27.3
2021-01-31 21:22:17     NULL    2       6.12    64.92
2021-01-31 21:36:44     NULL    2       0.0     16.8
2021-01-31 21:26:27     NULL    2       0.0     31.8
Time taken: 0.098 seconds, Fetched: 10 row(s)
```

5. Eso intente al principio, pero tenia un error al cargar:

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.providers.apache.spark.operators.spark_submit import SparkSubmitOperator
from datetime import datetime, timedelta
from airflow.operators.empty import EmptyOperator
```



Tuve que modificar ingest_yellow_data porque airflow no leia bien el archivo.sh

```
  GNU nano 4.8                                                  ingest_yellow_data.sh
#!/bin/bash
export HADOOP_HOME=/home/hadoop/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
# ================================================
# 1. VARIABLES DE CONFIGURACION
# ================================================
```

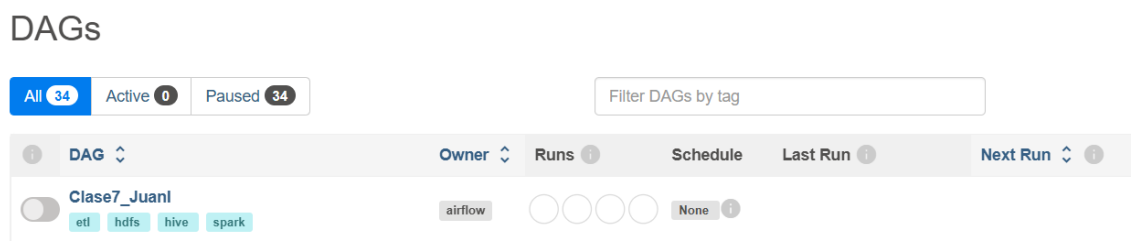
Lo mismo con airport.py, tuve que agregar la ip y el puerto a la ruta para que lo lea.

DAG:

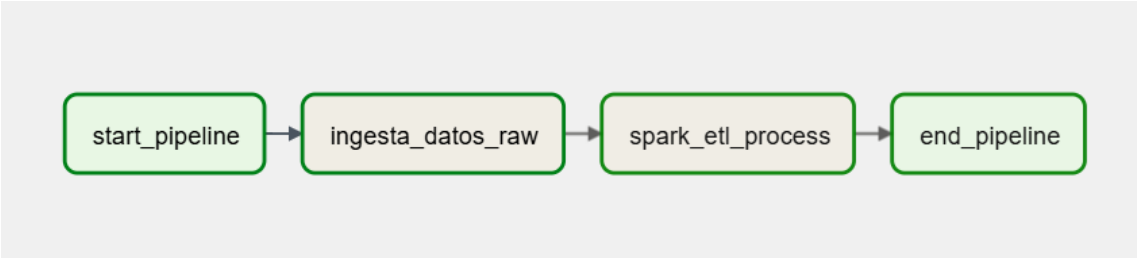
```
  GNU nano 4.8                                                  airport_dag.py
from airflow import DAG
from airflow.operators.bash import BashOperator
#from airflow.providers.apache.spark.operators.spark_submit import SparkSubmitOperator
from datetime import datetime, timedelta
from airflow.operators.empty import EmptyOperator

default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'start_date': datetime(2025, 1, 1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}

with DAG(
    dag_id='Clase7_JuanI',
    default_args=default_args,
    description='Flujo de Ingesta (Bash) y ETL con Bash (Transformacion y SOBRESCRITURA en Hive).',
    schedule_interval=None,
    catchup=False,
    tags=['ingest', 'transform'],
) as dag:
        start_pipeline = EmptyOperator(
                task_id='start_pipeline',
        )
```

```
ingesta_datos_raw = BashOperator(
        task_id='ingesta_datos_raw',
        bash_command="bash -c 'bash /home/hadoop/airflow/dags/scripts/ingest_yellow_data.sh'",
)

spark_etl_process = BashOperator(
        task_id='spark_etl_process',
        bash_command=f"""
                ssh hadoop@172.17.0.2 /home/hadoop/spark/bin/spark-submit \
                --files /home/hadoop/hive/conf/hive-site.xml \
                /home/hadoop/airflow/dags/scripts/airport.py
        """,
)

end_pipeline = EmptyOperator(
task_id='end_pipeline',
)

start_pipeline >> ingesta_datos_raw >> spark_etl_process >> end_pipeline
```

Se cargó el DAG



Lo ejecuto:



Para comprobar que todo funcionaba, borre la carpeta en hive, y si todo estaba OK se iba a volver a crear, y asi fue.