Index

[Create a folder to organize the project 2](#_Toc162811703)

[Creating the yaml Document 2](#_Toc162811704)

[Access the Airflow graphical interface 3](#_Toc162811705)

[Creating connections to databases 3](#_Toc162811706)

[First connection - POSTGRESQL 4](#_Toc162811707)

[Second Connection - MySQL 5](#_Toc162811708)

[Verification and Saving of connections 7](#_Toc162811709)

[Creation of the DAG. 7](#_Toc162811710)

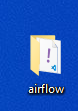
[Checking within databases 12](#_Toc162811711)

[Mysql 12](#_Toc162811712)

[PostgreSQL 13](#_Toc162811713)

[Annexes 13](#_Toc162811714)

## Create a folder to organize the project



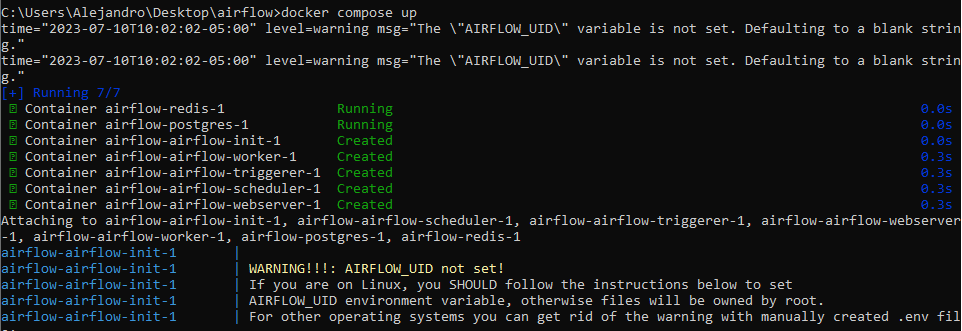
## Creating the yaml Document

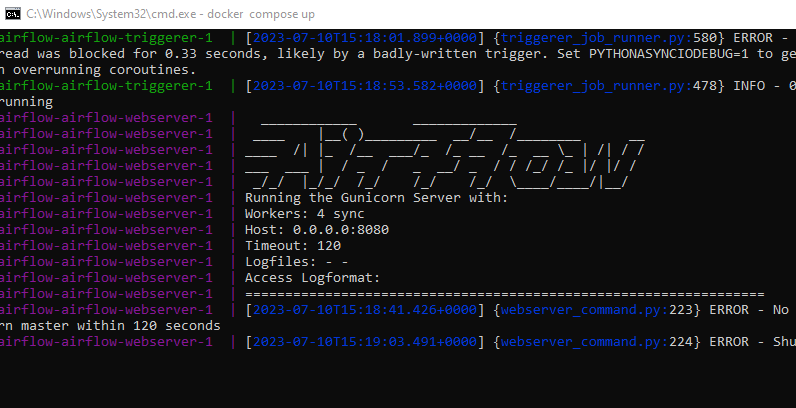
<https://airflow.apache.org/docs/apache-airflow/2.6.2/docker-compose.yaml>

With the previous link, copy all its content and save it in a document with the name docker-compose-yaml.

run all airflow

docker compose up

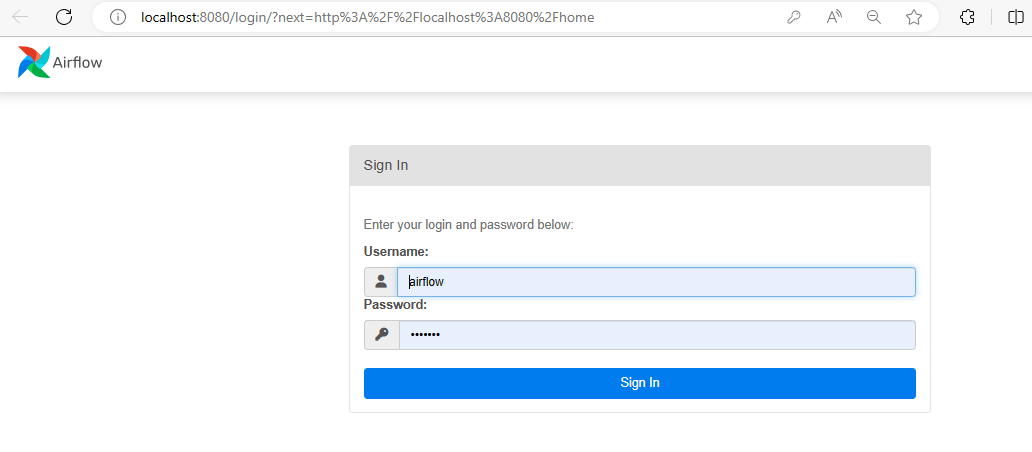


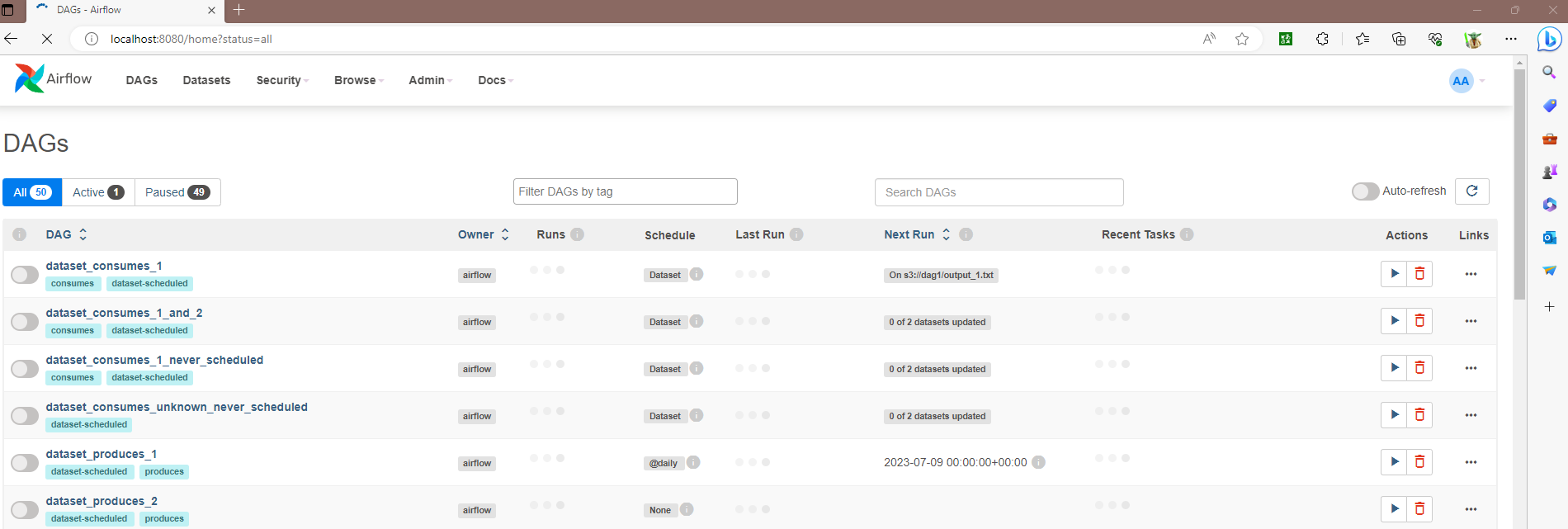


## Access the Airflow graphical interface

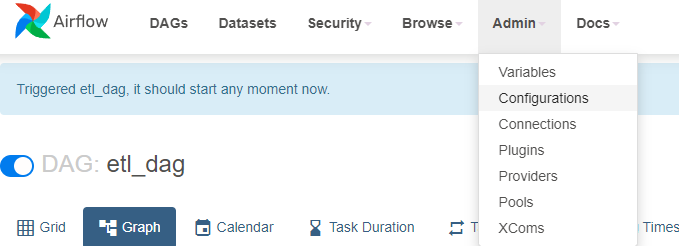
With the following link we can enter through the browser: [localhost](http://localhost:8080/)

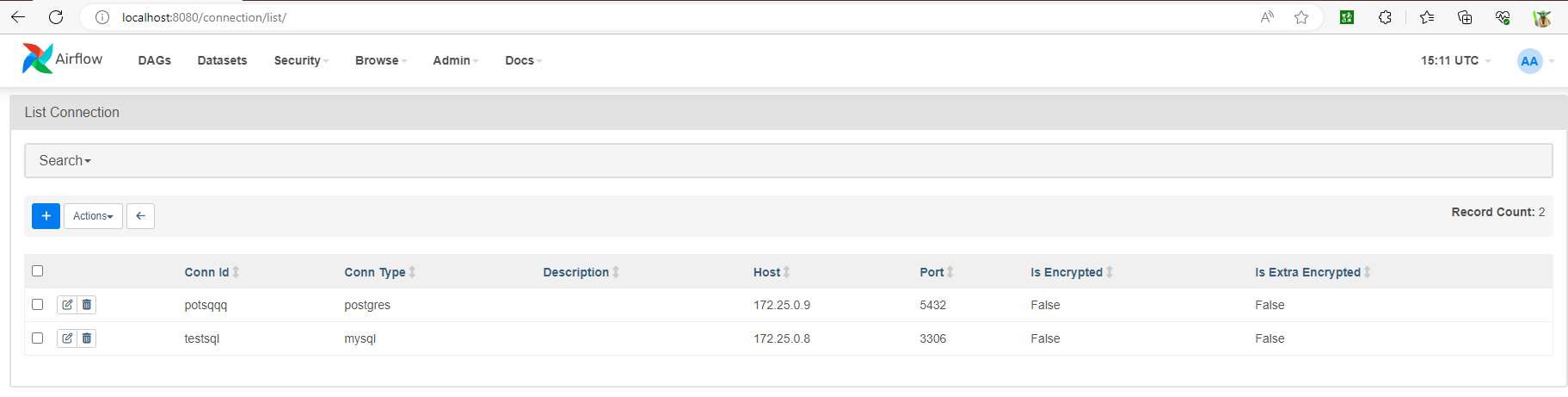
The username and password are airflow





## Creating connections to databases

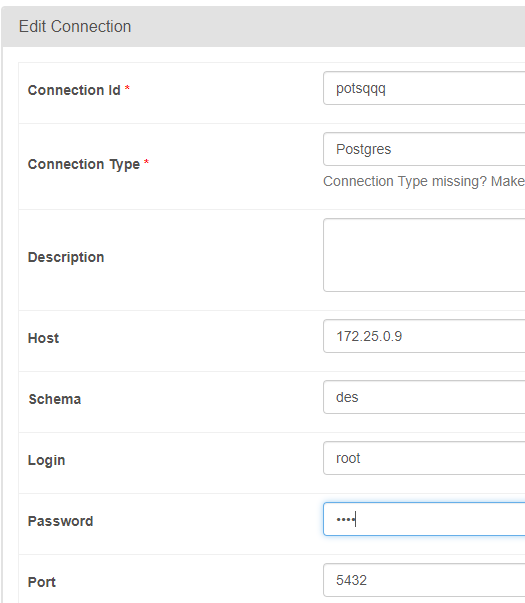
In the upper menu several options are shown, in the Admin option there is also an option called Connections, in that option we do not heal and we will click.



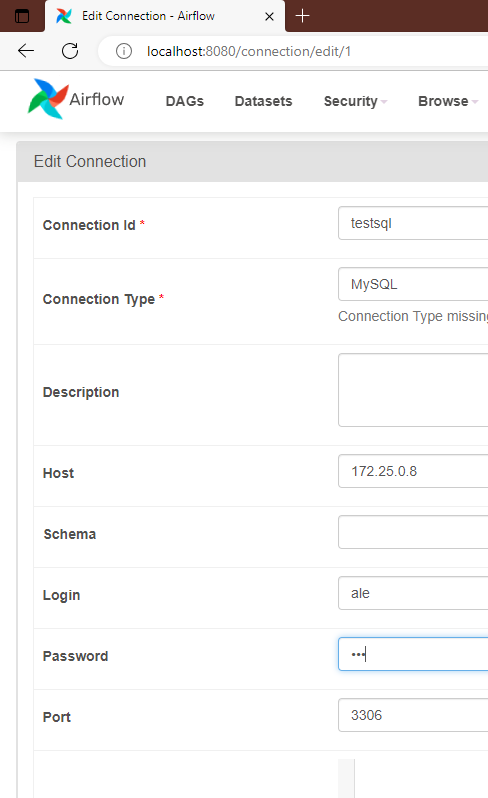
We will click on the button with the plus symbol to add a connection

### First connection - POSTGRESQL

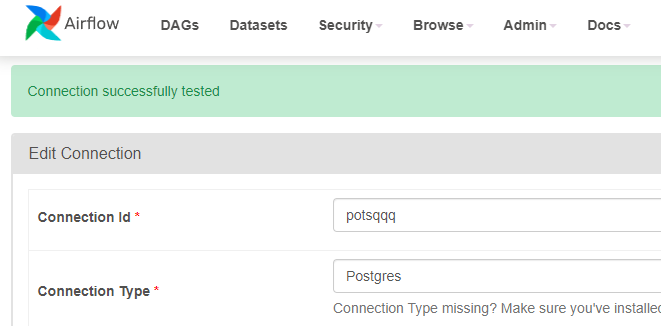
In the name of Connection Id, it can be given whatever we want, what must be taken into account is that all the containers must be in the same network, in this example the same network with which the containers were created was taken. airflow, so it was only necessary to add the mysql and postgres containers.

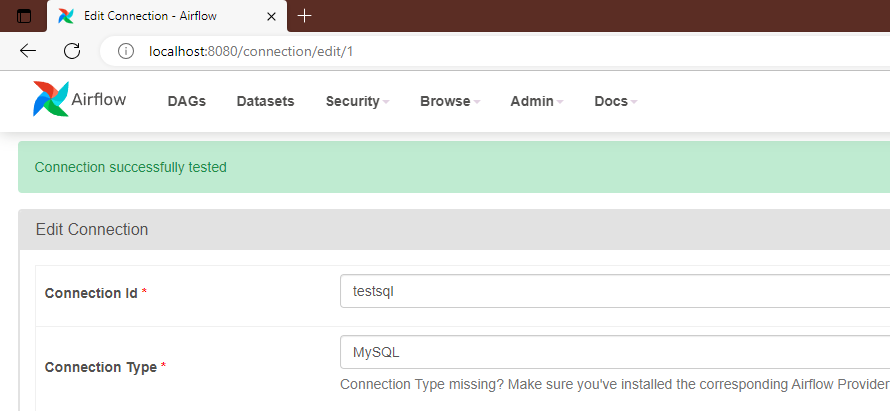


### Second Connection - MySQL

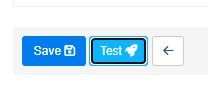


### Verification and Saving of connections





With the Test button you can verify if there is a connection with the database, once verified we will proceed to click on the Save button before exiting.



## Creation of the DAG.

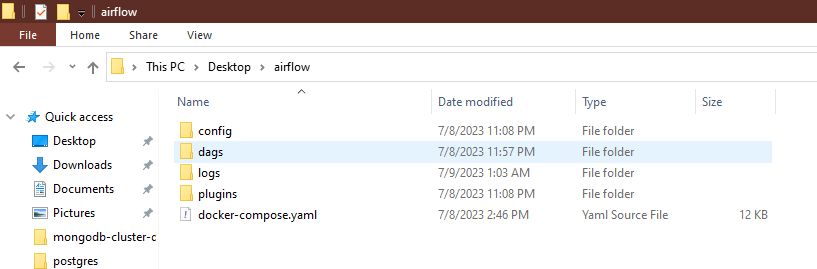
In the folder that was created at the beginning, 4 .

● ./dags - you can put your DAG files here.

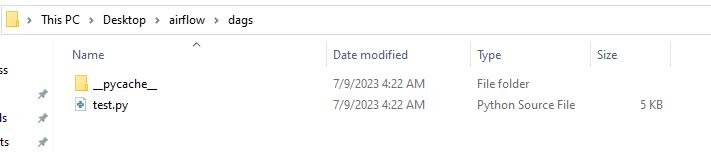
● ./logs – Contains logs of task execution and scheduler.

● ./config – You can add custom log parser or add airflow\_local\_settings.py to configure cluster policy.

● ./plugins - you can put your custom plugins here



Inside the dags folder the new dags were created, these will be python files.



In this example an etl will be performed extracting the data from mysql to postgres

from datetime import datetime

from airflow import DAG

from airflow.operators.python\_operator import PythonOperator

from airflow.hooks.mysql\_hook import MySqlHook

import pandas as pd

from sklearn.decomposition import PCA

from sqlalchemy import create\_engine

from sqlalchemy import create\_engine

from airflow.hooks.postgres\_hook import PostgresHook

def extract():

# Extracción de datos desde una base de datos MySQL utilizando la conexión configurada en Airflow

hook = MySqlHook(mysql\_conn\_id='testsql')

query = 'SELECT \* FROM medata.tu\_tabla'

df = hook.get\_pandas\_df(query)

return df

def transform(df):

df['YearStart'] = df['YearStart'].astype(str)

df['YearEnd'] = df['YearEnd'].astype(str)

# Convertir las cadenas de texto en columnas 'YearStart' y 'YearEnd' a objetos datetime

df['YearStart'] = df['YearStart'].apply(lambda x: datetime.strptime(x, '%Y'))

df['YearEnd'] = df['YearEnd'].apply(lambda x: datetime.strptime(x, '%Y'))

# Calcular la duración en días

df['Duration'] = (df['YearEnd'] - df['YearStart']).dt.days

# Transformación de los datos

df[['Latitude', 'Longitude']] = df['GeoLocation'].str.strip('POINT ()').str.split(expand=True).astype(float)

# Puedes eliminar las claves originales que ya no necesitas

df.pop('YearStart')

df.pop('YearEnd')

df.pop('GeoLocation')

df.pop('DatavalueFootnote')

return df

def apply\_pca(df):

df.replace("", pd.NA, inplace=True)

# Aplicar PCA a los datos transformados

features = ['Latitude', 'Longitude']

X = df[features]

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

# Añadir las componentes principales al DataFrame transformado

df['PCA1'] = X\_pca[:, 0]

df['PCA2'] = X\_pca[:, 1]

# df['DataValueUnit'] = pd.to\_numeric(df['DataValueUnit'], errors='coerce')

# df['DataValueFootnoteSymbol'] = df['DataValueFootnoteSymbol'].replace('-', float('nan'))

empty\_columns = df.columns[df.isnull().all()].tolist()

# Eliminar columnas sin datos

df.dropna(axis=1, how='all', inplace=True)

# Eliminar filas con valores faltantes

df.dropna(axis=0, how='all', inplace=True)

# # Rellenar los valores faltantes con la media de la columna

df['DataValue'].fillna(df['DataValue'].mean(), inplace=True)

df['DataValueAlt'].fillna(df['DataValueAlt'].mean(), inplace=True)

# # Rellenar los valores faltantes con la mediana de la columna

# df.fillna(df.median(), inplace=True)

# # Rellenar los valores faltantes con un valor específico

# df.fillna(0, inplace=True)

df.drop('DataValueUnit', axis=1, inplace=True)

df.drop('DataValueFootnoteSymbol', axis=1, inplace=True)

return df

def load\_transformed\_data(df):

conn = PostgresHook(postgres\_conn\_id='potsqqq')

engine = create\_engine(conn.get\_uri())

# Nombre de la tabla de destino en la base de datos PostgreSQL

table\_name = 'transformed\_table'

# Carga los datos en la tabla de la base de datos

df.to\_sql(table\_name, engine, if\_exists='replace', index=False)

# Definición del DAG

with DAG(dag\_id="etl\_dag",

schedule\_interval=None,

start\_date=datetime(2023, 7, 8),

catchup=False) as dag:

# Tarea de extracción

extract\_task = PythonOperator(

task\_id="extract\_data",

python\_callable=extract,

dag=dag

)

# Tarea de transformación

transform\_task = PythonOperator(

task\_id="transform\_data",

python\_callable=transform,

op\_args=[extract\_task.output],

provide\_context=True,

dag=dag

)

# Tarea de aplicar PCA

pca\_task = PythonOperator(

task\_id="apply\_pca",

python\_callable=apply\_pca,

op\_args=[transform\_task.output],

provide\_context=True,

dag=dag

)

# Tarea de carga

load\_task = PythonOperator(

task\_id="load\_data",

python\_callable=load\_transformed\_data,

op\_args=[pca\_task.output],

provide\_context=True,

dag=dag

)

# Definición del flujo de ejecución

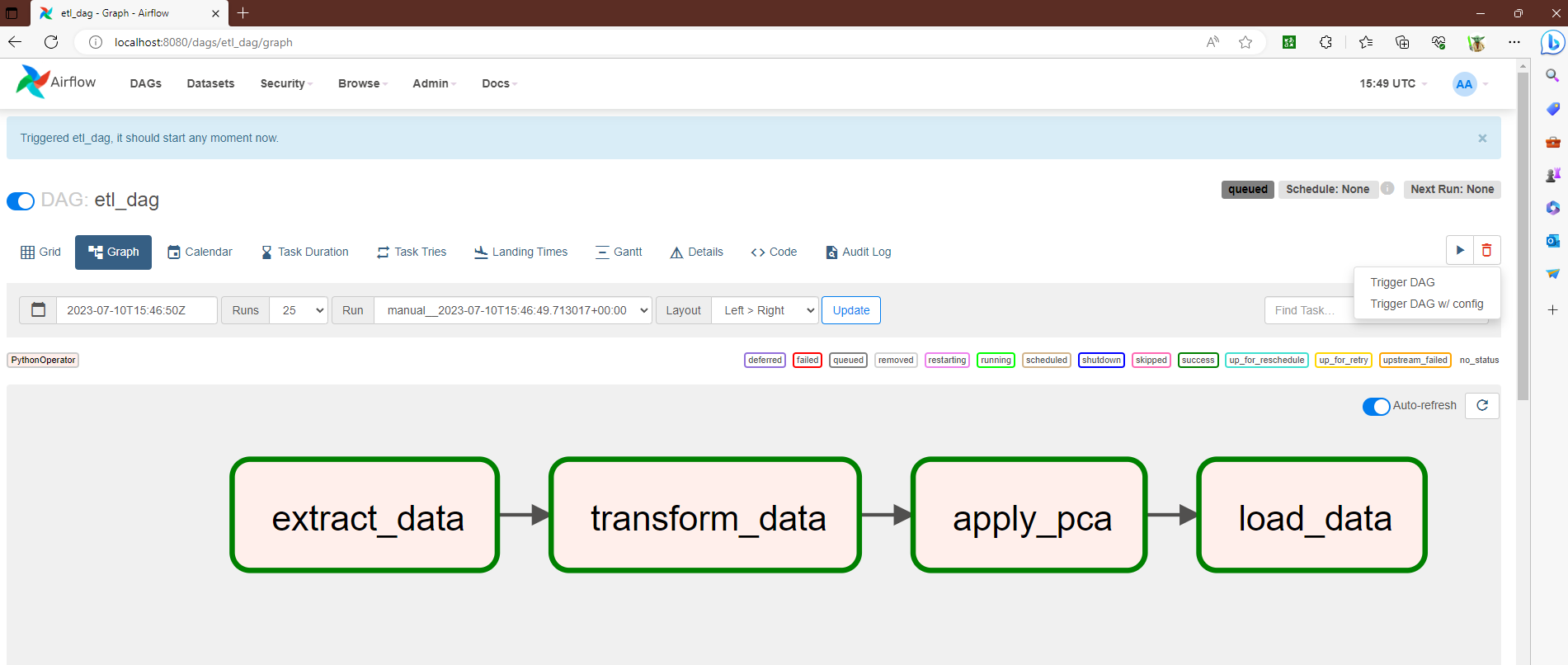
extract\_task >> transform\_task >> pca\_task >> load\_task

Once finished, it is saved and must be executed from the graphical console in the browser.



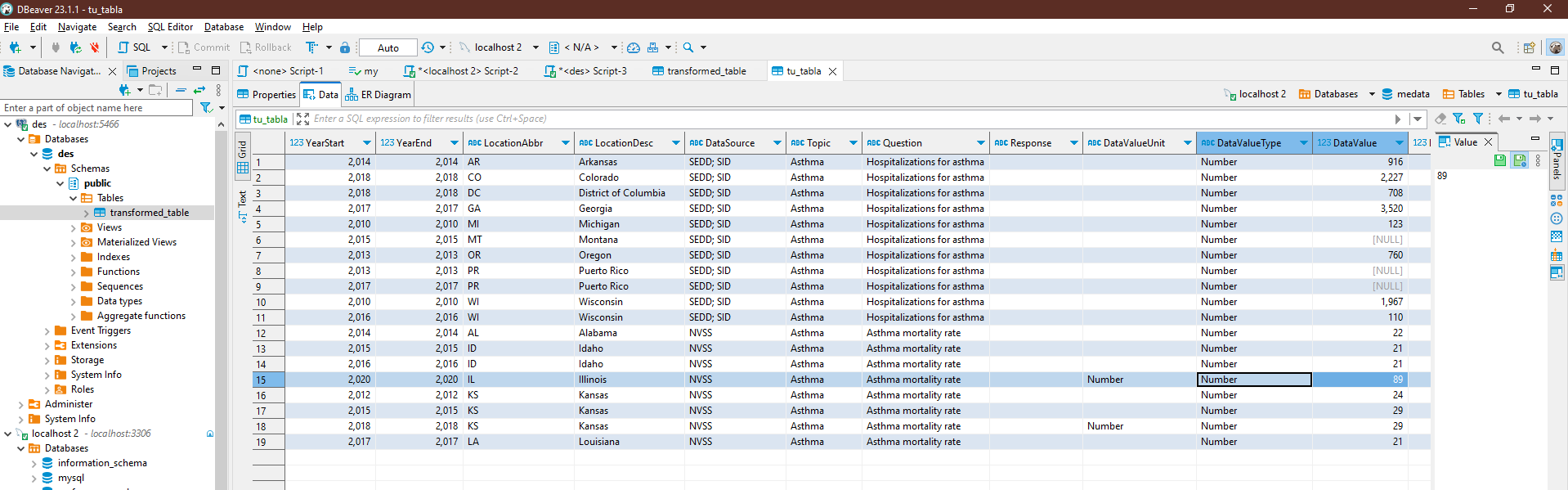
In case the dag does not appear automatically, it is recommended to start the airflow service again.

Inside the dag I can run it with the play button and it takes it to the first option, once this is done the dag will be executed and you can see it in the Graph part of its process

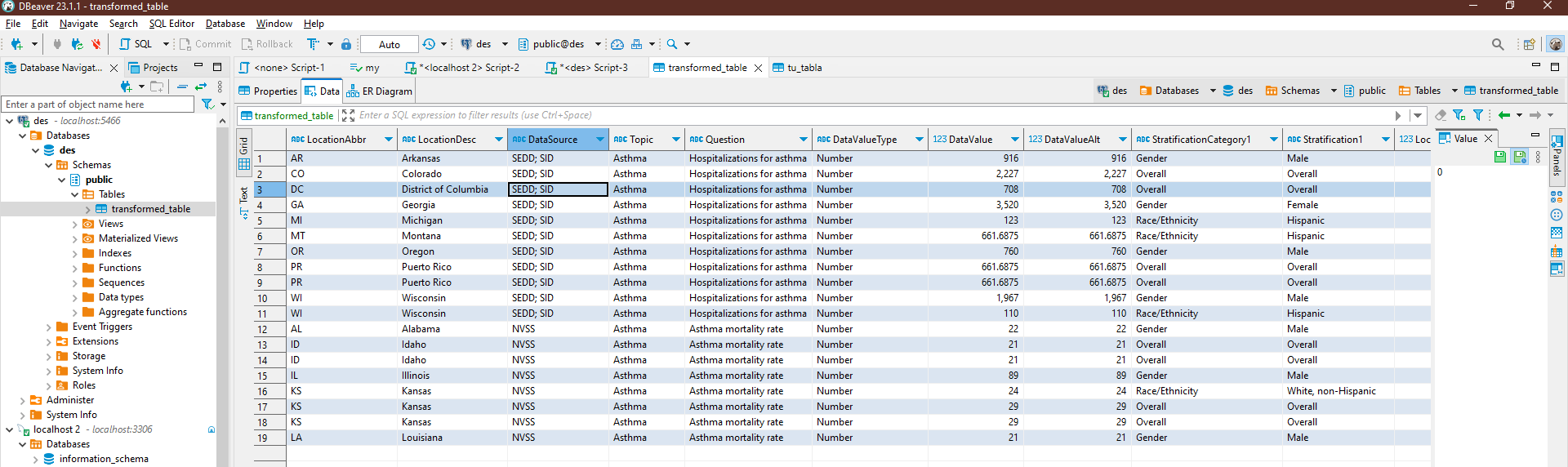


## Checking within databases

### Mysql



### PostgreSQL



## Annexes

Commands for network connection

docker inspect airflow-airflow-webserver-1 (to know which is the id of the network where the container is connected)

docker network connect [networkId] [container name to connect]

docker inspect [networkId] (to know which ips were assigned)

Raise postgres service

docker-compose.yml

version: '3.8'

services:

db:

container\_name: pg\_container

image: postgres:latest

restart: always

environment:

POSTGRES\_USER: root

POSTGRES\_PASSWORD: root

POSTGRES\_DB: des

ports:

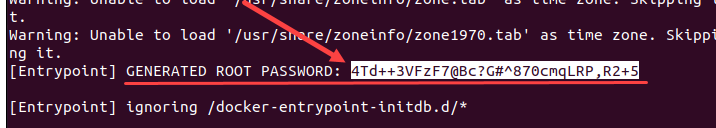
- "5466:5432"

Start MySQL service

docker pull mysql/mysql-server:latest

docker run --name=[container\_name] -d mysql/mysql-server:latest

docker logs [container\_name]



docker exec -it [container\_name] bash

mysql -u root -p

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY '[newpassword]';

CREATE USER 'new\_user'@'localhost' IDENTIFIED BY 'password';

GRANT ALL PRIVILEGES ON \* . \* TO 'new\_user'@'localhost';

FLUSH PRIVILEGES;