Tejashree G
tejashreeg.ai2021@dce.edu.in

Building Data Pipeline in Airflow

This document describes the process of building a simple data pipeline using Apache Airflow. The pipeline retrieves external data, loads it into a Postgres database, and then cleans and merges the data into a final table.

By the end of the process, the pipeline is able to:

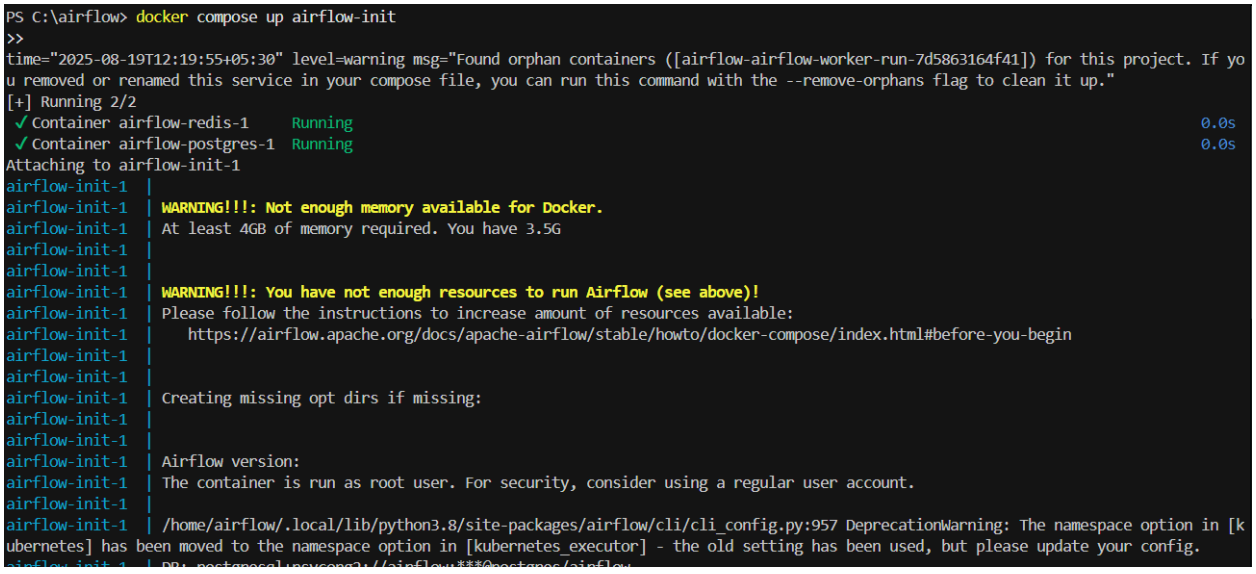
- Download a CSV file from GitHub
- Load the raw data into a staging table (employees_temp)
- Clean and upsert the data into a final table (employees)

This exercise demonstrates Airflow's features such as DAG authoring, SQL operators, hooks, and connections in practice.

Initial Setup

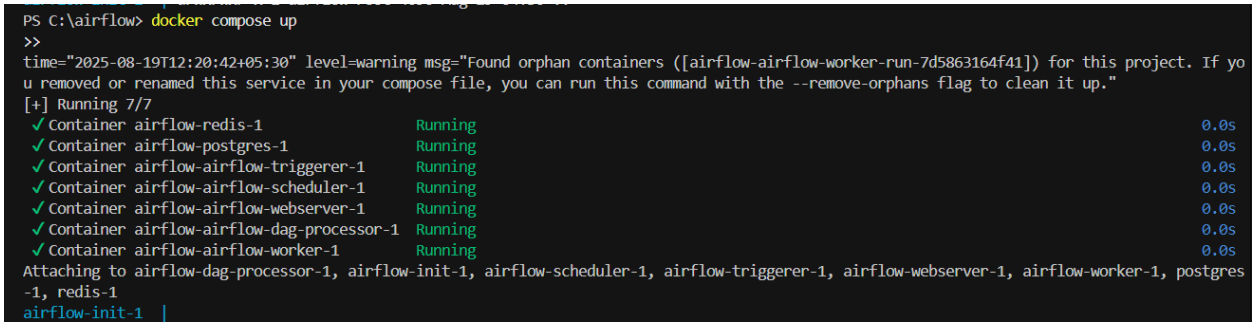
Step 1: Initialize the database

docker compose up airflow-init



Step 2: Start all services

docker compose up



Once the setup was complete, the Airflow UI was accessed at http://localhost:8080 with the following credentials:

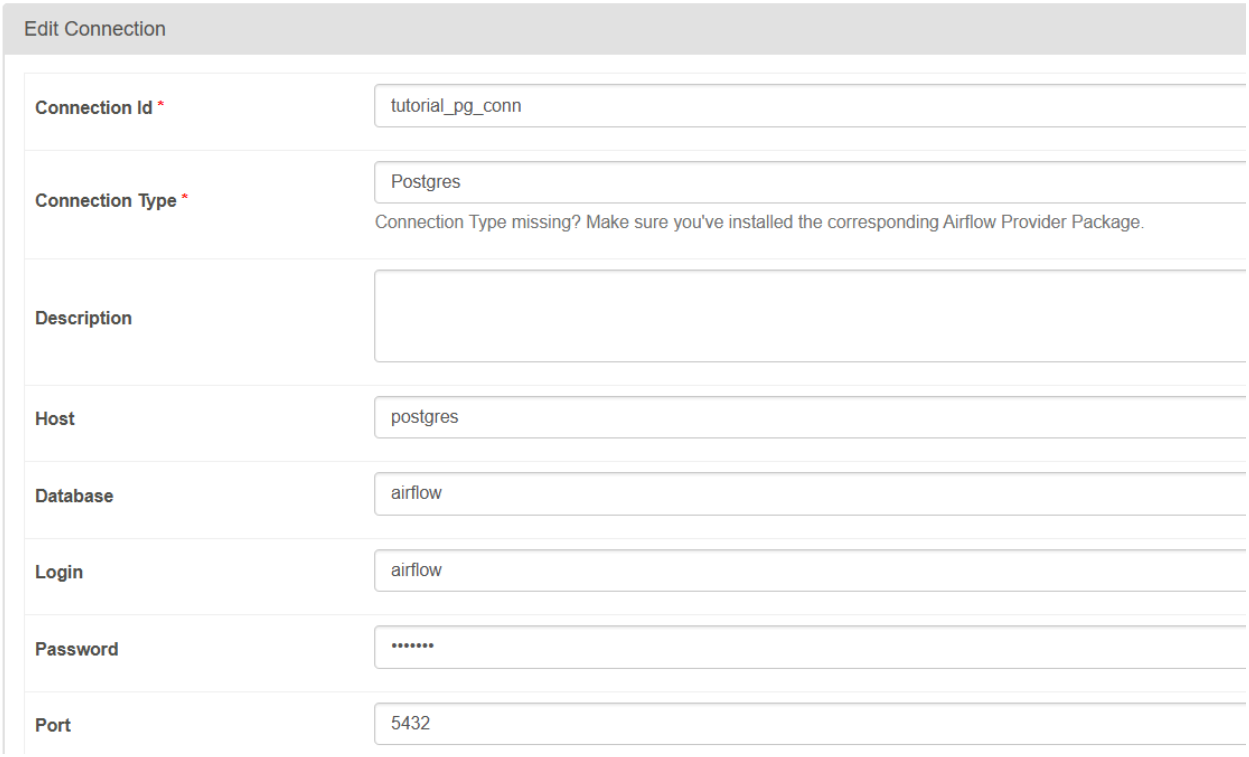
- Username: airflow
- Password: airflow

Creating a Postgres Connection

Before writing data to Postgres, a connection was configured in the Airflow UI under Admin > Connections with the following details:

- Connection ID: tutorial_pg_conn
- Type: Postgres
- Host: postgres
- Database: airflow
- Login/Password: airflow / airflow
- Port: 5432

This connection allowed Airflow to communicate with the Postgres database running inside Docker.



Defining the DAG

The tasks were combined into a DAG called process_employees, with the following workflow:

1. Create staging and final tables
2. Download and load data into the staging table
3. Merge cleaned data into the final table

The DAG definition was saved as dags/process_employees.py.

```
import datetime
import pendulum
import os
import requests

from airflow.decorators import dag, task
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.providers.common.sql.operators.sql import
SQLExecuteQueryOperator

@dag(
    dag_id="process_employees",
```

```
    schedule="0 0 * * *",
    start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
    catchup=False,
    dagrun_timeout=datetime.timedelta(minutes=60),
)
def ProcessEmployees():
    create_employees_table = SQLExecuteQueryOperator(
        task_id="create_employees_table",
        conn_id="tutorial_pg_conn",
        sql="""
            CREATE TABLE IF NOT EXISTS employees (
                "Serial Number" NUMERIC PRIMARY KEY,
                "Company Name" TEXT,
                "Employee Markme" TEXT,
                "Description" TEXT,
                "Leave" INTEGER
            );""",
    )

    create_employees_temp_table = SQLExecuteQueryOperator(
        task_id="create_employees_temp_table",
        conn_id="tutorial_pg_conn",
        sql="""
            DROP TABLE IF EXISTS employees_temp;
            CREATE TABLE employees_temp (
                "Serial Number" NUMERIC PRIMARY KEY,
                "Company Name" TEXT,
                "Employee Markme" TEXT,
                "Description" TEXT,
                "Leave" INTEGER
            );""",
    )

    @task
    def get_data():
        # NOTE: configure this as appropriate for your airflow environment
        data_path = "/opt/airflow/dags/files/employees.csv"
        os.makedirs(os.path.dirname(data_path), exist_ok=True)
```

```
        url =
"https://raw.githubusercontent.com/apache/airflow/main/airflow-core/docs/t
utorial/pipeline_example.csv"

        response = requests.request("GET", url)

        with open(data_path, "w") as file:
            file.write(response.text)

        postgres_hook = PostgresHook(postgres_conn_id="tutorial_pg_conn")
        conn = postgres_hook.get_conn()
        cur = conn.cursor()
        with open(data_path, "r") as file:
            cur.copy_expert(
                "COPY employees_temp FROM STDIN WITH CSV HEADER DELIMITER
AS ',' QUOTE '\"'",
                file,
            )
        conn.commit()

    @task
    def merge_data():
        query = """
            INSERT INTO employees
            SELECT *
            FROM (
                SELECT DISTINCT *
                FROM employees_temp
            ) t
            ON CONFLICT ("Serial Number") DO UPDATE
            SET
              "Employee Markme" = excluded."Employee Markme",
              "Description" = excluded."Description",
              "Leave" = excluded."Leave";
        """
        try:
            postgres_hook =
PostgresHook(postgres_conn_id="tutorial_pg_conn")
            conn = postgres_hook.get_conn()
            cur = conn.cursor()
```

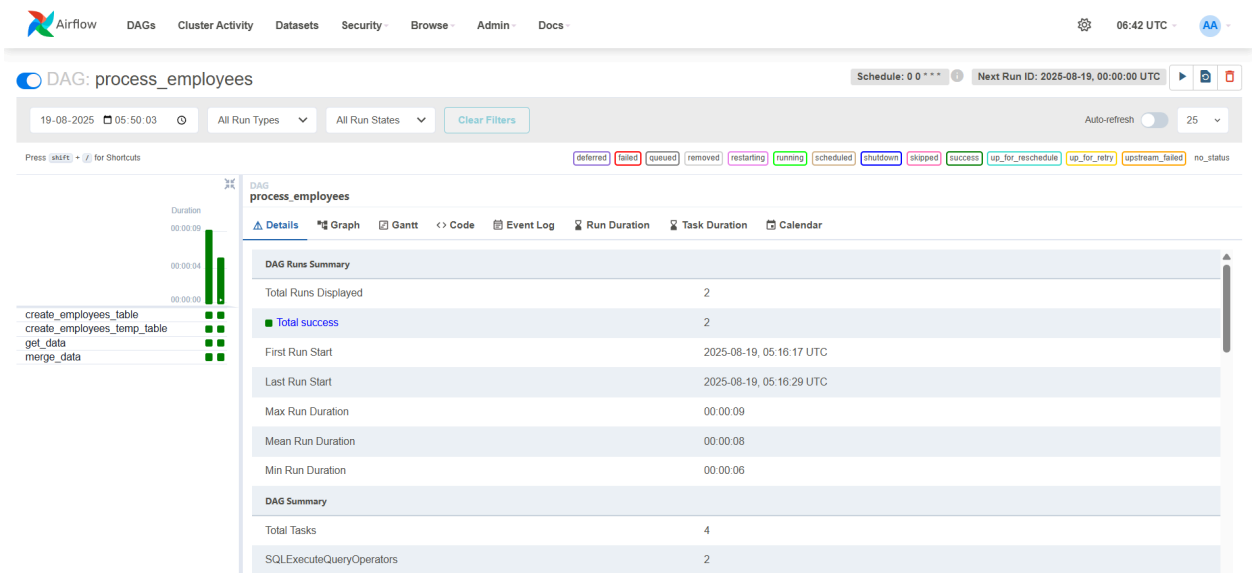
```
            cur.execute(query)
            conn.commit()
            return 0
        except Exception as e:
            return 1

    [create_employees_table, create_employees_temp_table] >> get_data() >>
merge_data()


dag = ProcessEmployees()
```

Running the DAG

Once the DAG was saved, it appeared in the Airflow UI. The DAG was triggered manually, and the pipeline successfully:

- Downloaded the CSV file
- Inserted the data into the staging table
- Merged and cleaned the data into the final table

This confirmed that the data pipeline was functioning correctly end-to-end.



DAG
process_employees

Details | Graph | Gantt | Code | Event Log | Run Duration | Task Duration | Calendar

DAG Details	
Dag display name	process_employees
Dag id	process_employees
Description	null
Fileloc	/opt/airflow/dags/process_employees.py
Has import errors	false
Has task concurrency limits	false
Is active	true
Is paused	false
Is subdag	false
Last expired	null

DAG
process_employees

Details | Graph | Gantt | Code | Event Log | Run Duration | Task Duration | Calendar

End date	null
Is paused upon creation	null
Last parsed	2025-08-19T05:43:06.000644+00:00
Orientation	LR
Render template as native obj	false
Start date	2021-01-01T00:00:00+00:00
Template searchpath	null
Timezone	UTC
Owners	
Tags	No tags
Schedule interval	0 0 * * *
Dag run timeout	3600 seconds