Coding Assessment
Building a Simple Data Pipeline with Apache Airflow

Introduction

Apache Airflow is a workflow orchestration tool designed to automate, schedule, and monitor complex data processes. Instead of manually running scripts, Airflow lets you organize them into workflows called **DAGs (Directed Acyclic Graphs)**. It is widely used in data engineering for ETL pipelines, analytics, and machine learning tasks.

Core Features of Airflow

1. **DAG-based workflows**
   - Workflows are represented as DAGs, where tasks are connected in a clear order without loops.
2. **Powerful Scheduling**
   - You can schedule jobs using CRON expressions, intervals, or manual triggers.
   - Missed jobs can be backfilled easily.
3. **Extensible Operators**
   - Airflow provides operators like `PythonOperator`, `BashOperator`, and SQL operators to cover most data tasks.
   - You can also write custom operators.
4. **Scalability**
   - Supports distributed execution with Celery, Kubernetes, or Local executors.
5. **User Interface**
   - A clean web UI to track DAGs, view logs, retry tasks, and monitor system health.

6. **Integration Support**

   ○ Works with databases, cloud platforms, data lakes, APIs, and more.

Steps to Build a Pipeline

Step 1: Environment Setup

We'll use Docker Compose to launch Airflow locally.

Get the official docker-compose file

https://airflow.apache.org/docs/apache-airflow/stable/docker-compose.yaml

Create required directories

mkdir -p ./dags ./logs ./plugins

Set user environment variable

echo -e "AIRFLOW_UID=$(id -u)" > .env

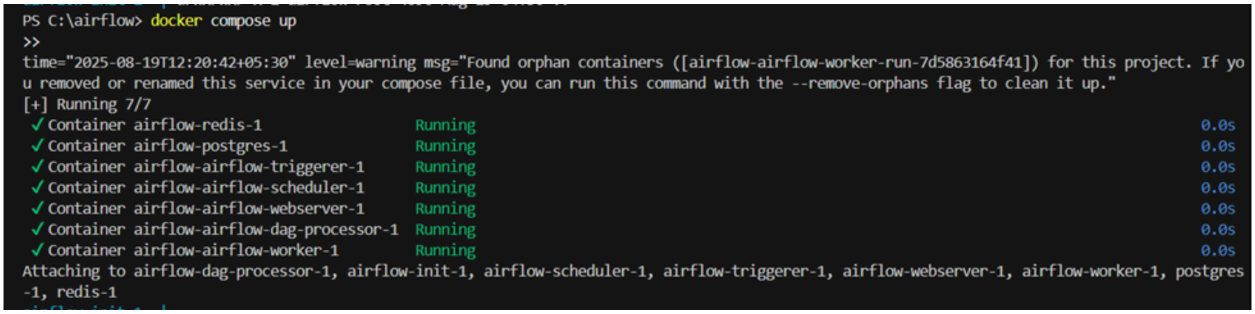
Initialize database

docker compose up airflow-init

```
PS C:\Airflow> docker compose up airflow-init
[+] Running 2/2
 ✓Container airflow-postgres-1  Running                          0.0s
 ✓Container airflow-redis-1     R...                             0.0s
Attaching to airflow-init-1
airflow-init-1  |
airflow-init-1  | WARNING!!!: Not enough memory available for Docker.
airflow-init-1  | At least 4GB of memory required. You have 3.8G
airflow-init-1  |
airflow-init-1  |
airflow-init-1  | WARNING!!!: You have not enough resources to run Airflow (see above)!
airflow-init-1  | Please follow the instructions to increase amount of resources available:
airflow-init-1  |    https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html#before-you-begin
airflow-init-1  |
airflow-init-1  |
airflow-init-1  | Creating missing opt dirs if missing:
airflow-init-1  |
airflow-init-1  |
airflow-init-1  | Airflow version:
airflow-init-1  | The container is run as root user. For security, consider using a regular user account.
airflow-init-1  |
```

Start all services

docker compose up



```
PS C:\airflow> docker compose up
>>
time="2025-08-19T12:20:42+05:30" level=warning msg="Found orphan containers ([airflow-airflow-worker-run-7d5863164f41]) for this project. If yo
u removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up."
[+] Running 7/7
 ✓ Container airflow-redis-1                Running                                                                                    0.0s
 ✓ Container airflow-postgres-1             Running                                                                                    0.0s
 ✓ Container airflow-airflow-triggerer-1    Running                                                                                    0.0s
 ✓ Container airflow-airflow-scheduler-1    Running                                                                                    0.0s
 ✓ Container airflow-airflow-webserver-1    Running                                                                                    0.0s
 ✓ Container airflow-airflow-dag-processor-1 Running                                                                                   0.0s
 ✓ Container airflow-airflow-worker-1       Running                                                                                    0.0s
Attaching to airflow-dag-processor-1, airflow-init-1, airflow-scheduler-1, airflow-triggerer-1, airflow-webserver-1, airflow-worker-1, postgres
-1, redis-1
```

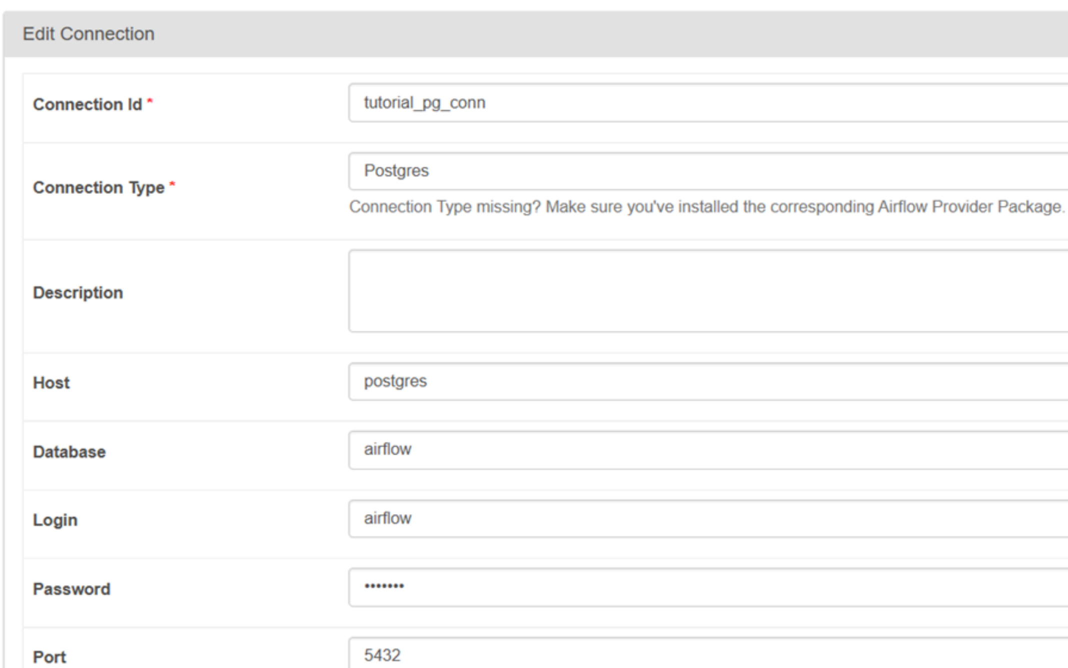
Open the UI at http://localhost:8080 and log in:

- Username: **airflow**
- Password: **airflow**

Step 2: Create a Postgres Connection

In the UI, navigate to Admin → Connections and add:

- **Conn ID:** tutorial_pg_conn
- **Type:** Postgres
- **Host:** postgres
- **Database:** airflow
- **Login:** airflow
- **Password:** airflow
- **Port:** 5432



Step 3: Define Tables

We'll create two tables:

- **employees_raw → temporary staging table**
- **employees_clean → final processed table**

These will be created using the SQLExecuteQueryOperator.

Step 4: Download and Load Data

Use Python tasks to download a CSV file and insert it into the staging table. Airflow hooks handle the database interaction.

Step 5: Merge and Clean Data

A second task will merge deduplicated data into the final table using INSERT ... ON CONFLICT.

Example DAG

```
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.postgres.operators.postgres import PostgresOperator



@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

    create_employees_table = PostgresOperator(

        task_id="create_employees_table",

        postgres_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 = PostgresOperator(

        task_id="create_employees_temp_table",
```

```
        postgres_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():
        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/tutorial/pipeline_example.csv"
        response = requests.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
```

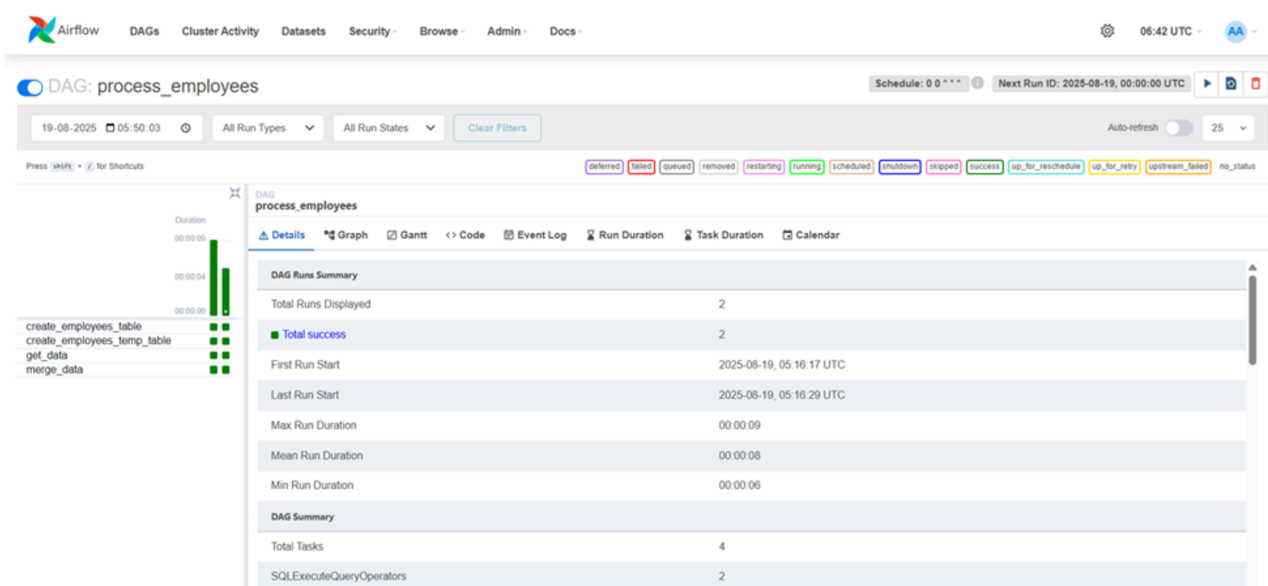
```
            "Employee Markme" = excluded."Employee Markme",
            "Description" = excluded."Description",
            "Leave" = excluded."Leave";
        """
        try:
            postgres_hook =
PostgresHook(postgres_conn_id="tutorial_pg_conn")
            conn = postgres_hook.get_conn()
            cur = conn.cursor()
            cur.execute(query)
            conn.commit()
            return 0
        except Exception as e:
            return 1

    # Task dependencies
    [create_employees_table, create_employees_temp_table] >> get_data() >>
merge_data()


dag = ProcessEmployees()
```

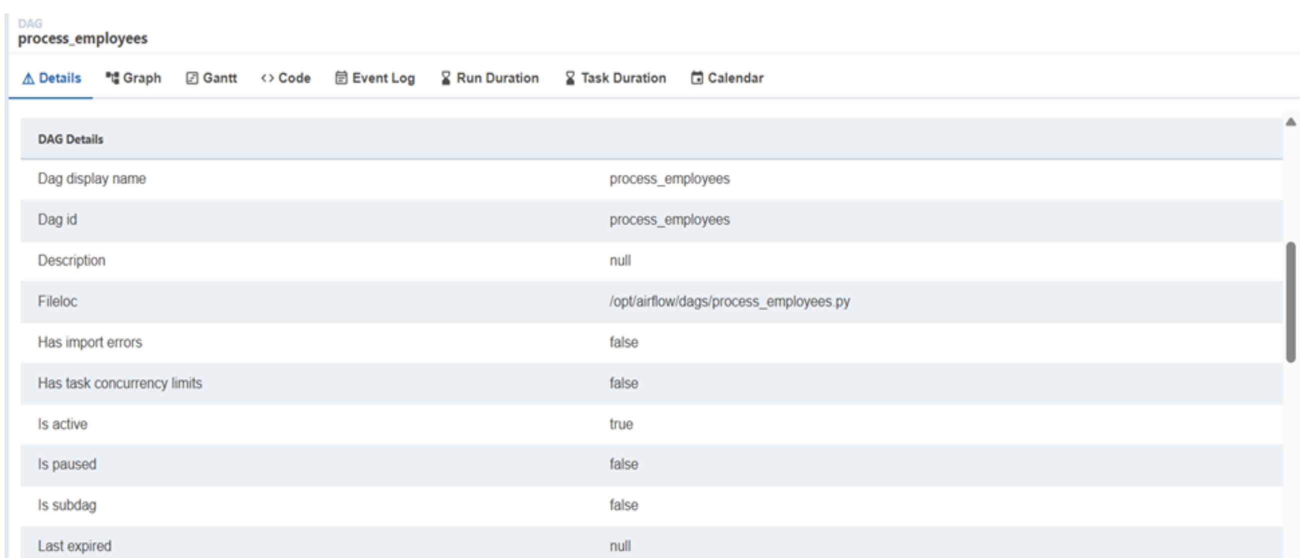
Running the DAG

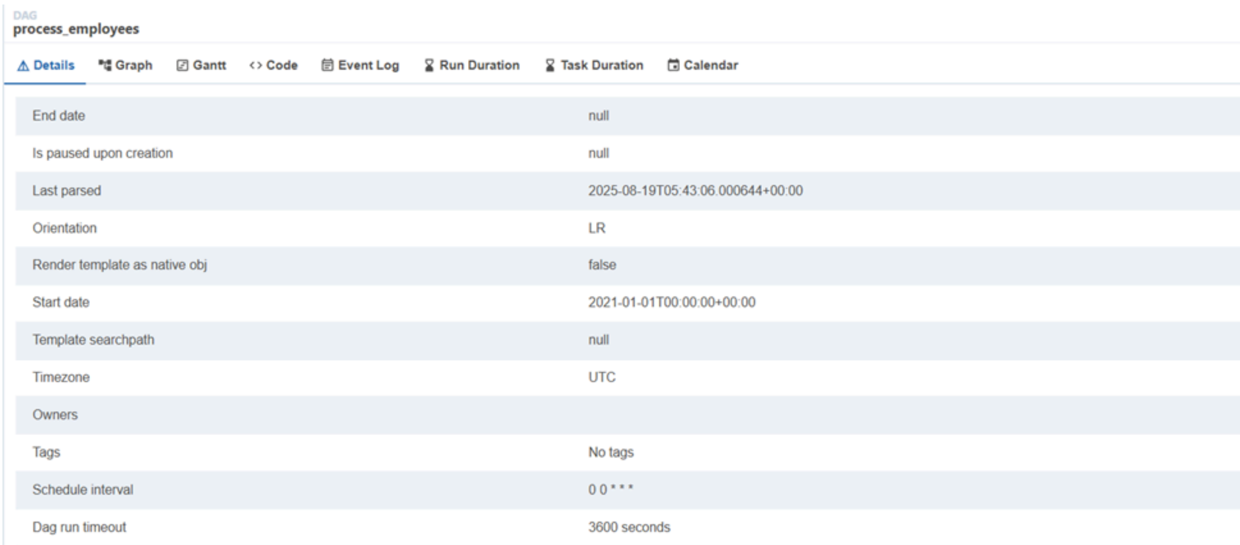
Once the DAG was saved, it appeared in the Airflow UI. The DAG was triggered manually, and the pipeline successfully:

- 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.







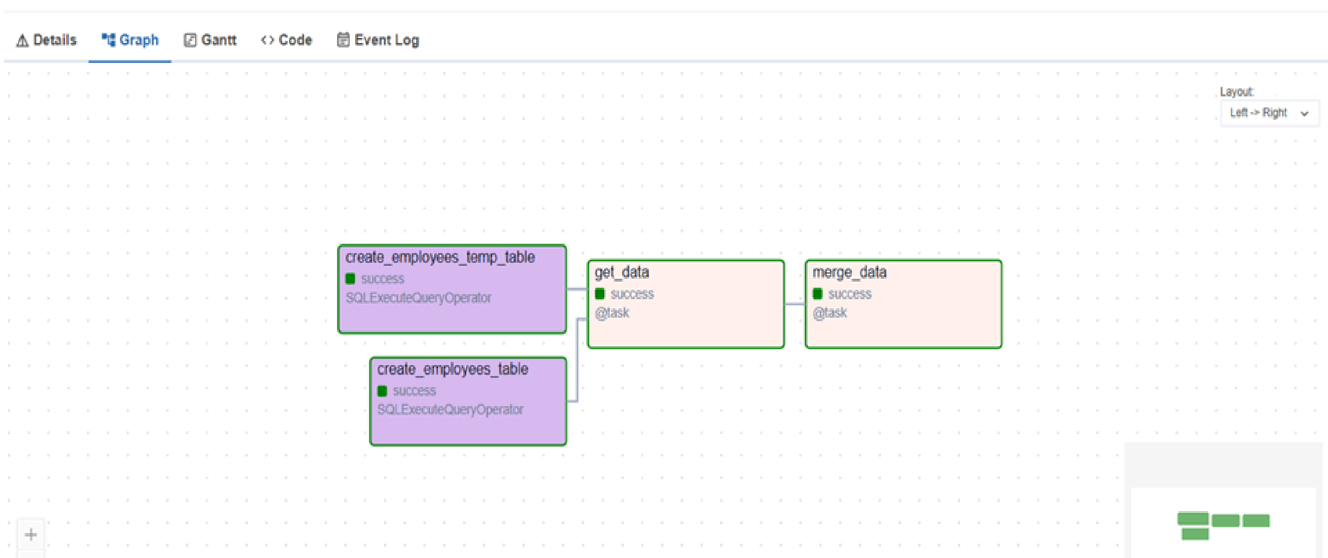
DAG
process_employees

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

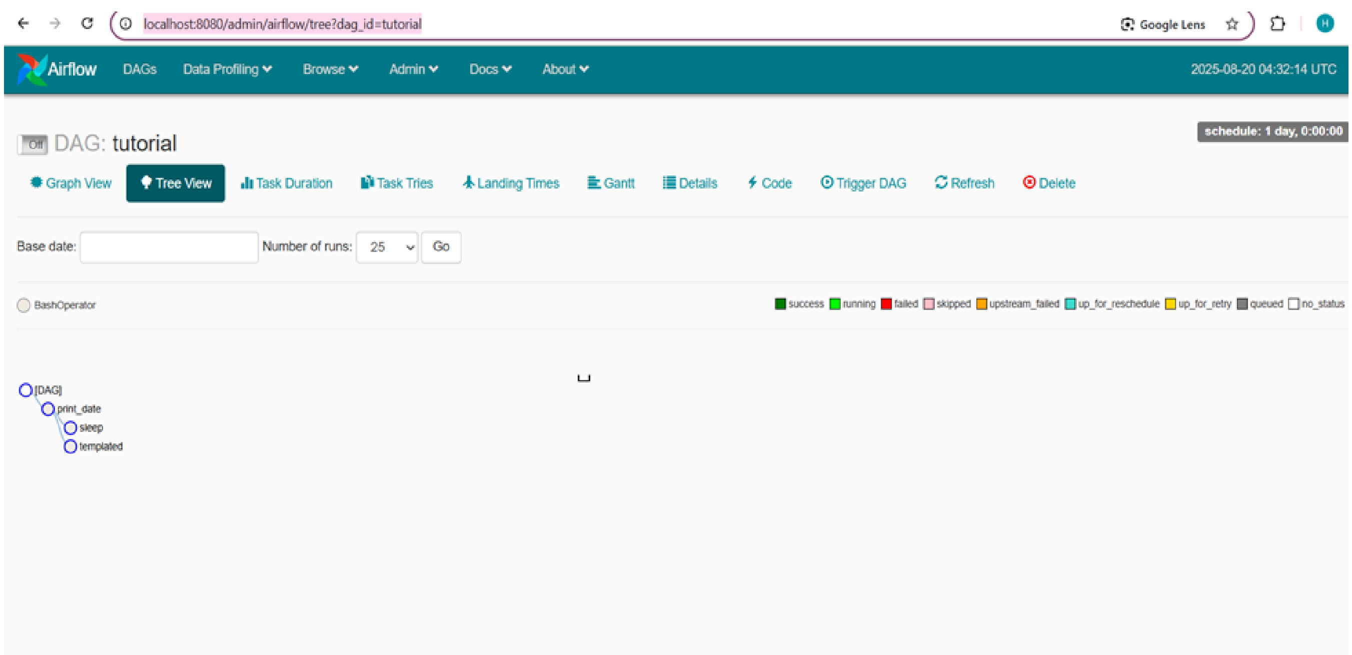
End date	null
Is paused upon creation	null
Last parsed	2025-08-19T05:43:06.000644+00:00
Orientation	LR
Render template as native obj	false
Start date	2021-01-01T00:00:00+00:00
Template searchpath	null
Timezone	UTC
Owners	
Tags	No tags
Schedule interval	0 0 * * *
Dag run timeout	3600 seconds

Exploring Airflow Views

- **Graph View** → Displays task dependencies in a node graph format.
- **Gantt View** → Timeline view of task duration and overlaps.
- **Tree View** → Quick overview of success/failure across historical runs.




DAG
process_employees
Run
2025-08-19, 00:00:00 UTC
Clear
Mark state as...
Details
Graph
Gantt
Code
Event Log


localhost:8080/admin/airflow/tree?dag_id=tutorial
Google Lens
Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
2025-08-20 04:32:14 UTC
schedule: 1 day, 0:00:00
Off
DAG: tutorial
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Trigger DAG
Refresh
Delete
Base date:
Number of runs:
25
Go
BashOperator
success
running
failed
skipped
upstream_failed
up_for_reschedule
up_for_retry
queued
no_status
[DAG]
print_date
sleep
templated