Tejashree G
tejashreeg.ai2021@dce.edu.in

Apache Airflow Coding Assessment

Introduction

Apache Airflow is an open-source platform used to programmatically author, schedule, and monitor workflows. It is widely adopted in data engineering for orchestrating ETL processes, data pipelines, and machine learning workflows.

Features of Apache Airflow

1. **Directed Acyclic Graphs (DAGs):**
    a. Workflows in Airflow are represented as DAGs.
    b. Each DAG defines the order of task execution without cycles.

2. **Task Scheduling:**
    a. Provides powerful scheduling options using CRON expressions or preset intervals.
    b. Supports backfilling (running past missed jobs).

3. **Scalability:**
    a. Can scale horizontally with multiple workers.

4. **Extensible Operators:**
    a. Comes with predefined operators (PythonOperator, BashOperator, SQL operators, etc.).

5. **Monitoring UI:**
    a. Web-based UI to track DAG runs, task states, and logs.
    b. Provides retry options and error debugging.

6. **Integration Support:**
    a. Connects with databases, cloud storage, data warehouses, and APIs.
    b. Supports plugins for extended functionality.

Building Pipeline Steps

Step 1: Initial Setup
Open Vscode and open the terminal
Run this command to download the docker-compose.yaml file
curl -LfO 'https://airflow.apache.org/docs/apache-airflow/stable/docker-compose.yaml'

Create the expected directories to work with airflow and set an expected environment variable

mkdir -p ./dags ./logs ./plugins
echo -e "AIRFLOW_UID=$(id -u)" > .env

Initialize the database
docker compose up airflow-init

Start up all services
docker compose up

```
PS C:\airflow> docker compose up airflow-init
>>
time="2025-08-19T12:19:55+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 2/2
 ✓ Container airflow-redis-1     Running                                                                                                0.0s
 ✓ Container airflow-postgres-1  Running                                                                                                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 3.5G
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  |
airflow-init-1  | /home/airflow/.local/lib/python3.8/site-packages/airflow/cli/cli_config.py:957 DeprecationWarning: The namespace option in [k
ubernetes] has been moved to the namespace option in [kubernetes_executor] - the old setting has been used, but please update your config.
```

```
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
airflow-init-1  |
```

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

- Username: airflow
- Password: airflow

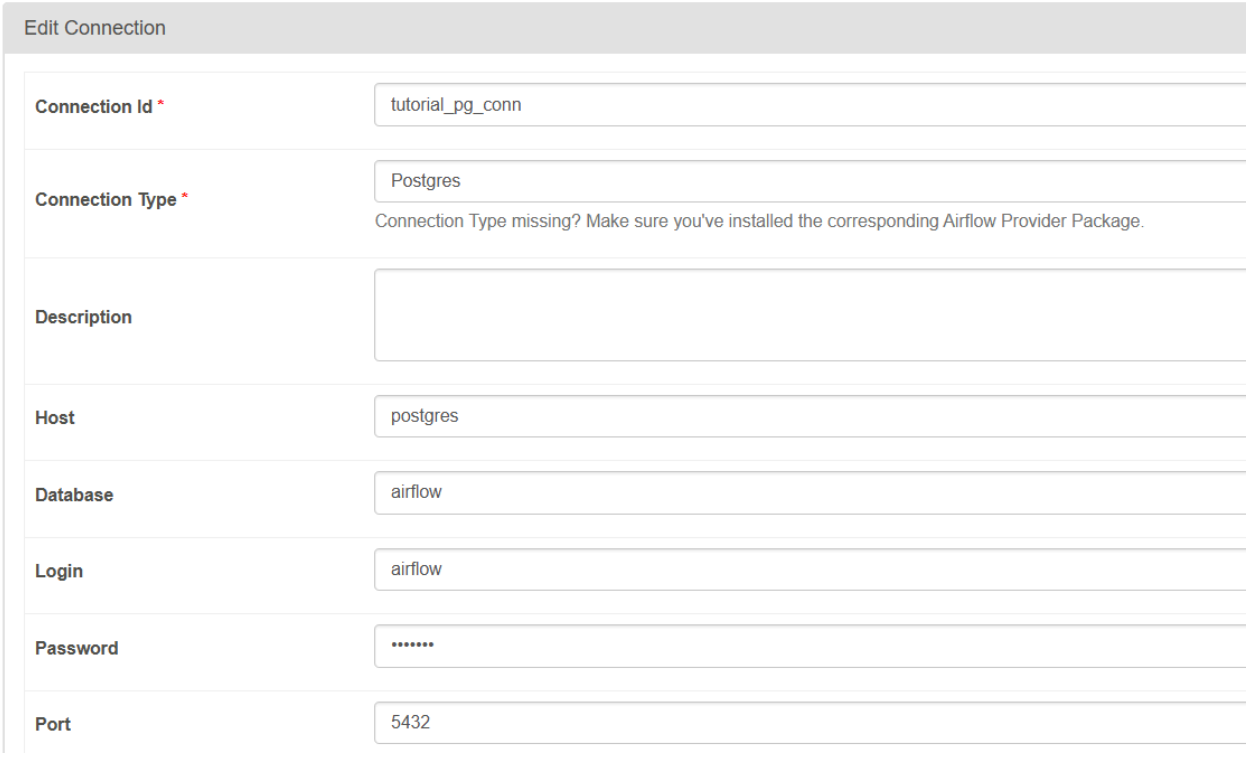
Step 2: Create Postgres Connection

Postgres 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.



Step3: Defining the DAG

Inside the dag folder create a python file called process_employees
This file will follow this 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:
```

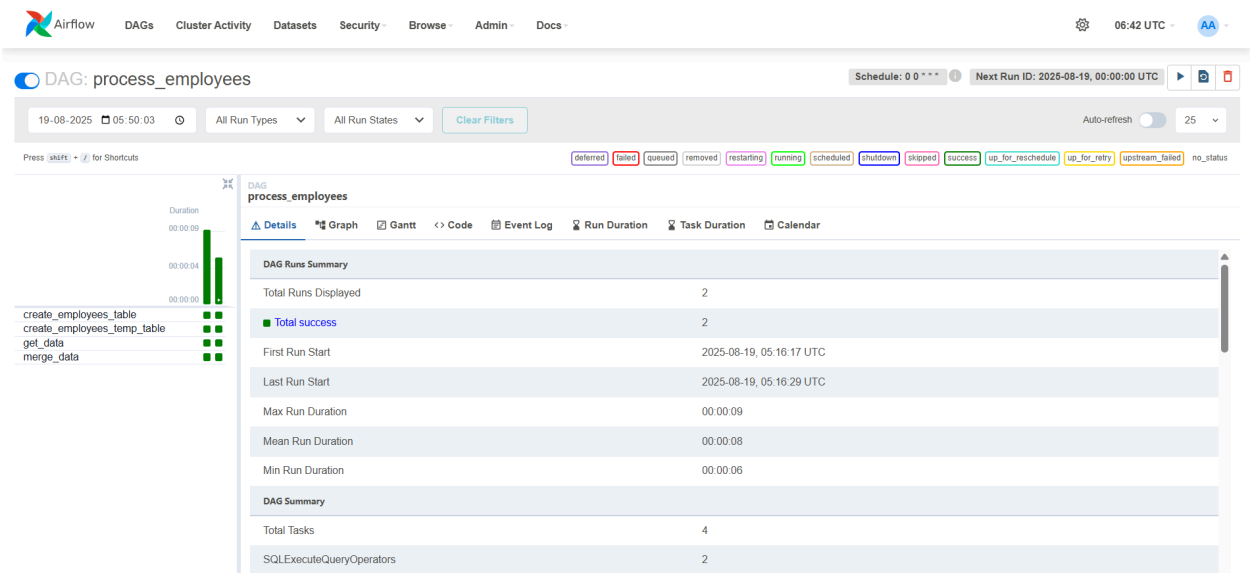
```
            postgres_hook =
PostgresHook(postgres_conn_id="tutorial_pg_conn")
            conn = postgres_hook.get_conn()
            cur = conn.cursor()
            cur.execute(query)
            conn.commit()
            return 0
        except Exception as e:
            return 1

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


dag = ProcessEmployees()
```

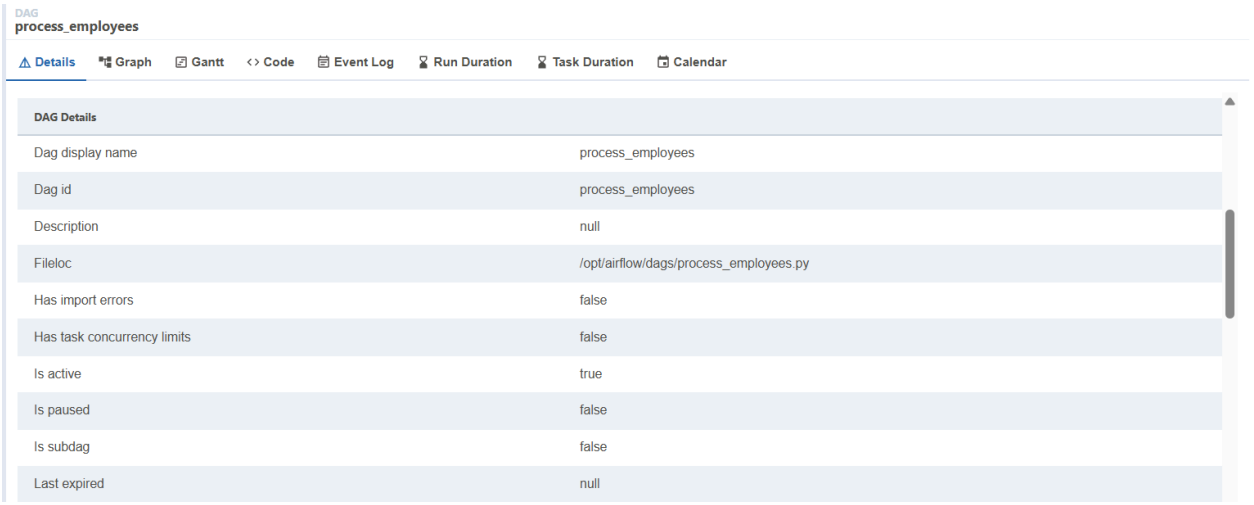
Step 4: Running the DAG

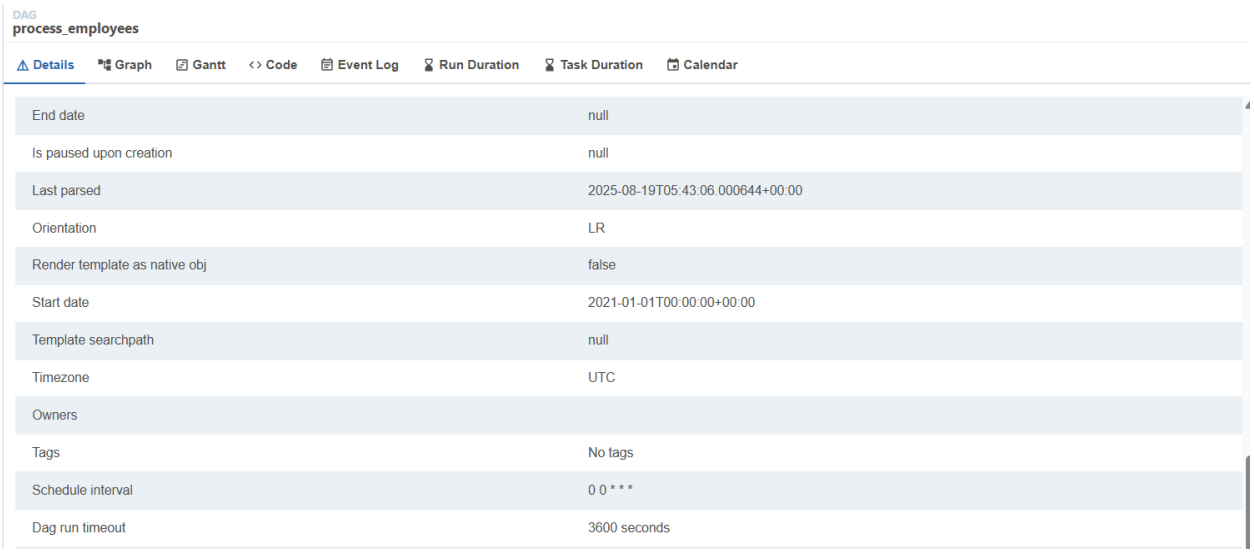
Once the DAG was saved, it appeared in the Airflow UI. The DAG was triggered manually, and the pipeline successfully:

- 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.



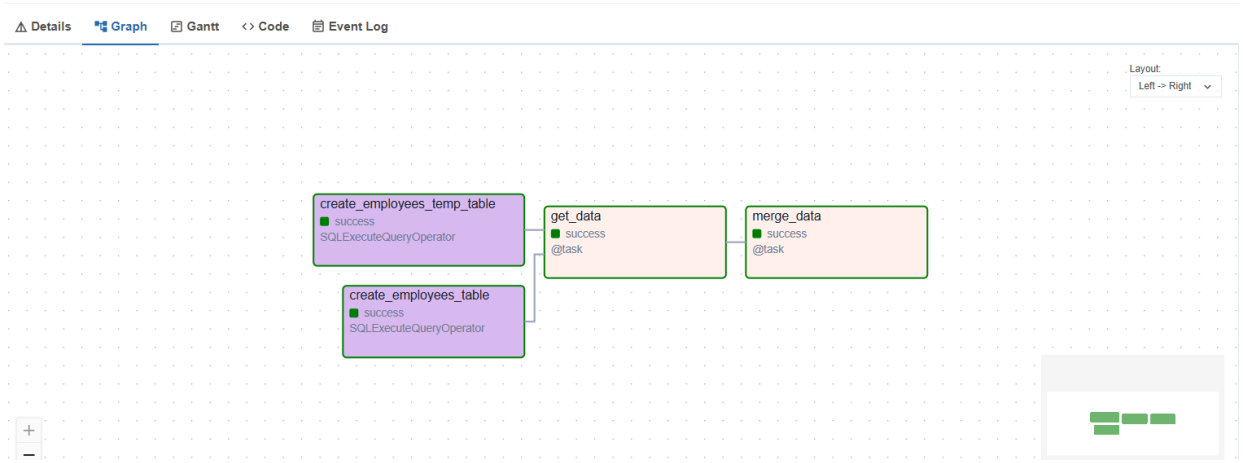




Airflow DAG Views

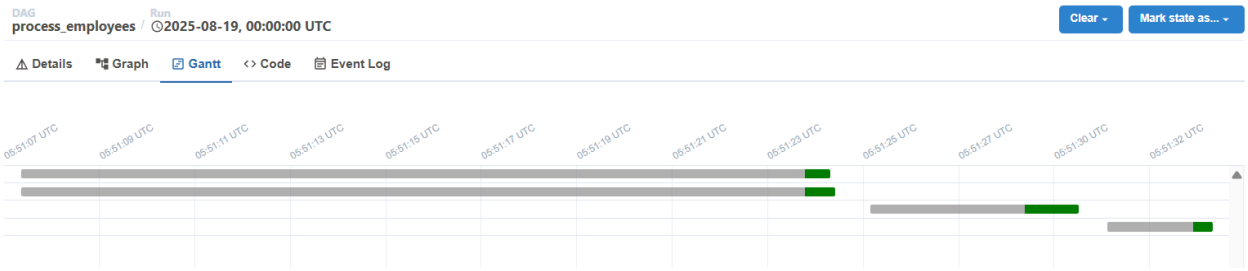
Graph View

- Shows tasks and their dependencies in a DAG graph format.
- Useful for visualizing execution flow.



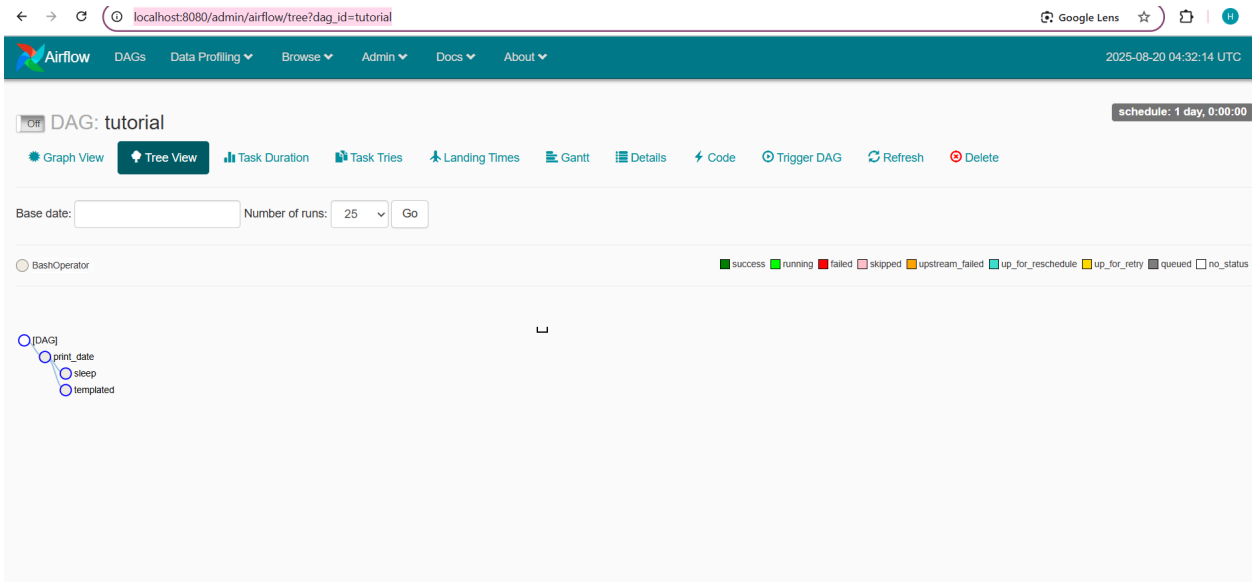
Gantt View

- Shows execution time of each task for a specific run.
- X-axis: timeline, Y-axis: tasks.



Tree View

- Displays tasks in a tree-like timeline.
- Each square shows task status (success, failed, skipped, etc).
- Helpful for quickly scanning multiple DAG runs.


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