CODING CHALLENGE (Apache Airflow)

TRAINEE NAME: Swathi Baskaran

Apache Airflow is an open-source workflow orchestration tool. It helps us to define, schedule and monitor workflows as code. Pipelines are defined as DAGs using Python code, tasks are created with operators and dependencies are set to define workflow execution. Using Airflow, we can automate repetitive tasks, orchestrate data movement, and manage dependencies between processes. This was first developed by Airbnb. It later became a part of Apache.

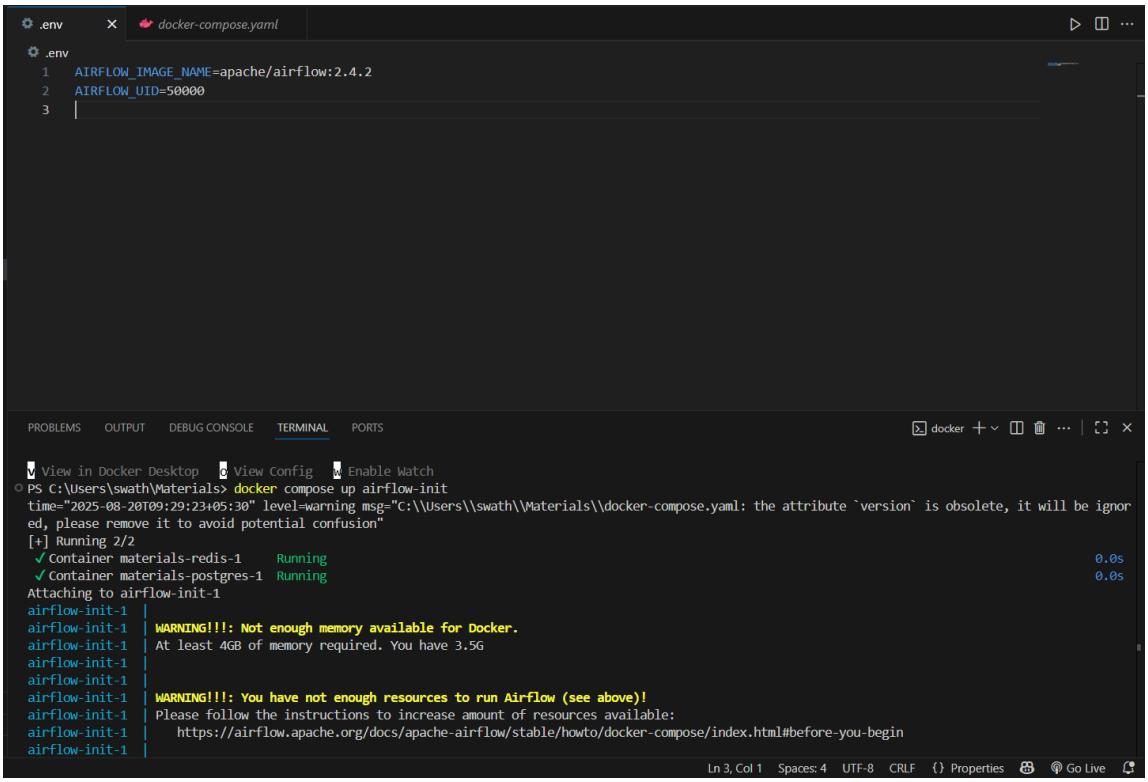
Features of Apache Airflow:

1. **Dynamic Workflow Creation**: Pipelines are defined as Python code, making them dynamic, extensible, and easy to modify.
2. **DAGs (Directed Acyclic Graphs)**: Workflows are represented as a graph of tasks with clear dependencies.
3. **Task Scheduling**: Workflows can be scheduled based on time, events or external triggers.
4. **Scalability**: Supports execution across distributed environments using Celery, Kubernetes or local Executors
5. **Monitoring and Logging**: Apache Airflow provides an extensive UI to monitor running tasks, view logs, and retry failed tasks.
6. **Extensibility**: Offers integrations with multiple data sources, cloud platforms, and third-party systems.
7. **Version Control**: Workflows can be managed under Git or any versioning system.
8. **Data Lineage**: Helps in tracking dependencies and flow of data between tasks.

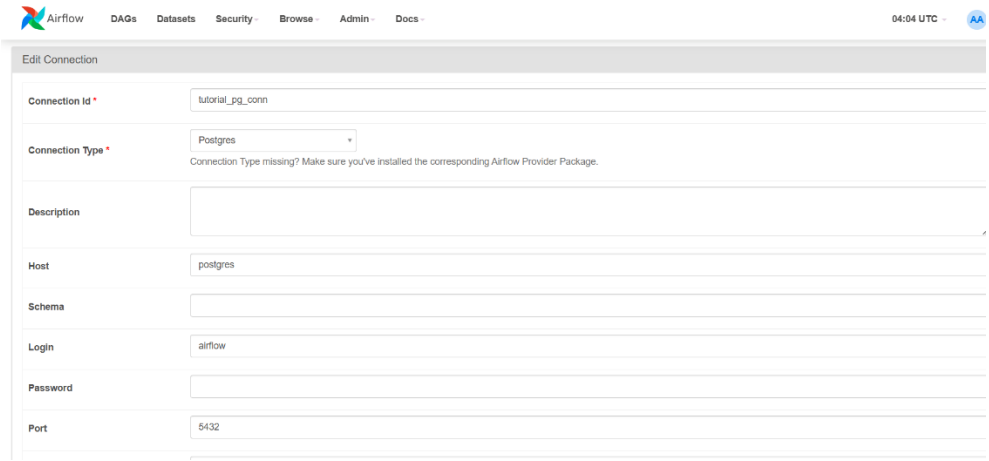
Building a pipeline in Apache Airflow:

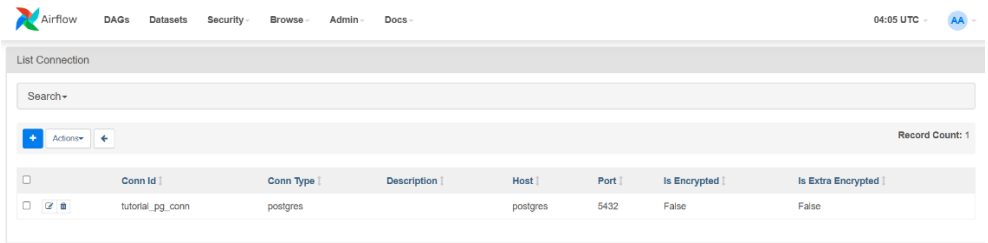
1. Install and set up Airflow
    - Install Apache Airflow using pip or Docker.
    - Initialize the Airflow metadata database (docker compose up airflow-init)

- Start Airflow components: webserver, scheduler, and workers. (docker compose up)

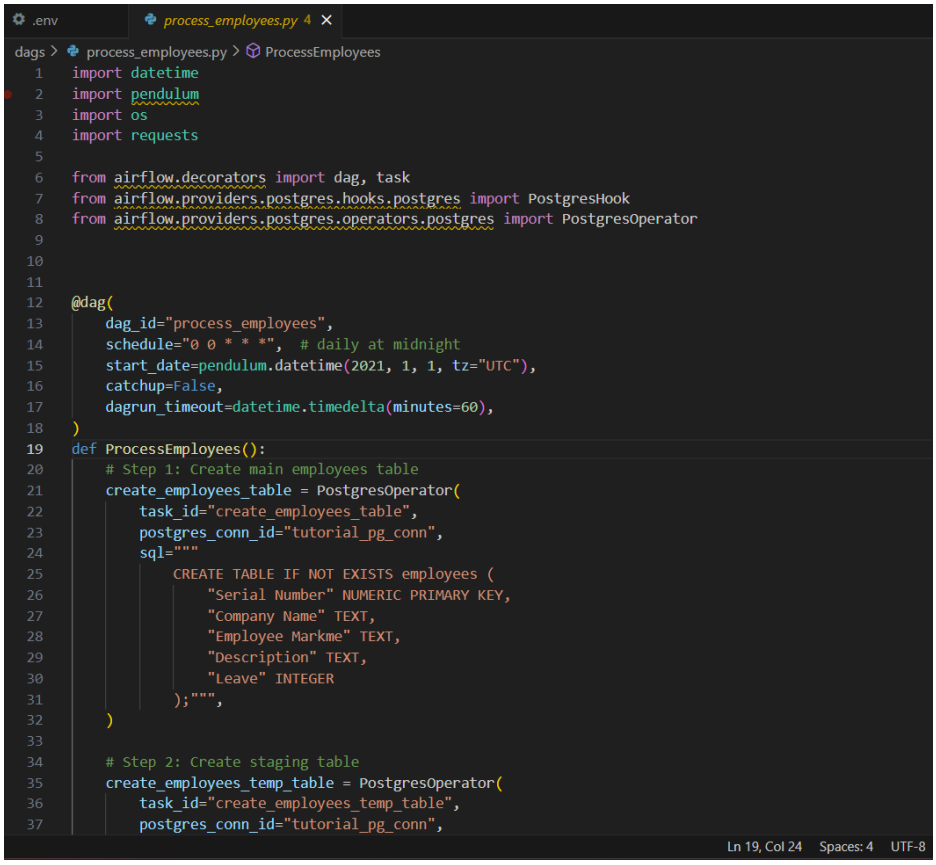


2. Creating a connection with Apache UI
    - In the Apache UI, a connection is created which mentions the port, username, password, database and other details that are required.





3. Define a DAG (Directed Acyclic Graph)
    - Create a Python script inside the dags folder
    - Define default arguments (owner, retries, retry_delay, start_date)
    - Define the DAG with DAG() function and a schedule interval.



4. Define Tasks
    - Use Operators such as PostGresOperator, BashOperator, PythonOperator to define tasks.
    - The operations that we choose to perform are defined as tasks

.env load_data_dag.py 2

dags > load_data_dag.py > get_data

```python
import os
import requests
from airflow.decorators import task
from airflow.providers.postgres.hooks.postgres import PostgresHook


@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/tutorial/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()
```

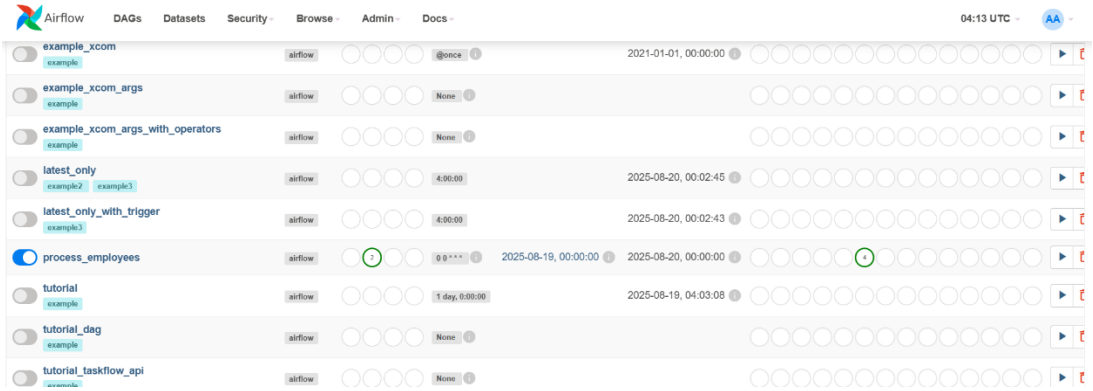
.env merge_task.py 2

dags > merge_task.py > merge_data

```python
from airflow.decorators import task
from airflow.providers.postgres.hooks.postgres import PostgresHook


@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
```

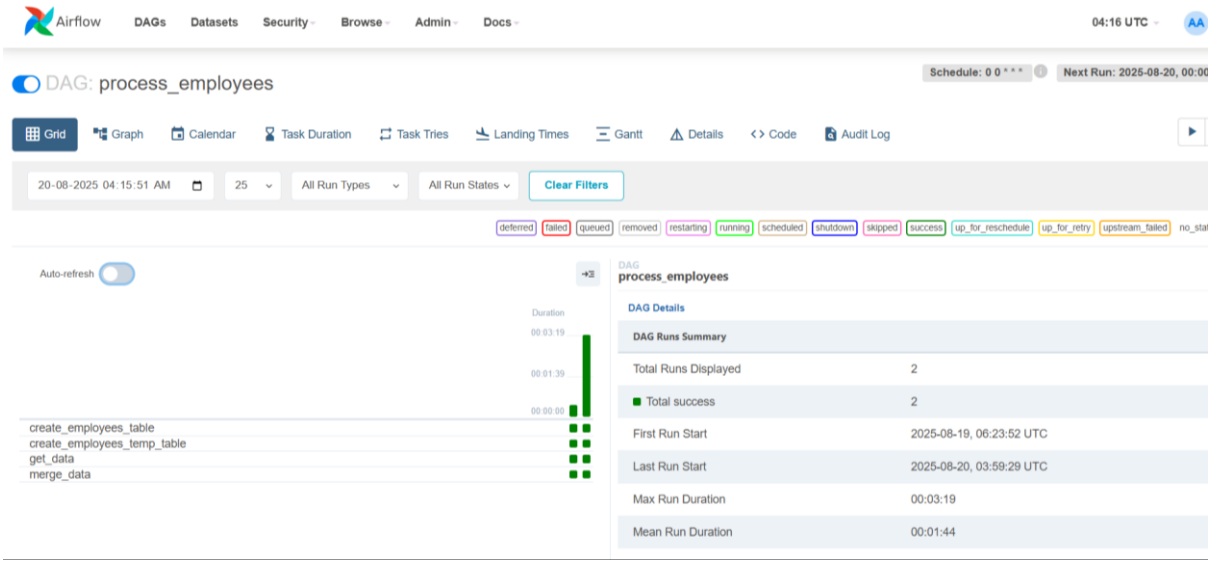
5. Set Task Dependencies
    - Use >> (Right shift) and >> (Left shift) to set execution order.
6. Deploying the DAG
    - The DAG file is placed in the dags directory
    - Scheduler automatically picks it up
    - The DAG is either triggered manually or we have to wait for its schedule



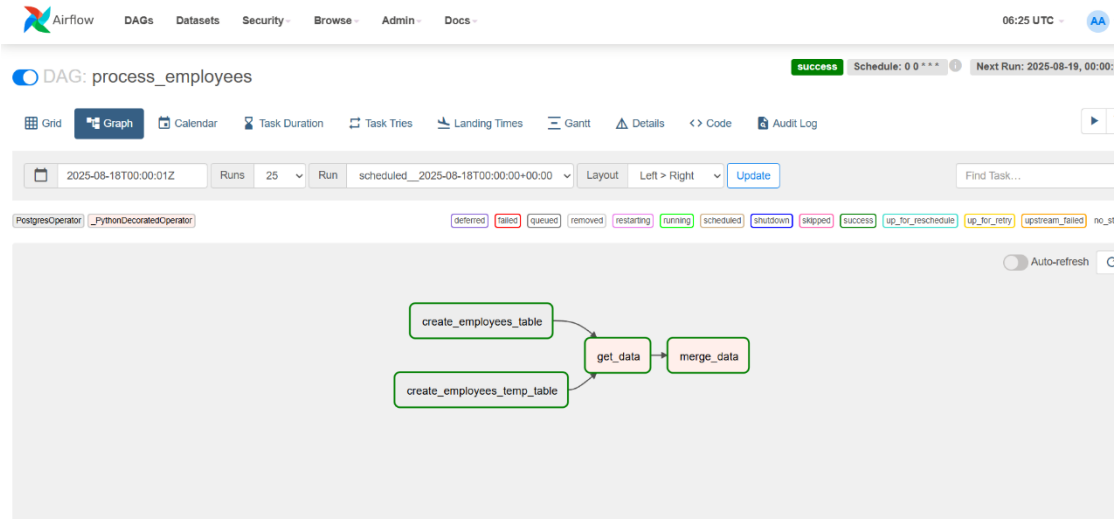
7. Monitor execution of DAG
    - Open Airflow UI (http://localhost:8080)
    - Check DAG runs, task statuses and logs.
    - Retry failed tasks if needed.

Viewing Pipelines in Airflow:

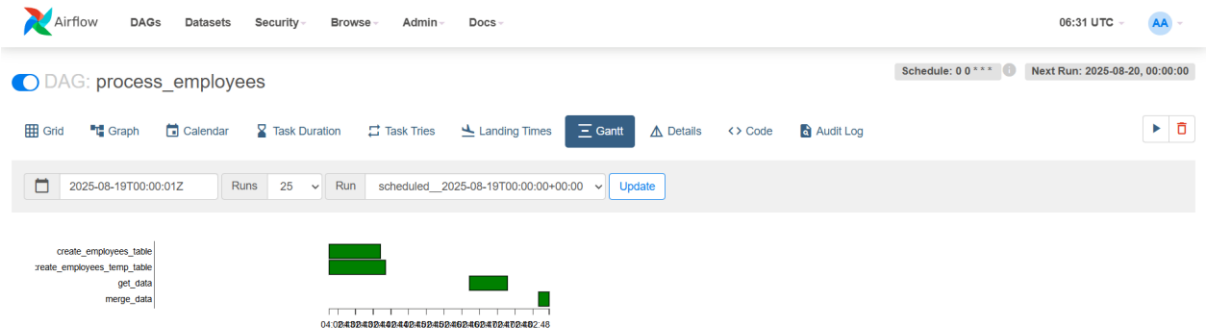
- **Grid View:** Shows DAG runs in a tabular format with task statuses, making it easy to spot failures and successes over multiple runs.



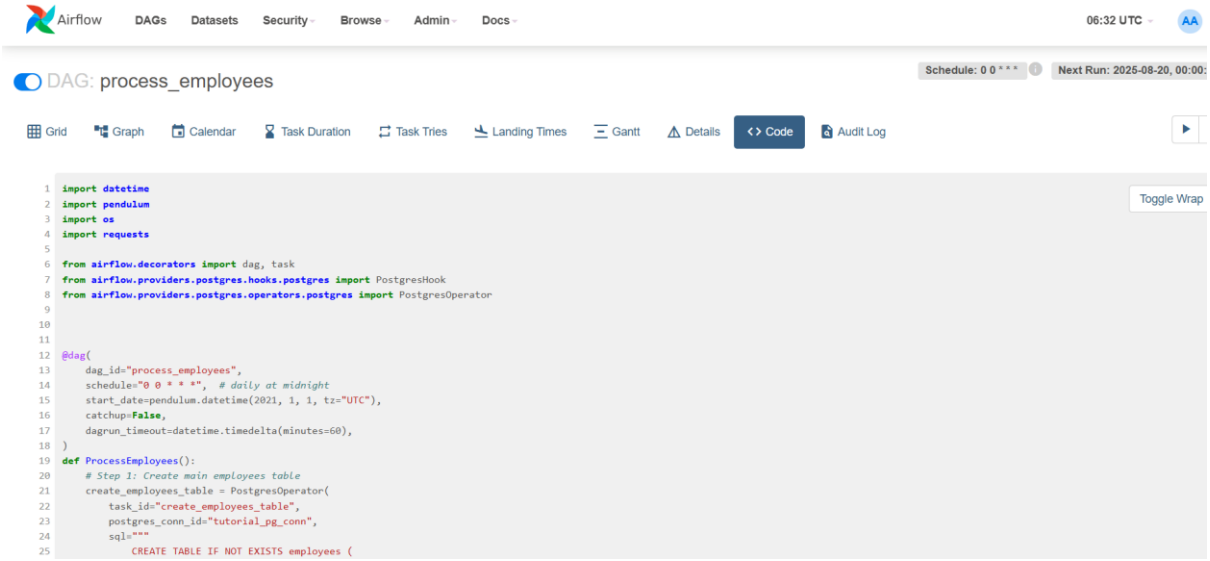
- **Graph View:** It is a visual representation of the workflow DAG showing task dependencies.



- **Tree View:** Displays task statuses across multiple DAG runs.
- **Gantt Chart:** Shows task execution timeline for performance monitoring.



- **Code View:** Displays DAG's Python code directly in the UI.



- **Task Instance Logs:** Each task has execution logs viewable in the UI.