Coding Challenge

Airflow

Apache Airflow – Features and Pipeline Building

1. Introduction

Apache Airflow is an open-source workflow orchestration tool developed by Airbnb and later contributed to the Apache Software Foundation.
It allows users to programmatically author, schedule, and monitor workflows (pipelines) in the form of Directed Acyclic Graphs (DAGs).

Airflow is widely adopted in data engineering, machine learning, ETL (Extract–Transform–Load) processes, and DevOps for automating complex tasks.

Why Airflow?

- Manual scheduling and managing scripts quickly becomes inefficient.
- Airflow provides automation, visibility, and scalability to data workflows.
- It integrates with cloud platforms, databases, and big data tools.

2. Key Features of Apache Airflow

- ❖ Dynamic Pipeline Generation: Pipelines are defined as Python code, making them dynamic and extensible.
- ❖ Scalability: Supports scaling across multiple workers using Celery, Kubernetes, or other executors.
- ❖ Rich UI: Provides a web-based UI for monitoring, managing, and debugging workflows.
- ❖ Scheduling: Built-in scheduler for running tasks at defined intervals.

❖ Extensibility: Support for custom operators, sensors, and hooks to integrate with third-party systems.

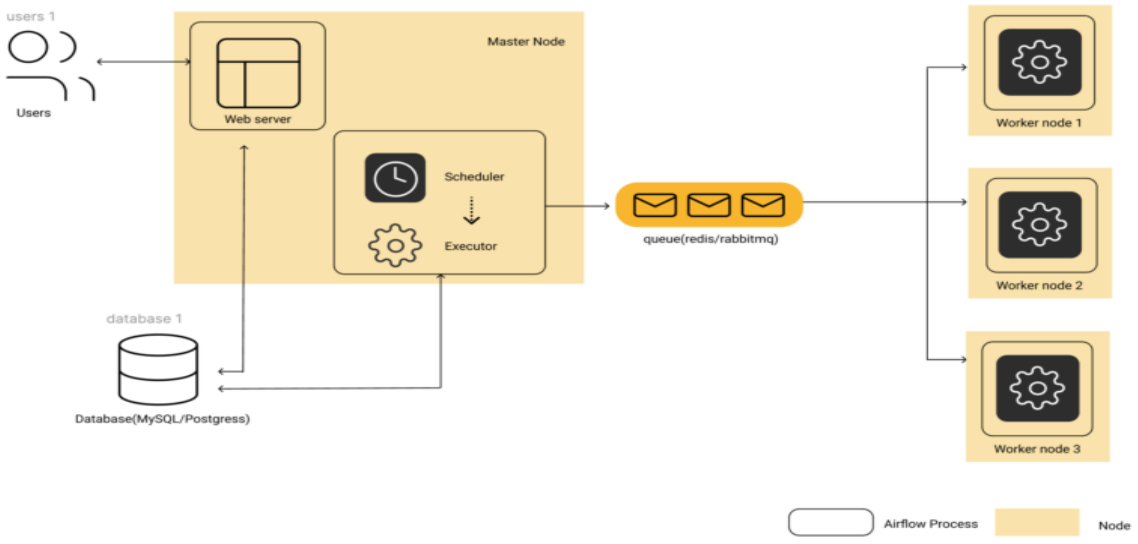
❖ Monitoring: Logs and alerts help track workflow execution and failures.

❖ Community: Strong open-source community with a wide range of pre-built operators.

3. Airflow Architecture Overview

Airflow consists of the following main components:

- **Scheduler**: Decides what tasks need to run and when.
- **Executor**: Runs the tasks (Local, Celery, Kubernetes, etc.).
- **Web Server (UI)**: Flask-based web application to monitor workflows.
- **Metadata Database**: Stores DAGs, task states, logs, variables, and connections (usually MySQL/Postgres).
- **Workers**: Execute the tasks assigned by the scheduler.



4. Steps to Build a Pipeline (DAG)

Step 1: Install Apache Airflow

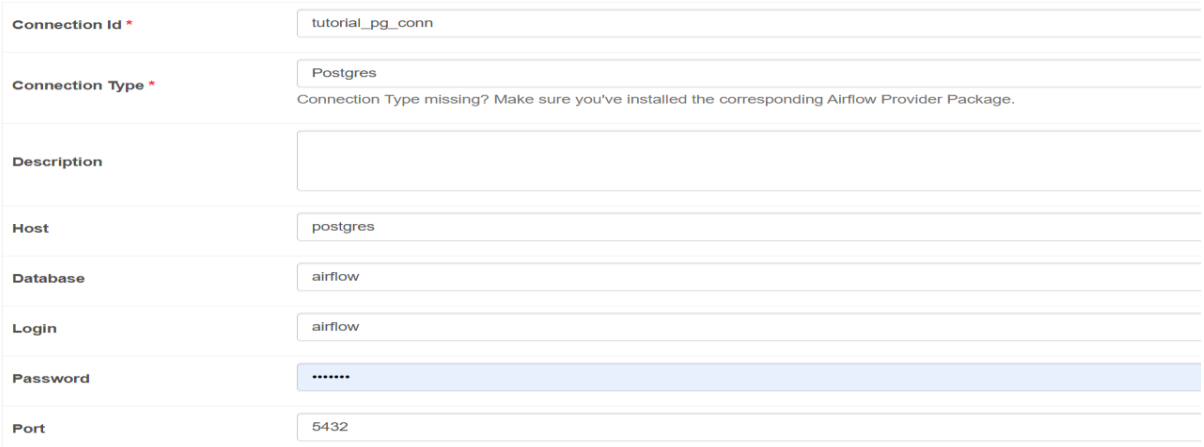
- Option 1: Using Docker Compose
- Option 2: Using pip install apache-airflow
- Option 3: Managed service (Astronomer, Google Cloud Composer, AWS MWAA).

Step 2: Configure Environment

- Initialize database: airflow db init
- Start scheduler: airflow scheduler
- Start webserver: airflow webserver -p 8080

Step 3: Create a DAG File

- Place .py file in the /dags directory.
- DAG definition includes:
    - dag_id
    - start_date
    - schedule_interval
    - catchup flag



Step 4: Define Tasks with Operators

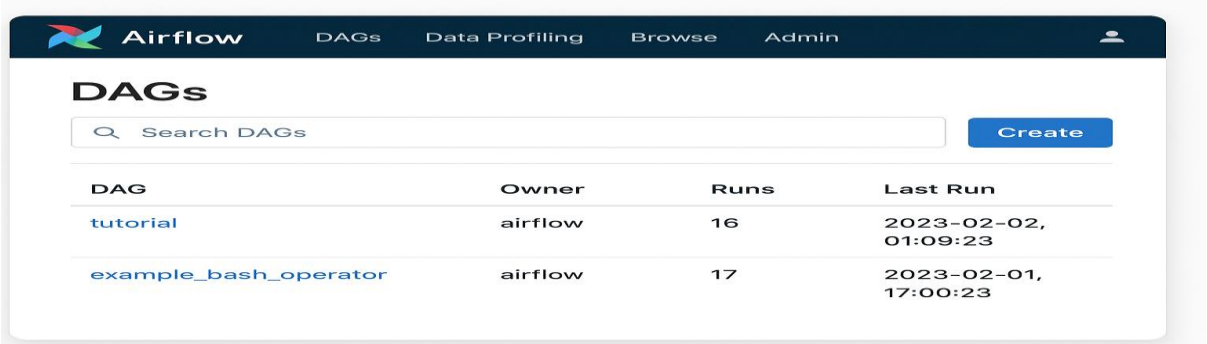
- Example operators:
  - PythonOperator (run Python functions)
  - BashOperator (execute bash commands)
  - EmailOperator, MySqlOperator, S3Operator, etc.

Step 5: Set Dependencies

- Use >> or << to set task order.
- Example: task1 >> task2 >> task3.

Step 6: Deploy and Enable DAG

- Refresh Airflow UI.
- Toggle DAG ON.



Step 7: Trigger and Run Pipeline

- Run manually or wait for scheduler.

Step 8: Monitor Execution

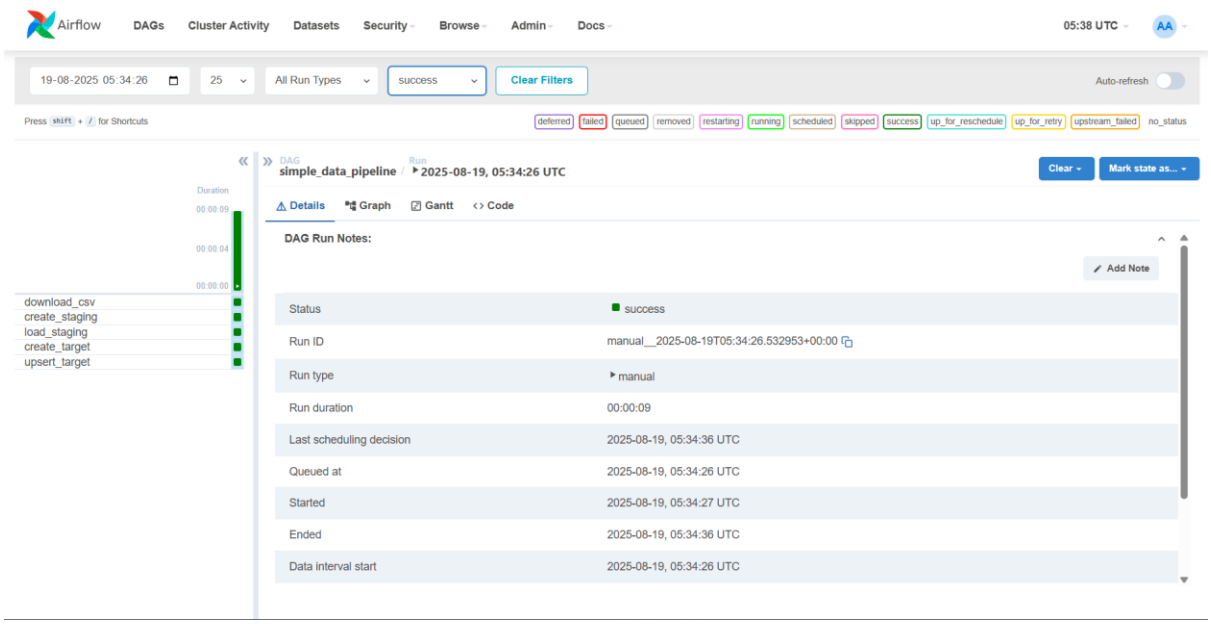
- Graph View, Tree View, Logs, Gantt View.

5. Example DAG Code

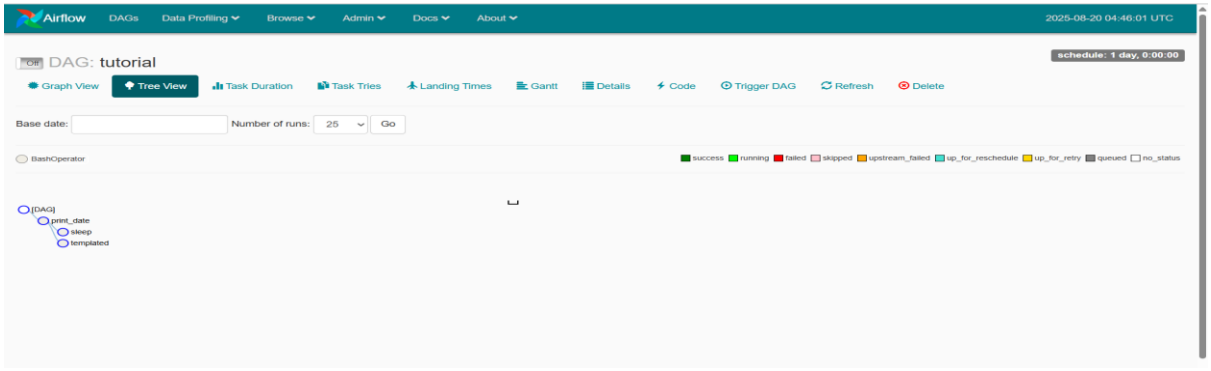
```
 from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime
# Define Python functions
def extract():
    print("Extracting data...")
def transform():
    print("Transforming data...")
def load():
    print("Loading data into destination...")
# Define DAG
with DAG(
    dag_id="etl_pipeline",
    description="A simple ETL pipeline example",
    start_date=datetime(2025, 1, 1),
    schedule_interval="@daily",
    catchup=False,
) as dag:
    t1 = PythonOperator(task_id="extract", python_callable=extract)
    t2 = PythonOperator(task_id="transform", python_callable=transform)
    t3 = PythonOperator(task_id="load", python_callable=load)
    # Setting dependencies
    t1 >> t2 >> t3
```

6. Viewing and Monitoring Pipelines

- DAGs Page: Lists workflows.



- Graph View: Shows task dependencies visually.
- Tree View: Tracks executions over time.



- Gantt View: Shows task execution duration.
- Task Logs: Debugging information.

7. Conclusion

Airflow provides a scalable, flexible, and code-first way to build and monitor pipelines.
By combining DAGs, operators, and scheduling, it helps automate workflows reliably.