

## Coding Challenge

### Apache Airflow – Features and Pipeline Building

Bavatharani S

#### 1. Introduction

Apache Airflow is an open-source workflow orchestration platform designed for managing, scheduling, and monitoring complex data pipelines. It was first created at Airbnb to solve the problem of scaling data workflows and was later donated to the Apache Software Foundation, where it has since grown into one of the most popular tools in modern data engineering.

Airflow allows engineers and analysts to design workflows as **Directed Acyclic Graphs (DAGs)**, where tasks are connected by dependencies that define their execution order. Each DAG describes how data flows from extraction, through transformation, to its final loading stage.



#### 2. Core Features

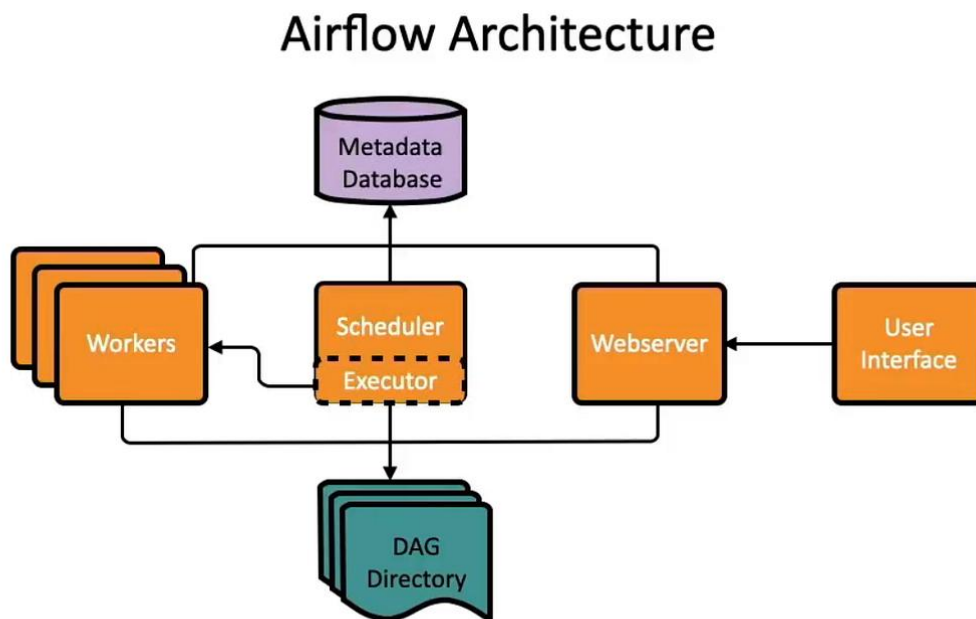
Apache Airflow is rich in features that make it suitable for enterprises handling data at scale:

- **Code-as-Configuration:** Pipelines are defined entirely in Python, enabling modular, testable, and version-controlled workflows. Developers can leverage Python's flexibility to dynamically generate tasks and DAGs.
- **Scalable Execution:** Airflow supports various executors such as Local, Celery, and Kubernetes, allowing workflows to run from small laptops to large distributed clusters. This scalability ensures it can handle both small jobs and enterprise-scale data workloads.
- **User-Friendly Web UI:** Airflow's web-based dashboard provides DAG-level monitoring, graphical task dependencies, and detailed logs. Developers can track execution history, success rates, and durations in real time.

- **Scheduling Engine:** Instead of relying on simple cron jobs, Airflow provides a powerful scheduling engine capable of handling daily, hourly, or complex time-based schedules. It also supports event-based triggering.
- **Extensible Plugins:** Through custom operators, hooks, and sensors, Airflow integrates with nearly any data source or cloud platform. Common integrations include Amazon S3, Google BigQuery, Apache Spark, and PostgreSQL.
- **Observability:** Airflow provides detailed logs, retry policies, and failure alerts. It can notify users through email, Slack, or other channels when something goes wrong.
- **Community and Ecosystem:** A large open-source community contributes plugins, bug fixes, and documentation. This ensures continuous improvement and adoption of modern data practices.

### 3. High-Level Architecture

The architecture of Airflow follows a distributed design with clear separation of concerns. Its main components are:



- **Scheduler** – The brain of Airflow, responsible for deciding which tasks need to run and when. It parses DAG definitions, determines task dependencies, and pushes tasks into the execution queue.
- **Executor** – The executor determines *how* and *where* tasks are executed. In small setups, the Local Executor runs tasks sequentially, while in larger environments, Celery or Kubernetes Executors distribute tasks across multiple worker nodes.

- **Web Server (UI)** – A Flask-based web application that provides visualization and monitoring. It allows users to pause or unpause DAGs, view dependency graphs, check logs, and manually trigger workflows.
- **Metadata Database** – Stores all essential information including DAG definitions, task instances, run history, variables, and connection settings. Typically, this database is backed by PostgreSQL or MySQL.
- **Workers** – Worker processes execute the actual tasks. Depending on the executor, tasks may run locally or across a distributed cluster.
- **Message Queue (optional for distributed setups)** – In Celery-based deployments, a message broker such as RabbitMQ or Redis is used to coordinate communication between the scheduler and workers.

This modular architecture ensures Airflow can be deployed flexibly, from single-node environments to large-scale clusters orchestrating hundreds of workflows.

## 4. Building a Pipeline (DAG)

Building a pipeline in Airflow involves creating a DAG file and defining tasks inside it. The steps are as follows:

### Step 1: Installation Options

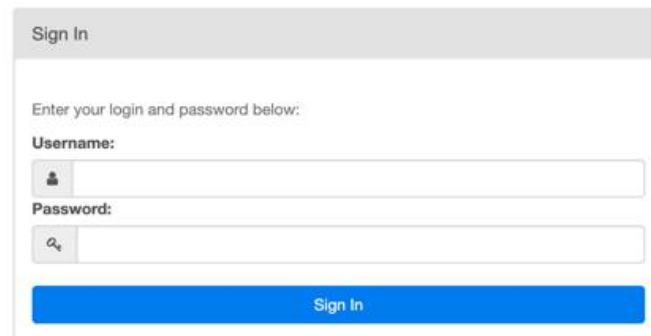
Airflow can be installed in several ways depending on the environment:

- Using pip install apache-airflow for local installations.
- Running via Docker Compose for containerized environments.
- Leveraging managed services like AWS Managed Workflows for Apache Airflow (MWAA), Google Cloud Composer, or Astronomer, which handle infrastructure overhead.

### Step 2: Initialize Environment

Once installed, initialize the metadata database with `airflow db init`. Then start the services:

- `airflow scheduler` – Runs in the background to schedule tasks.
- `airflow webserver -p 8080` – Launches the UI on port 8080.



A screenshot of the Airflow web interface's sign-in page. The page has a light gray header with the 'Sign In' title. Below the header, there is a white box containing the login form. The form includes a prompt 'Enter your login and password below:', followed by 'Username:' and 'Password:' labels. Each label is next to a text input field with a small icon (a person for username, a key for password). At the bottom of the form is a blue button labeled 'Sign In'.

### Step 3: Create a DAG File

Place a .py file inside the /dags directory. A DAG definition typically includes:

- dag\_id – Unique identifier for the workflow.
- start\_date – The beginning point for scheduling.
- schedule\_interval – Defines frequency (@daily, cron expression, etc.).
- catchup – Whether missed runs should be backfilled.

### Step 4: Define Tasks

Tasks represent individual units of work. Airflow provides a variety of operators, such as:

- PythonOperator for running Python functions.
- BashOperator for executing shell commands.
- Cloud-specific operators like S3Operator and BigQueryOperator.

### Step 5: Set Dependencies

Dependencies describe the order of execution. Tasks can be chained using >> or <<. For example:

```
task1 >> task2 >> task3
```

### Step 6: Deploy & Enable DAG

Once created, refresh the Airflow UI to load the DAG. Toggle the switch to enable it.

### Step 7: Trigger & Run

DAGs can run automatically based on their schedule, or manually through the UI or CLI.

### Step 8: Monitor Execution

Airflow provides multiple monitoring views: Graph, Tree, Gantt, and logs for detailed insights.

## 5. Example DAG

```
from airflow import DAG

from airflow.operators.python import PythonOperator

from datetime import datetime

def extract():
    print("Fetching raw data...")

def transform():
    print("Applying transformations...")

def load():
    print("Saving results into database...")

with DAG(
    dag_id="etl_workflow",
    description="ETL workflow example",
    start_date=datetime(2025, 1, 10),
    schedule_interval="@daily",
    catchup=False,
) as dag:

    extract_task = PythonOperator(task_id="extract", python_callable=extract)

    transform_task = PythonOperator(task_id="transform", python_callable=transform)

    load_task = PythonOperator(task_id="load", python_callable=load)

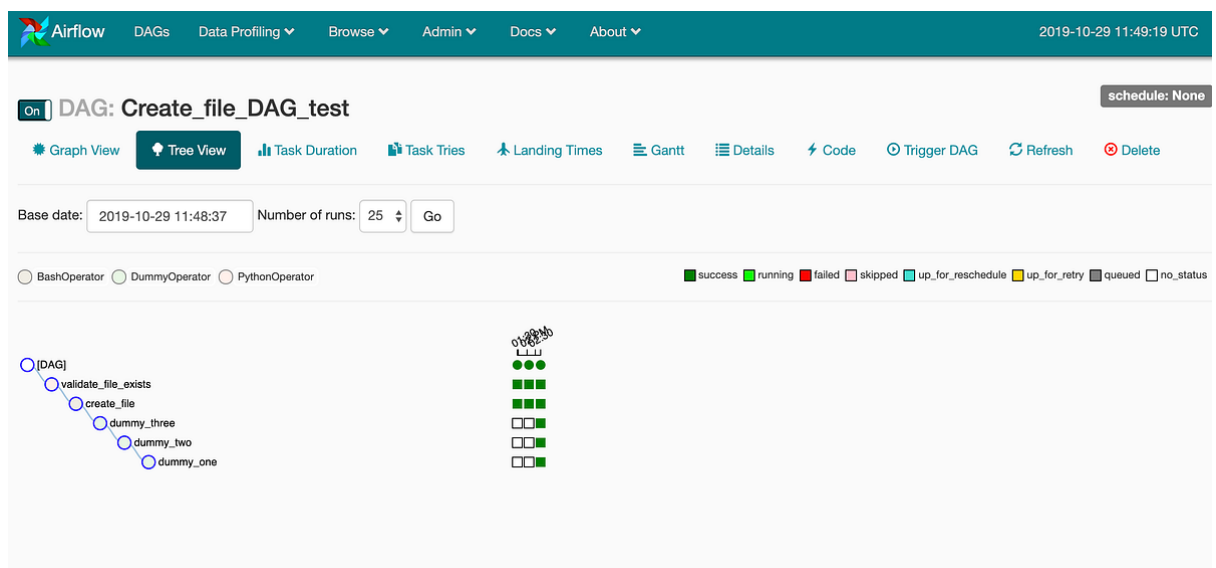
    extract_task >> transform_task >> load_task
```

This DAG demonstrates a classic ETL pipeline where data is extracted, transformed, and then loaded into a storage system. Each stage runs independently but in sequence, ensuring proper workflow execution.

## 6. Monitoring Workflows

Monitoring is one of Airflow's strongest features. The platform provides multiple tools for observing workflow execution:

- DAGs Page: Lists all available workflows along with their current status (running, paused, failed, success).
- Graph View: A visual representation of task dependencies, showing how data flows from start to finish.
- Tree View: Displays task runs over time, making it easy to spot recurring failures.
- Gantt Chart: Highlights task durations, overlaps, and bottlenecks in execution.
- Task Logs: Provides detailed debugging information with timestamps, error messages, and retries.



Additionally, Airflow supports integrations with monitoring tools like Prometheus, Grafana, and ELK stack, offering real-time metrics and alerting capabilities.

## 7. Conclusion

Apache Airflow has become a cornerstone in modern data platforms, offering flexibility, scalability, and transparency in workflow management. Its DAG-based design allows teams to break down complex processes into manageable steps, ensuring reliability and maintainability.

By combining a powerful scheduler, extensible operators, and a rich UI, Airflow helps organizations automate repetitive tasks, reduce manual errors, and improve productivity.

Whether used for ETL jobs, machine learning pipelines, or cloud data orchestration, Airflow provides a unified and code-driven approach to workflow automation, making it one of the most trusted tools for data engineers worldwide.