# APACHE AIRFLOW FEATURES AND BUILDING PIPELINE STEPS AND VIEW

# Theophila Murphy S

DE-Batch 6

# 1. INTRODUCTION

Apache Airflow is an open-source platform developed by Airbnb and now part of the Apache Software Foundation. It enables users to programmatically author, schedule, and monitor workflows. Airflow has become a standard tool in the data engineering and machine learning ecosystem due to its scalability, flexibility, and ability to integrate with various platforms. Workflows in Airflow are represented as Directed Acyclic Graphs (DAGs), where each node represents a task.

# 2. FEATURES OF APACHE AIRFLOW

Apache Airflow offers several powerful features that make it an essential tool for managing data workflows:

- **Workflow Orchestration:** Automates workflows as DAGs with clear dependencies.

- **Dynamic Pipelines:** Pipelines are written in Python, enabling dynamic generation of tasks.

- **Scalable Execution:** Multiple executors (Local, Celery, Kubernetes) allow scaling across clusters.

- **Monitoring & UI:** Provides a web UI for tracking DAGs, task statuses, and logs.

- **Extensible:** Custom operators, sensors, and hooks can be created to extend functionality.

- **Cloud & Tool Integration:** Works seamlessly with AWS, Azure, GCP, Hadoop, and modern data tools.

- **Retry & Alert Mechanism:** Automatic retries, alerts, and SLA monitoring ensure reliability.

# 3. AIRFLOW ARCHITECTURE

Airflow follows a modular architecture, where different components work together to orchestrate workflows:

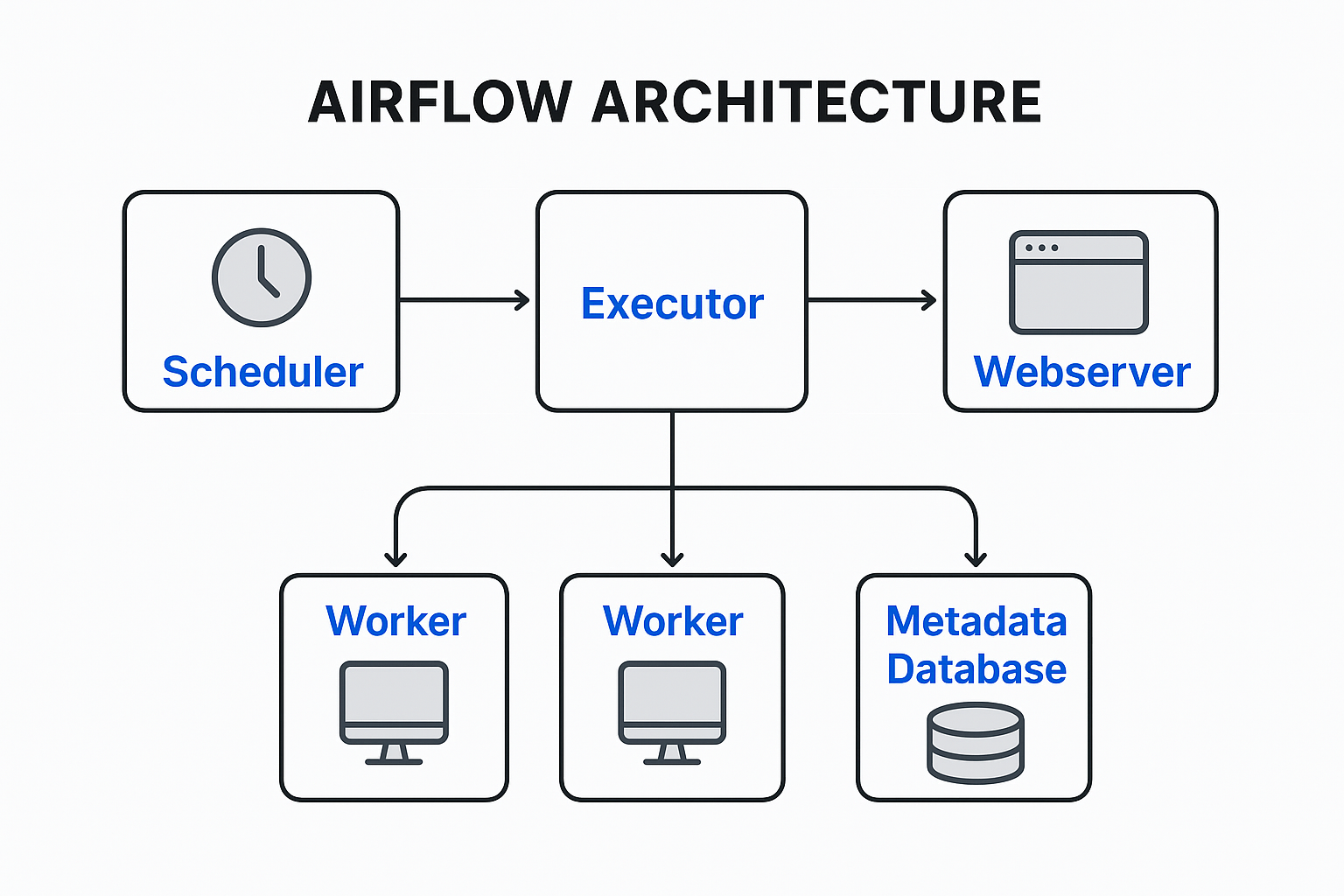
1. **Scheduler** – Decides task execution order and schedules jobs.

2. **Executor** – Defines how tasks run (Local, Celery, Kubernetes).

3. **Workers** – Machines/containers that execute the tasks.

4. **Webserver** – A user interface to visualize DAGs, tasks, and logs.

5. **Metadata Database** – Stores information about DAGs, task states, and execution history.



# 4. STEPS TO BUILD A PIPELINE IN AIRFLOW

Building a pipeline involves several systematic steps:

**Step 1: Set Up Environment**

- Install Apache Airflow using pip or Docker.  
- Initialize the database with `airflow db init`.  
- Start the webserver and scheduler services.  
- Access Airflow UI at http://localhost:8080.

**Step 2: Create a DAG**

- Define workflows in Python scripts inside the `dags/` directory.  
- DAGs contain metadata like `dag\_id`, `schedule\_interval`, and `start\_date`.  
- Example: A DAG scheduled daily to load transaction data.

**Step 3: Define Tasks**

- Use operators like `PythonOperator`, `BashOperator`, or `SqlOperator`.  
- Each task represents a unit of work (Extract, Transform, Load).  
- Example: A task that extracts data from a CSV file.

**Step 4: Set Dependencies**

- Define task order using `>>` and `<<` operators.  
- Example: `extract >> transform >> load` ensures correct workflow sequence.

**Step 5: Trigger & Monitor DAG**

- DAGs can be triggered manually or as per schedule.  
- Logs and retries can be checked in the Airflow UI.

**Step 6: Optimize & Scale**

- Configure parallelism and task retries.  
- Use CeleryExecutor or KubernetesExecutor for distributed scaling.

# 5. VIEWING PIPELINES IN AIRFLOW

Airflow provides rich visualizations in its web UI to help monitor pipelines:

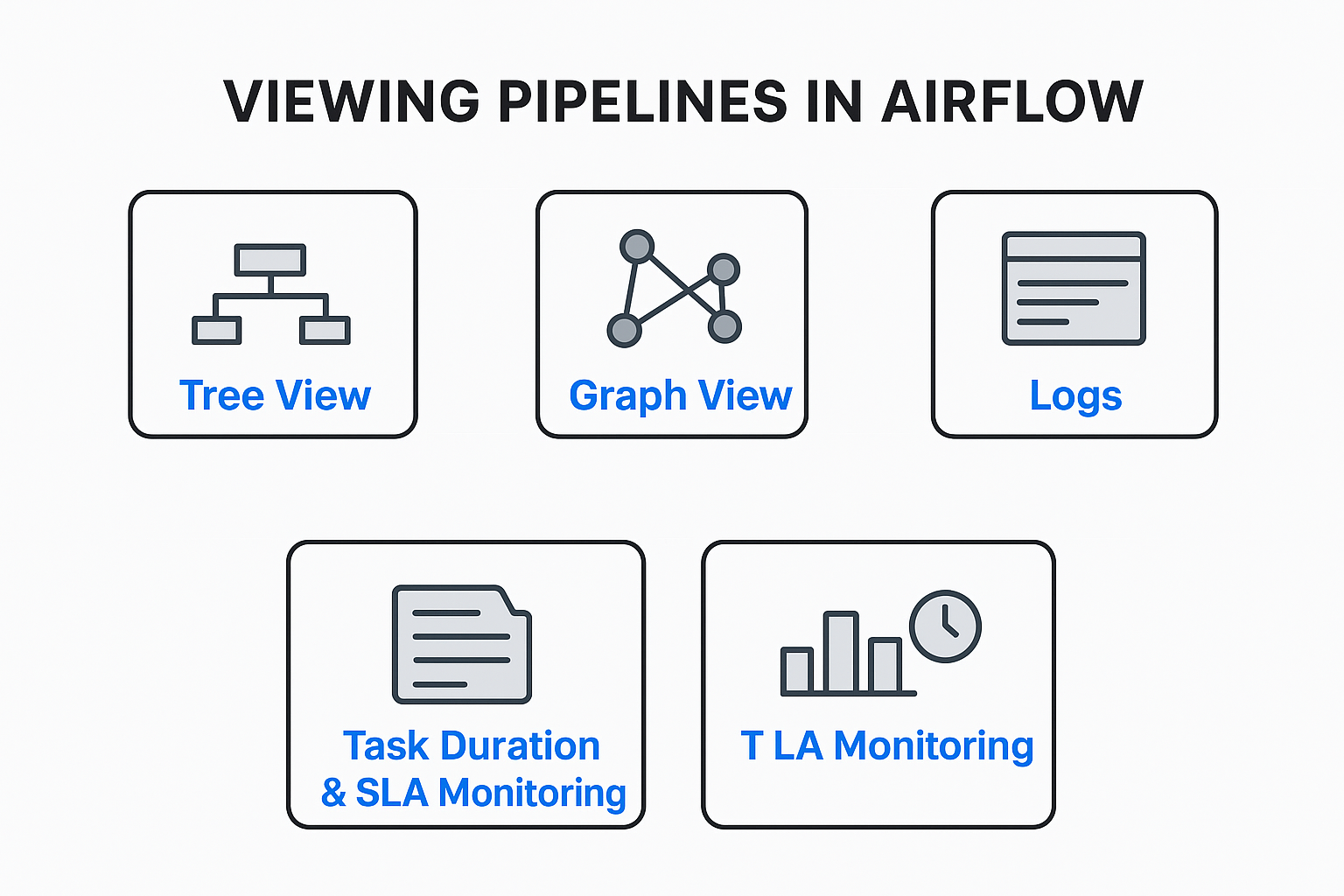
- **Tree View:** Displays task status in a tree format.

- **Graph View:** Shows dependencies and flow of tasks in the DAG.

- **Gantt Chart:** Illustrates task duration and execution timeline.

- **Logs:** Provides detailed logs for each task execution.

- **Task Duration & SLA Monitoring:** Helps track performance metrics.



# 6. BEST PRACTICES

- Keep DAGs modular and avoid writing heavy logic inside tasks.  
- Use clear and meaningful task names.  
- Apply retries, timeouts, and alerts for reliability.  
- Secure credentials using Airflow Connections or Vault.  
- Use version control (Git) for DAG scripts.  
- Regularly optimize and clean the metadata database.