Apache Airflow and Its Components

- **Ashwin Harish P**

What is Apache Airflow?

Apache Airflow is an open-source platform used to programmatically author, schedule, and monitor workflows or data pipelines. It allows you to define workflows as code (using Python), making it flexible, scalable, and maintainable. Airflow helps automate complex computational workflows and ensures tasks are executed in the correct order with dependencies.

Why Airflow?

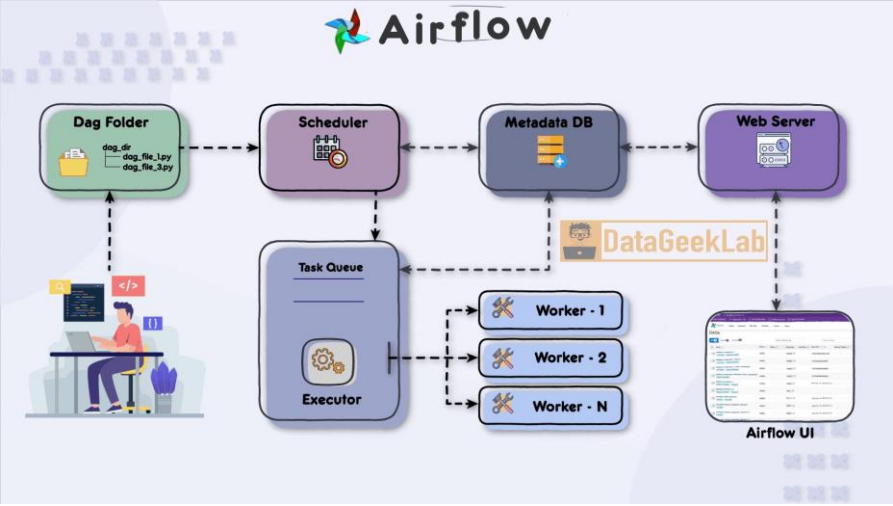
Airflow is a platform for orchestrating batch workflows. It offers a flexible framework with a wide range of built-in operators and makes it easy to integrate with new technologies.

If your workflows have a clear start and end and run on a schedule, they're a great fit for Airflow DAGs.

If you prefer coding over clicking, Airflow is built for you. Defining workflows as Python code provides several key benefits:

- **Version control**: Track changes, roll back to previous versions, and collaborate with your team.
- **Team collaboration**: Multiple developers can work on the same workflow codebase.
- **Testing**: Validate pipeline logic through unit and integration tests.
- **Extensibility**: Customize workflows using a large ecosystem of existing components — or build your own.

Architecture of Airflow

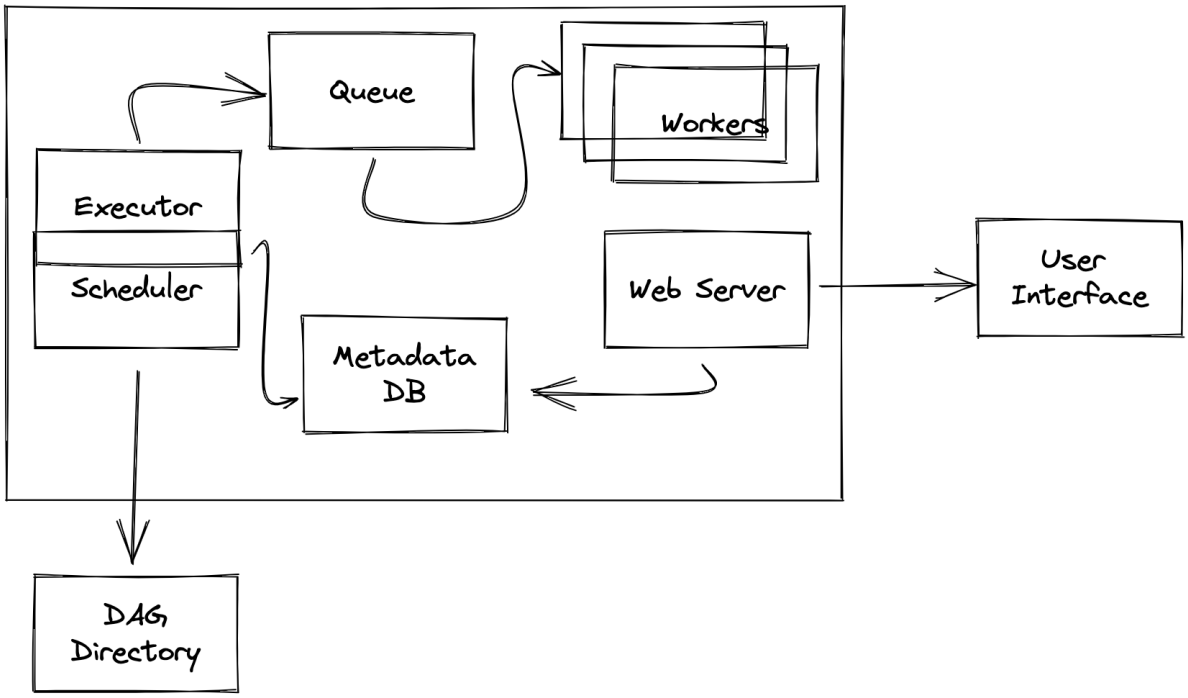


Key Components of Airflow

1. **DAG (Directed Acyclic Graph)**
    - A DAG is a collection of all the tasks you want to run, organized with dependencies and relationships.
    - It represents the workflow or pipeline as a graph where nodes are tasks and edges define dependencies.
    - DAGs are defined in Python scripts.
2. **Task**
    - A single unit of work in a DAG.
    - Tasks can be anything from running a bash command, executing a Python function, transferring files, or triggering another process.
    - Common operators like BashOperator, PythonOperator, etc., are used to define tasks.
3. **Scheduler**
    - The component that monitors DAGs and triggers task instances whose dependencies are met.
    - It decides when to execute tasks based on scheduling intervals or external triggers.
4. **Executor**
    - Responsible for executing the tasks.
    - There are multiple types like SequentialExecutor (for testing), LocalExecutor, CeleryExecutor (for distributed execution), etc.
5. **Metadata Database**
    - Airflow uses a metadata database (typically PostgreSQL or MySQL) to store information about DAGs, task status, schedules, logs, and more.
    - This database is crucial for tracking the state and history of workflows.
6. **Web Server (Airflow UI)**
    - A user-friendly interface to visualize DAGs, monitor task progress, trigger manual runs, and troubleshoot issues.
    - You can see logs, task durations, and manage workflows here.
7. **Workers**

- In setups like CeleryExecutor, workers are machines or processes that pick up tasks and execute them asynchronously.

Architecture Diagram



When to Prefer Airflow Over Databricks Native Pipelines

Scenario	Use Airflow	Use Databricks Native Jobs
Complex workflows with many cross-system dependencies	✅	❌
Need centralized orchestration across multiple platforms	✅	❌
Simple Spark job scheduling and monitoring	❌ (native is sufficient)	✅
Requirement for fine-grained retry, SLA, alerting	✅	Limited
Code-centric, testable pipeline management	✅	UI-driven, less code-centric
Heavy integration with other tools (S3, databases, APIs)	✅	Difficult or limited