# Apache Airflow Overview

## Definition

Apache Airflow is an open-source workflow orchestration platform created by Airbnb and later donated to the Apache Software Foundation. It is used to author, schedule, and monitor complex workflows (data pipelines) as DAGs (Directed Acyclic Graphs) using Python.  
  
“Airflow pipelines are defined in Python, allowing for dynamic pipeline generation.”

## Framework Architecture

The architecture of Airflow is modular, distributed, and scalable. It follows the master-worker pattern.

Key Architecture Components:

Web Server – UI for monitoring DAGs and tasks  
Scheduler – Triggers tasks based on time or event  
Metadata Database – Stores DAGs, logs, task states  
Workers – Executes the tasks  
Executors – Handles how tasks are run (e.g., Celery, Kubernetes)

## Core Components

|  |  |
| --- | --- |
| Component | Description |
| DAG | Represents a workflow; nodes are tasks; edges define dependencies. |
| Task | A unit of work (e.g., Python script, Bash command, SQL job). |
| Operator | Defines the type of task (e.g., BashOperator, PythonOperator). |
| Scheduler | Decides when to execute a task based on time or dependencies. |
| Executor | Handles how tasks are run — LocalExecutor, CeleryExecutor, KubernetesExecutor. |
| Worker | Executes the actual task instances. |
| Web UI | Provides a graphical interface to monitor DAGs and task status. |
| Metadata DB | Stores metadata about DAGs, runs, task states using PostgreSQL/MySQL. |

## Pros of Apache Airflow

• Python-native: Workflows are defined in Python, enabling dynamic generation of tasks.

• Extensible: Custom operators, hooks, sensors can be built.

• Scalable: Supports distributed execution with Celery/Kubernetes.

• Web UI: Rich GUI to monitor DAGs, task duration, retries, etc.

• Modular Architecture: Decouples scheduler, worker, UI for flexibility.

• Integrations: With cloud providers (AWS, GCP, Azure), Databricks, Spark, etc.

## Cons of Apache Airflow

• Complex Setup: Requires setup of database, scheduler, executor, and workers.

• Not Real-time: Best suited for batch jobs, not streaming or event-driven tasks.

• UI Performance: Slows down with many DAGs or high task volume.

• Limited Data Handling: Doesn't process data directly (not an ETL engine itself).

• Scaling Issues: Requires tuning for large deployments (Celery/K8s setup).

## Alternatives & Similar Tools

• Prefect: Python-native orchestration system; simpler setup, event-driven.

• Dagster: Modern tool with software engineering principles; strong typing, testing.

• Luigi (Spotify): Workflow engine from Spotify; good for ETL but lacks UI maturity.

• Kubeflow Pipelines: Pipelines for ML on Kubernetes; best for ML workflows.

• Argo Workflows: Kubernetes-native workflow engine; YAML-based, container-native.

• Google Cloud Composer: Managed Airflow on GCP; easier setup and scaling.

• AWS MWAA: AWS-hosted Airflow; reduces DevOps burden.

## Summary

|  |  |
| --- | --- |
| Aspect | Summary |
| Use Case | Batch workflow orchestration, ETL jobs, data pipelines |
| Language | Python (DAGs defined in Python code) |
| Execution Model | Directed Acyclic Graph (DAG)-based |
| Deployment | Self-hosted (Docker/Kubernetes), or managed (GCP/AWS) |
| Strength | Flexibility, extensibility, rich ecosystem |
| Weakness | High learning curve, complexity in setup |