AIRFLOW

What is Apache Airflow?

Apache Airflow is an open-source platform used to programmatically author, schedule, and monitor workflows. Developed at Airbnb and later donated to the Apache Software Foundation, Airflow enables data engineers to manage complex data pipelines reliably and efficiently.

Workflows are defined as **Directed Acyclic Graphs (DAGs)** of tasks, and each task represents a unit of work. Airflow is designed to handle dependencies between tasks, retry logic, and scheduling of jobs in a highly scalable manner.

Key Features

- **Dynamic Pipelines**: Define workflows in Python.
- **Extensible**: Supports custom plugins and operators.
- **UI & Monitoring**: Rich user interface for monitoring and troubleshooting.
- **Scalable**: Works with Celery, Kubernetes, and other executors for large-scale workflows.
- **Retry & Alerting**: Built-in retry, SLA monitoring, and email alerts.

Airflow Core Components

1. DAG (Directed Acyclic Graph)

A DAG is the central concept in Airflow. It defines the structure of the workflow — how tasks are related and their order of execution.

- Written in Python code.
- Must be **acyclic** (no circular dependencies).
- Scheduled to run at a specific interval (cron-like syntax).

2. Task

A unit of work in a DAG — it can be anything from running a script to calling an API.

- Represented by an **Operator**.
- Executed by a **Worker** process.
- Can be retried on failure.

3. Operator

Defines a single task in a workflow. Airflow provides several operators out-of-the-box:

- **BashOperator**: Executes bash commands.
- **PythonOperator**: Runs a Python function.

- **DummyOperator**: Used as placeholders or to structure DAGs.
- **BranchPythonOperator**: Controls branching logic.
- **Custom Operators**: Can be created to handle custom tasks.

System Components

4. Scheduler

The **heart** of Airflow. It monitors DAGs, triggers task instances, and submits them to the executor.

- Runs continuously in the background.
- Decides when to run a task based on the schedule and DAG state.

5. Executor

Determines how and where a task is run.

- **SequentialExecutor**: For testing.
- **LocalExecutor**: Runs tasks locally in parallel.
- **CeleryExecutor**: Distributed execution using Celery workers.
- **KubernetesExecutor**: Runs each task in a separate pod.

6. Workers

Execute the actual tasks. Used in executors like Celery or Kubernetes.

- Receive tasks from the Scheduler.
- Run the associated command/script.

Supporting Components

7. Web Server

A Flask-based UI to:

- View DAGs and task status.
- Trigger or pause DAGs manually.
- Monitor logs and progress.
- View Gantt charts and task durations.

8. Metadata Database (Metastore)

Stores everything about DAGs, task instances, schedules, and logs.

- Typically uses **PostgreSQL** or **MySQL**.
- Acts as the central state for the system.

- Accessed by Scheduler and Web Server.

9. CLI (Command-Line Interface)

Airflow provides a powerful CLI to interact with the system.

- Example commands: airflow dags list, airflow tasks run, airflow db init.

Execution Flow Summary

1. DAGs are loaded from Python files into the Airflow scheduler.
2. Scheduler evaluates DAGs and schedules tasks.
3. Executor assigns tasks to workers.
4. Workers execute the tasks and report results to the metadata DB.
5. The Web UI displays real-time updates.

Common Use Cases

- ETL pipelines
- Data warehouse loading (e.g., Snowflake, BigQuery)
- Machine learning workflows
- Automated reporting
- API and data validation tasks

Conclusion

Apache Airflow is a robust and flexible workflow orchestration tool that plays a key role in modern data engineering. With its modular architecture and ability to scale, it's widely adopted in production environments across industries.