Name: Simantini M. Patil Data Engineering Batch 2

# AIRFLOW

## What is Apache Airflow?

Apache Airflow is an open-source **workflow orchestration tool** used to programmatically author, schedule, and monitor workflows (also called *DAGs*). Developed by Airbnb and later donated to the Apache Foundation, Airflow is widely adopted in data engineering, ETL (Extract, Transform, Load) pipelines, and machine learning workflows.

### Key Features

- **Dynamic Pipeline Generation**: Workflows are defined in Python code.
- **Scalable & Extensible**: Plugins and operators can extend functionality.
- **UI Monitoring**: Track progress, logs, and task dependencies visually.
- **Scheduler and Executor Architecture**: Manages and distributes task execution.
- **Retries, Logging & SLA**: Supports retries on failure, audit logs, and service level agreements.

## Basic Terminologies

| **Term** | **Description** |
|---|---|
| **DAG** | *Directed Acyclic Graph* that represents a workflow; contains task sequence. |
| **Task** | A single unit of work (e.g., Bash command, Python function, SQL query). |
| **Operator** | Defines what a task does (e.g., PythonOperator, BashOperator). |
| **Executor** | Defines how tasks are executed (e.g., LocalExecutor, CeleryExecutor). |
| **Scheduler** | Determines when each task runs, based on DAG schedule. |
| **Webserver** | Provides a web-based UI to monitor DAGs and tasks. |
| **Metadata DB** | Stores DAG structure, task status, logs, schedules, and configuration. |

## Airflow Architecture Components

### 1. Web Server

- UI to monitor DAGs, task logs, execution status.
- Enables triggering, pausing, and retrying DAGs manually.

### 2. Scheduler

- Watches DAGs and schedules tasks based on their dependencies and cron-like intervals.
- Places executable tasks into a queue.

**3. Executor**

- Pulls tasks from the queue and runs them.
- Multiple types:
  - SequentialExecutor: Single-threaded (for testing).
  - LocalExecutor: Parallel task execution on the same machine.
  - CeleryExecutor: Distributed task execution using Celery and message brokers (e.g., RabbitMQ).
  - KubernetesExecutor: Launches each task in its own Kubernetes pod.

**4. Worker**

- Executes the task code.
- Required in distributed executors like Celery or Kubernetes.

**5. Metadata Database**

- Stores DAG runs, task status, variables, connections, and logs.
- Backed by RDBMS (e.g., PostgreSQL or MySQL).

## Airflow Workflow Lifecycle

1. **DAG Definition**: Written in Python using @dag or standard syntax.
2. **DAG Parsing**: Scheduler parses DAGs and understands dependencies.
3. **Scheduling**: Scheduler queues tasks based on schedule.
4. **Execution**: Executor picks tasks and hands off to workers.
5. **Monitoring**: UI and logs track execution and errors.

## 🔌 Airflow Plugins & Operators

- **Operators**:
  - PythonOperator: Executes Python function.
  - BashOperator: Executes shell commands.
  - EmailOperator, DockerOperator, SQL Operators, etc.
- **Sensors**: Wait for a condition to be met (e.g., file existence).
- **Hooks**: Interface to external services (e.g., AWS, GCP, S3, Hive).

**Use Cases**

- Data pipeline automation
- ETL processing
- Machine Learning model training
- Report generation and delivery
- Periodic batch jobs

Let’s understand Airflow with the help of an example:

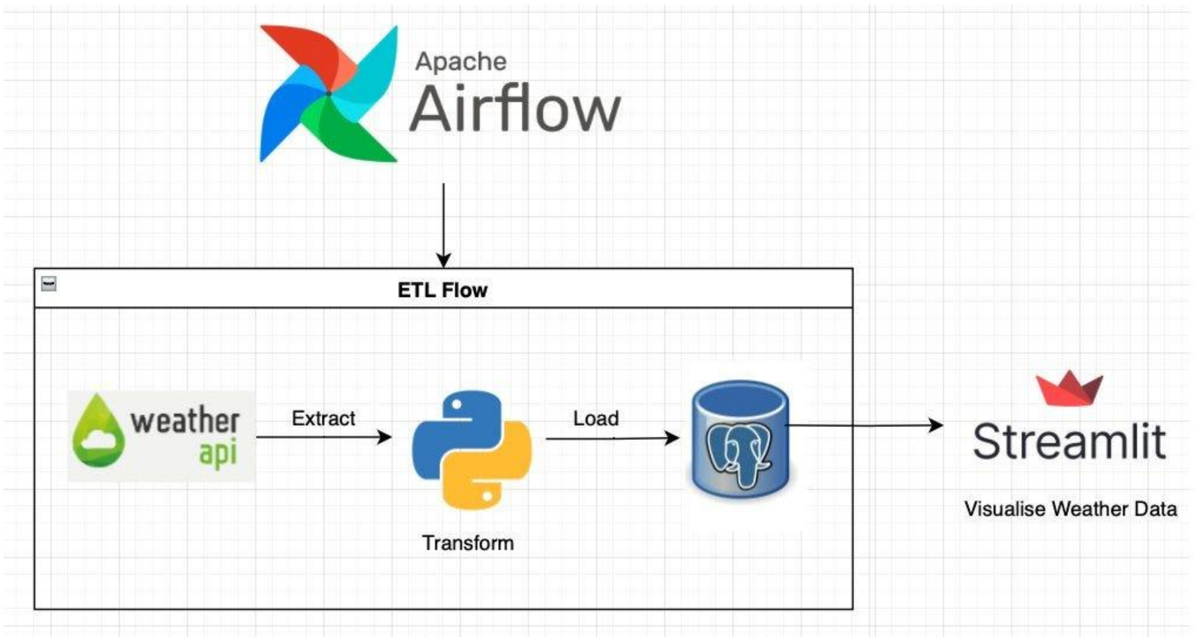

Fig: Apache Airflow Example

This diagram illustrates an **end-to-end data pipeline using Apache Airflow** to manage a weather data ETL (Extract, Transform, Load) workflow, which is then visualized using **Streamlit**.

**Goal**:
Automate fetching weather data from an API, process and store it in a database, and visualize it via a dashboard.

**1. Apache Airflow (Top Component)**

Airflow is orchestrating the **ETL pipeline**, which includes scheduling and managing the entire data flow.

- Airflow triggers and coordinates the ETL process.

**2. ETL Flow Box (Main Processing Section)**

- **Extract**
  - **Source**: weather API
  - Airflow uses an API call to **extract weather data** (like temperature, humidity, wind speed, etc.).
- **Transform**
  - **Tool**: Python (represented by the Python logo)
  - Extracted data is passed to a Python script for **cleaning, transformation**, and restructuring:
    - Convert formats (e.g., timestamps, units)
    - Filter unnecessary fields
    - Add computed columns (e.g., heat index)
- **Load**
  - **Target**: PostgreSQL (or a similar relational database)
  - The transformed data is **loaded into a database** for persistent storage and future querying.

**3. Streamlit (Rightmost Component)**

- Streamlit is used to **create an interactive web dashboard**.
- It connects to the database to **fetch and display weather data** in real-time (charts, tables, etc.).

Imagine this pipeline running every 30 minutes. Airflow ensures new weather data is regularly pulled, processed, and displayed for users (e.g., city weather dashboard, agriculture planning tool, etc.).