# **Apache Airflow: Features, Pipeline Building, and Views**

## **1. Introduction to Apache Airflow**

Apache Airflow is an **open-source workflow orchestration tool** that helps to programmatically author, schedule, and monitor data pipelines. It is widely used in data engineering, ETL, and machine learning workflows.

## **2. Key Features of Apache Airflow**

1. **Dynamic DAGs (Directed Acyclic Graphs)** Workflows are defined as Python code, making them dynamic and easy to maintain.
2. **Scalability** Airflow can scale from a single machine to distributed clusters using Celery, Kubernetes, or other executors.
3. **Extensible Operators** Provides built-in operators (e.g., BashOperator, PythonOperator, PostgresOperator) and allows custom operator creation.
4. **Rich UI** Provides a web-based UI to visualize DAGs, monitor tasks, and check logs.
5. **Scheduler** Handles time-based and event-based scheduling of workflows.
6. **Robust Monitoring** Retry policies, SLA monitoring, alerting, and logging capabilities.
7. **Integration Support** Works with databases, cloud services (AWS, GCP, Azure), and big data frameworks (Spark, Hadoop).

## 

## **3. Core Components of Airflow**

* **DAG (Directed Acyclic Graph):** Defines the pipeline and dependencies between tasks.
* **Task:** A single unit of work within a DAG.
* **Operator:** A template that defines the work (e.g., run SQL, execute Python, call API).
* **Executor:** Handles task execution (LocalExecutor, CeleryExecutor, KubernetesExecutor).
* **Scheduler:** Decides when tasks should run.
* **Web UI:** Provides monitoring and management.
* **Metadata Database:** Stores DAGs, task states, and logs.

## **4. Steps to Build a Pipeline in Airflow**

### **Step 1: Install & Setup**

* Install via **Docker Compose** or pip.
* Start Airflow services (scheduler, webserver, workers).

### **Step 2: Create a DAG File**

* Write Python code defining DAG, schedule, and tasks.

Example DAG structure:  
  
 from airflow import DAG

from airflow.operators.bash import BashOperator

from airflow.utils.dates import days\_ago

with DAG("example\_dag", start\_date=days\_ago(1), schedule\_interval=None, catchup=False) as dag:

task1 = BashOperator(

task\_id="print\_date",

bash\_command="date"

)

### **Step 3: Define Tasks**

* Use operators to define tasks (BashOperator, PythonOperator, PostgresOperator).

### **Step 4: Set Task Dependencies**

Use >> or << operators to define execution order.  
  
 task1 >> task2

### **Step 5: Place DAG in dags/ Folder**

* Save your DAG file into the dags directory.
* Airflow automatically detects new DAGs.

### **Step 6: Run the Pipeline**

* Start the webserver (localhost:8080).
* Trigger DAG manually or wait for scheduled run.

### **Step 7: Monitor Execution**

* Check task logs, retries, and failures in the UI.

## **5. Important Views in Airflow UI**

1. **DAGs View** – List of available DAGs with options to enable/disable and trigger runs.
2. **Grid View** – Timeline of DAG runs and task execution states.
3. **Graph View** – DAG visualized as a graph showing task dependencies.
4. **Calendar View** – Execution history in a calendar format.
5. **Gantt View** – Execution duration and overlap visualization.
6. **Task Instance View** – Logs, retries, and execution details of each task.

## **6. Example: Simple Data Pipeline**

A basic ETL pipeline in Airflow:

1. **Download CSV file** using BashOperator.
2. **Create staging table** in Postgres using PostgresOperator.
3. **Load CSV into staging**.
4. **Transform & clean data**.
5. **Insert into final table**.

## **7. Conclusion**

Apache Airflow provides a powerful way to **author, schedule, and monitor pipelines** with a flexible Python-based approach. Its UI, extensibility, and integrations make it a preferred tool in the modern data engineering stack.