HEXAWARE TRAINING

APACHE AIRFLOW AND ITS COMPONENTS

1. Introduction to Apache Airflow

Apache Airflow is an open-source platform developed by Airbnb and now maintained under the Apache Software Foundation. It is designed to programmatically author, schedule, and monitor workflows. Airflow is widely used in data engineering, ETL (Extract, Transform, Load) pipelines, machine learning model orchestration, and other complex job scheduling tasks.

Key Features:

- Dynamic pipeline generation via Python code
- Scalable architecture (executor-based)
- Robust monitoring UI
- Extensibility via plugins
- Supports retries, SLA, logging, and alerting

2. Architecture of Apache Airflow

Apache Airflow follows a **modular and scalable architecture** composed of several components that work in coordination. The major components include:

2.1 Scheduler

The Scheduler is the brain of Airflow. It monitors DAG definitions and schedules the DAG runs. Based on the schedule intervals defined in the DAGs, it triggers task instances by placing them in a queue for execution.

Responsibilities:

- Parsing DAGs
- Scheduling task instances
- Creating DAG runs at scheduled intervals
- Notifying the executor about tasks ready for execution

2.2 Web Server (UI)

The Web Server provides a rich and interactive UI to visualize, manage, and monitor workflows. It is built using Flask.

Features:

- View DAGs, task status, logs, and timelines

- Trigger DAG runs manually
- View task instance details
- Pause/resume DAGs
- Monitor performance and failures

2.3 Metadata Database

Airflow uses a metadata database (commonly PostgreSQL or MySQL) to store all configuration information, DAG runs, task statuses, logs, and other metadata.

Key tables in the DB:

- dag: information about DAGs
- dag_run: metadata for each DAG execution
- task_instance: status of each task
- log, variable, connection, etc.

This database is central to the functioning of Airflow as all other components rely on it for task state information.

2.4 Executor

The Executor is responsible for executing the tasks. Depending on the type of executor configured, it determines how and where the task is executed.

Types of Executors:

- **SequentialExecutor**: For testing/debugging; runs one task at a time.
- **LocalExecutor**: Runs multiple tasks in parallel on a single machine.
- **CeleryExecutor**: Distributed execution using Celery workers.
- **KubernetesExecutor**: Runs each task in its own Kubernetes pod.

3. Core Concepts in Airflow

3.1 DAG (Directed Acyclic Graph)

A DAG is a collection of tasks with defined relationships and execution order. It is defined in Python scripts and ensures that the workflow has no loops.

Example:

```
from airflow import DAG
from airflow.operators.bash_operator import BashOperator
from datetime import datetime

with DAG('sample_dag', start_date=datetime(2024, 1, 1), schedule_interval='@daily') as dag:
    task1 = BashOperator(task_id='print_hello', bash_command='echo Hello')
    task2 = BashOperator(task_id='print_world', bash_command='echo World')
    task1 >> task2  # sets task2 to run after task1
```

3.2 Task

A Task is a single unit of work in a DAG. It is created using Operators (e.g., BashOperator, PythonOperator, etc.) and represents one node in the DAG.

3.3 Operators

Operators are building blocks used to define the tasks. Airflow provides a wide range of operators:

- **BashOperator** – Executes a bash command.
- **PythonOperator** – Runs a Python function.
- **EmailOperator** – Sends emails.
- **DummyOperator** – Used as a placeholder or for control flow.
- **BranchPythonOperator** – Implements conditional logic (like if-else).

4. Use Cases of Airflow

- **ETL Pipelines**: Automating daily data ingestion, transformation, and loading into data warehouses.
- **Machine Learning**: Model training, validation, and deployment in a sequential workflow.
- **Data Quality Monitoring**: Scheduled checks on data accuracy, completeness, and consistency.
- **DevOps Automation**: Triggering builds, backups, or system checks.

5. Airflow Advantages and Limitations

Advantages

- Pure Python workflow definitions
- Easy to monitor and retry failed tasks
- Excellent community and extensibility
- Suitable for complex workflows with dependencies

Limitations

- Steep learning curve for beginners
- Not suitable for low-latency real-time processing
- Requires separate setup for distributed execution

6. Conclusion

Apache Airflow provides a powerful, flexible, and scalable solution for orchestrating complex workflows in data pipelines. With its modular components and Python-based configuration, it has become a standard choice for modern data engineering and analytics teams.

Understanding its components—such as the Scheduler, Web UI, Metadata DB, and Executors—is key to successfully building and deploying reliable workflows in production environments.