HEXAWARE TRAINING

INTRODUCTION AND CORE CONCEPTS

What is Apache Airflow?

Apache Airflow is a platform to **programmatically author, schedule, and monitor workflows**. It lets you define workflows as **Python code**, visualize them as **DAGs (Directed Acyclic Graphs)**, and track their execution.

Why Use Airflow?

- Dynamic pipeline creation using code
- Clear dependency management
- Scalable with multiple execution backends
- Centralized logging and alerting
- Built-in retry, SLA, and scheduling logic

Core Terminology

Concept	Description
DAG	A Directed Acyclic Graph – defines the workflow structure.
Task	A unit of work – one node in the DAG.
Operator	Defines the nature of the task (e.g., BashOperator, PythonOperator).
Task Instance	A run of a task for a particular DAG run and time.

Example DAG

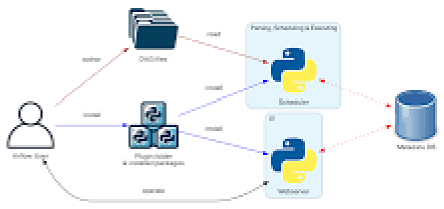
```
from airflow import DAG
from airflow.operators.bash import BashOperator
from datetime import datetime

with DAG('my_workflow', start_date=datetime(2024, 1, 1),
schedule_interval='@daily') as dag:
    start = BashOperator(task_id='start', bash_command='echo Start')
    end = BashOperator(task_id='end', bash_command='echo End')
    start >> end
```

The >> operator defines the execution order.

AIRFLOW ARCHITECTURE AND COMPONENTS

1. Airflow Architecture

Airflow is made up of **four main components**:



1. Scheduler

- Monitors DAGs and schedules task execution.
- Pushes tasks to the executor when they are ready.

2. Web Server (UI)

- Flask-based UI for DAG visualization.
- Allows manual triggering, monitoring, and error inspection.

3. Metadata Database

- Stores DAG structure, run history, task instances, variables, and logs.
- Typically uses PostgreSQL or MySQL.

4. Executor

- Executes the tasks using workers.
- Determines how tasks are run (locally, in parallel, or in distributed systems).

2. Types of Executors

Executor	Description
SequentialExecutor	Runs one task at a time – good for testing
LocalExecutor	Parallel task execution on one machine
CeleryExecutor	Distributed task execution using Celery workers
KubernetesExecutor	Spawns a Kubernetes pod for each task

Extended Components

- **Workers**: Used with Celery/Kubernetes to run tasks.
- **Message Broker**: Like Redis or RabbitMQ for task queueing.
- **Flower**: UI tool to monitor Celery workers.

USE CASES, ADVANTAGES, AND BEST PRACTICES

Real-World Use Cases

Area	Example
ETL Pipelines	Daily ingestion → transformation → data warehouse load
ML Workflows	Model training → evaluation → deployment
Reporting	Scheduled report generation and delivery
System Automation	Backups, data sync, email alerts

Advantages of Apache Airflow

- Pythonic: Define workflows in Python
- Scalable: Multiple executor options
- UI-rich: Monitor every task's status
- Extensible: Plugins and custom operators
- Resilient: Retry, backfill, alerting

Limitations to Be Aware Of

- Learning curve for DAG dependencies and Jinja templating
- Not ideal for real-time (low-latency) data processing
- Requires external setup for distributed execution (e.g., Celery + Redis)

Best Practices

- Keep DAGs modular and readable
- Use task retries with exponential backoff
- Externalize config using Airflow Variables/Connections
- Monitor with SLAs and alerts
- Separate heavy logic from the DAG file (use Python scripts/functions)
-

Apache Airflow offers a powerful, production-grade solution for automating and managing workflows. By mastering its components—DAGs, Operators, Scheduler, Executors, and UI—you unlock robust data orchestration tailored to your business logic.