AIRFLOW FEATURES AND BUILDING PIPELINE

Apache Airflow is an open source workflow orchestration platform designed to programmatically author, schedule, and monitor workflows. It is widely used in data engineering and analytics projects for building and managing ETL pipelines, automating repetitive tasks, and ensuring reliable execution of complex workflows. With Python-based DAGs and a feature rich UI, Airflow simplifies pipeline creation, monitoring, and troubleshooting.

Key Features of Apache Airflow

1. **Workflow Orchestration**
    - Automates and schedules complex workflows.
    - Manages dependencies between tasks.
2. **Dynamic Pipelines**
    - Workflows are defined as Python code, allowing dynamic generation of DAGs (Directed Acyclic Graphs).
3. **Monitoring and Management**
    - Web UI for tracking task status (running, success, failed).
    - Provides logs and retry mechanisms.

Steps to Build a Pipeline in Airflow

1. **Set Up Environment**
    - Install Apache Airflow using pip or Docker.
    - Configure Airflow home directory and database.
2. **Create a DAG**
    - Define DAG structure in a Python script.
    - Example parameters: dag_id, start_date, schedule_interval.
3. **Define Tasks**
    - Use Operators (e.g., PythonOperator, BashOperator, SQL operators).

- Each task performs a specific unit of work.

4. **Set Task Dependencies**
    - Define the sequence of execution using >> (downstream) or << (upstream).
    - Example: task1 >> task2 ensures task1 runs before task2.
5. **Deploy DAG**
    - Place the DAG Python file in the dags/ folder.
    - Airflow automatically detects new DAGs.
6. **Trigger and Run Pipeline**
    - Trigger manually via CLI (airflow dags trigger <dag_id>) or UI.
    - Scheduled runs based on schedule_interval.
7. **Monitor Execution**
    - Track status (success, failed, queued, running) in Airflow UI.
    - Check logs for debugging.

Viewing Pipelines in Airflow UI

1. **DAGs View**
    - Lists all available DAGs with options to enable/disable scheduling.
    - Provides controls for triggering runs manually.
2. **Graph View**
    - Visual representation of the pipeline.
    - Shows task dependencies in DAG format.
3. **Tree View**
    - Hierarchical view of DAG runs.
    - Useful for monitoring multiple runs over time.
4. **Task Instance Details**

- Logs, retries, execution times, and state for each task.

5. **Gantt Chart View**
    - Visualizes task durations and dependencies.
    - Helps analyze bottlenecks.

Example DAG Code

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


# Define the DAG
default_args = {
    'owner': 'airflow',
    'depends_on_past': False,
    'retries': 1
}


dag = DAG(
    dag_id='example_dag',
    default_args=default_args,
    description='A simple example DAG',
    schedule_interval='@daily',
    start_date=datetime(2025, 1, 1),
    catchup=False
)
```

```
# Define tasks
task1 = BashOperator(
    task_id='print_date',
    bash_command='date',
    dag=dag
)

task2 = BashOperator(
    task_id='print_message',
    bash_command='echo "Hello from Airflow!"',
    dag=dag
)

# Set task dependencies
task1 >> task2
```

This DAG contains two tasks: one prints the current date, and the other prints a message. Task dependencies ensure that the date is printed before the message.

With simple DAG definitions, flexible scheduling, and detailed monitoring, Airflow provides data engineers with the control and visibility needed for reliable pipeline execution.