|  |  |  |
| --- | --- | --- |
| Ex No: 6 | | Apache Airflow – Workflow Orchestration using Docker |
| Date: 10/10/25 |  | |

**Objective:**

This lab experiment provides hands-on experience in setting up and orchestrating data workflows using Apache Airflow. Students will learn to define, schedule, and monitor Directed Acyclic Graphs (DAGs) using Airflow operators, executors, and schedulers. The lab also introduces workflow automation in data engineering pipelines with containerized Airflow setup using Docker.

**Outcomes:**

* Understand the core architecture and components of Apache Airflow.
* Define and implement Directed Acyclic Graphs (DAGs) for workflow orchestration.
* Deploy and run Airflow using Docker and docker-compose.
* Monitor workflows using the Airflow web UI.
* Manage data pipeline tasks such as extraction, transformation, and loading (ETL).

**Materials:**

* Docker and Docker Compose installed on the system.
* Apache Airflow Docker image.
* Python environment for DAG scripting.
* Sample ETL scripts or data files for workflow tasks.

**Lab Procedure:**

1. **Introduction to Apache Airflow**

Apache Airflow is an open-source platform used to programmatically author, schedule, and monitor workflows. It is based on Directed Acyclic Graphs (DAGs) that define the order and dependency of tasks.

1. **Key Components of Airflow**

* DAG (Directed Acyclic Graph): Defines the workflow as a graph of tasks and dependencies, written in Python.
* Task: A single unit of work in a DAG, represented by an Operator.
* Operator: Defines the type of work a task performs. Examples: BashOperator, PythonOperator, SensorOperator.
* Scheduler: Schedules DAG runs and submits them to the executor.
* Executor: Determines how and where tasks are executed
* Workers: Machines or containers that actually run the task commands.
* Web Server (UI): A web-based dashboard for monitoring, triggering, and debugging DAGs.
* Metadata Database: Stores state information about DAGs, tasks, and runs.

1. **Setting up Apache Airflow using Docker**

1. Create a Dockerfile and docker-compose.yml for Airflow setup.  
2. Build the Airflow Docker image using:  
 docker build -t airflowsqlserver -f Dockerfile --no-cache .  
3. Run Airflow containers using:  
 docker-compose up  
4. Access the Airflow web interface at:  
 http://localhost:9099/home

1. **Creating and Deploying a DAG**

1. Write a Python script defining a simple ETL DAG (extract → transform → load).  
2. Save the DAG file in the Airflow ‘dags/’ directory.  
3. Trigger the DAG manually or let the scheduler handle execution.  
4. Monitor progress through the Airflow web UI and check logs for each task.

1. **Database Setup and Permissions**

Run SQL commands inside Airflow’s connected database (PostgreSQL ):  
CREATE SCHEMA IF NOT EXISTS etl\_staging;  
GRANT ALL PRIVILEGES ON SCHEMA etl\_staging TO etl;

1. **Verification and Monitoring**

Use the Airflow UI to verify task execution status (success, failed, queued, etc.). Inspect logs and confirm the DAG completed successfully.

**Result:**

The Apache Airflow environment was successfully set up using Docker. A sample ETL DAG was created, scheduled, and monitored using the Airflow web interface, demonstrating workflow orchestration and task management.

**Github Link :**