

Ex No: 6	Apache Airflow
Date:	

Objective:

The objective of this lab is to familiarize students with the fundamental concepts and practical implementation of **Apache Airflow** for workflow automation and data pipeline orchestration. Through this exercise, students will learn to understand Airflow's core components, including DAGs, tasks, operators, the scheduler, executor, and metadata database. They will gain hands-on experience in setting up and running Airflow using Docker, creating and deploying a simple Python-based DAG, and managing task dependencies and retries. Additionally, students will explore how to trigger, monitor, and analyze workflow runs through the Airflow Web UI, thereby developing the skills needed to design, schedule, and monitor data pipelines in a real-world environment.

Outcomes:

1. Explain core Airflow components: DAG, Task/Operator, Scheduler, Executor, Worker, Web UI and Metadata DB.
2. Deploy Airflow locally in Docker and access the Airflow web UI.
3. Write a valid Airflow DAG in Python containing at least three tasks (e.g., extract → transform → load pattern).
4. Configure task dependencies and basic retry behavior, and check logs for debugging.
5. Trigger DAGs manually and interpret run history and task states in the UI.

Materials

Computer with Docker & Docker Compose installed (Windows / Mac / Linux).

The `docker-compose.yml` and a sample `Dockerfile` (or use official `apache/airflow` image).

Text editor / IDE (VS Code, PyCharm, etc.).

Python 3.9+ (for authoring DAGs locally).

Example DAG file (provided below).

Browser to access Airflow UI (e.g., <http://localhost:9099/home> if using your compose).

(Optional) PostgreSQL or MySQL if configuring an external metadata DB — otherwise use default SQLite for local experiments.

Lab Procedure

A. Start Airflow with Docker Compose

1. From the lab folder with the docker-compose.yml, run:
 - docker-compose build (if you have a Dockerfile)
 - docker-compose up -d
2. Confirm containers are running: docker-compose ps or docker ps.
3. Open the Airflow UI in your browser (e.g., <http://localhost:9099/home>).

B. Deploy the DAG

1. Place example_lab6_dag.py into the host dags/ folder which is mounted to the Airflow dags directory.
2. In the Airflow UI, go to DAGs and refresh if needed. You should see lab6_etl_example.

C. Trigger & Monitor

1. Turn the DAG ON (toggle switch) and click Trigger DAG (or trigger a DagRun manually).
2. Open the Graph View to see task nodes and dependencies.
3. Click a task to view logs; examine stdout printed by the PythonOperator.
4. Note task states: queued → running → success (or failed if there is an error).
5. Review retry behaviour by intentionally causing an error (optional): change transform to raise an exception and watch retry.

D. Inspect Metadata DB (optional)

1. If using PostgreSQL/MySQL, connect to the DB and inspect the dag_run and task_instance tables to see run metadata. Your slides show how a metadata DB stores run information.

E. Cleanup

1. Stop containers: docker-compose down (use -v to remove volumes if desired).
2. Remove temporary files if created.

USN NUMBER:1RVU23CSE153

NAME:Ekshu DP

Results:

6. lab6_etl_example DAG appears in the Airflow UI.
7. The three tasks run in order: extract_task → transform_task → load_task.
8. Console/log output displays the simulated extract/transform/load prints.
9. Task logs and run history visible via the UI.
10. If an intentional error is introduced, the task will retry according to retries and retry_delay and the UI shows attempts.

```
Microsoft Windows [Version 10.0.26200.6725]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Pradeep>cd C:\RVU\Data engineering\Lab-6\Lab-6

C:\RVU\Data engineering\Lab-6\Lab-6>docker build -t airflowsqlserver -f Dockerfile --no-cache .
[+] Building 308.1s (6/6) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile                0.1s
=> => transferring dockerfile: 391B                                0.0s
=> [internal] load metadata for docker.io/apache/airflow:2.2.3    7.0s
=> [internal] load dockerignore                                     0.0s
=> transferring context: 2B                                         0.0s
=> [1/2] FROM docker.io/apache/airflow:2.2.3@sha256:e3fa6175d0f43f1e9b4ed331a69f219523925186909330d7caac827bdb 194.6s
=> => resolve docker.io/apache/airflow:2.2.3@sha256:e3fa6175d0f43f1e9b4ed331a69f219523925186909330d7caac827bdb 0.0s
=> sha256:dd2eec19a3cc99786e0e59b8563a1a6f1296c73a5d045feef139d33d614175b 23.55kB / 23.55kB 0.0s
=> sha256:0f22b8fd766b3ca3172fe1c5efa0d80b3fba9eed626e93cc99b1430fb5c50ddd 9.69MB / 9.69MB 7.9s
=> sha256:70603db774aee52ed14b574b23f065059b1e4f8a5afd4fb18fab130b5babb0c1 3.15kB / 3.15kB 0.0s
=> sha256:e3fa6175d0f43f1e9b4ed331a69f219523925186909330d7caac827bdbc2ceb 856B / 856B 0.0s
=> sha256:ffbb094f4f9e7c61d97c2b409f3e8154e2621a5074a0087d35f1849e665d0d34 27.15MB / 27.15MB 55.6s
=> sha256:2f746edc7f5a90ff637067db7635196acd5dd52be3b1a1a3e2071cdaaf4fed2 2.77MB / 2.77MB 4.3s
=> sha256:6209cd0804fbc0d85913bcd03a540b7d61928581d419c7a95dcf526323c220abf 232B / 232B 5.6s
=> sha256:740db2b47347f97fe2073eb630e8270d2025585b9fedae015c960a5e113f67f 2.50MB / 2.50MB 9.1s
=> sha256:723eafab37772a5567b7f2984d1b49e2c06c665b2acc8f01572592a2acaf60a9 43.73MB / 43.73MB 61.9s
=> sha256:da5409cf2d9e810486f151ad62857c351c7f73fd0869cf29953ad7f66eaf84c3 1.65kB / 1.65kB 10.2s
=> sha256:4126489c095cbb19a82b5ae10f5f11ca4295fe27701f78954189d80b184196 26.89MB / 26.89MB 78.3s
=> extracting sha256:ffbb094f4f9e7c61d97c2b409f3e8154e2621a5074a0087d35f1849e665d0d34 3.7s
=> sha256:30d53234592d7e87754fa9d5676d9bb30ebc95ce99d06ef7d9736b0970745ce8 175.97MB / 175.97MB 171.5s
=> extracting sha256:2f746edc7f5a90ff637067db7635196acd5dd52be3b1a1a3e2071cdaaf4fed2 0.4s
=> extracting sha256:0f22b8fd766b3ca3172fe1c5efa0d80b3fba9eed626e93cc99b1430fb5c50ddd 1.2s
=> extracting sha256:6209cd0804fbc0d85913bcd03a540b7d61928581d419c7a95dcf526323c220abf 0.0s
=> extracting sha256:740db2b47347f97fe2073eb630e8270d2025585b9fedae015c960a5e113f67f 0.8s
=> sha256:ead3608791c346e67ec429cabb27c37177a48e3a25cd4589652f464cf04198ac 4.51kB / 4.51kB 63.1s
=> extracting sha256:723eafab37772a5567b7f2984d1b49e2c06c665b2acc8f01572592a2acaf60a9 3.6s
=> sha256:97a57dd5b8ee2a54d50d98bafb89d69810345c1c2aa5df160d2ca76cda6ca 900B / 900B 64.4s
=> sha256:d8ab7119028eaab11b26c7524ed7ae3e202a5e351fd05355cf89a6a542e03660 6.08kB / 6.08kB 65.9s
=> sha256:231580b9293c6b2368634ad533adad1e9bc95fd3e90875a38f0e7ac6f3858866 535B / 535B 67.2s
=> extracting sha256:da5409cf2d9e810486f151ad62857c351c7f73fd0869cf29953ad7f66eaf84c3 0.0s
=> sha256:1d66cb1a86e1f91d440f61867b4b861d15de89e338303d1c32e63b454b119f2 3.31kB / 3.31kB 68.6s
=> sha256:286f1e773be047d43f630780af4b3403eee060239f661cdaa4811307ed623371 173B / 173B 70.1s
=> extracting sha256:4126489c095cbb19a82b5ae10f5f11ca4295fe27701f78954189d80b184196 1.1s
=> extracting sha256:30d53234592d7e87754fa9d5676d9bb30ebc95ce99d06ef7d9736b0970745ce8 21.6s
=> sha256:ead3608791c346e67ec429cabb27c37177a48e3a25cd4589652f464cf04198ac 0.0s
=> extracting sha256:97a57dd5b8ee2a54d50d98bafb89d69810345c1c2aa5df160d2ca76cda6ca 0.0s
=> extracting sha256:d8ab7119028eaab11b26c7524ed7ae3e202a5e351fd05355cf89e6a542e03660 0.0s
```