

<b>Ex No:</b> 6	<b>Apache Airflow</b>
<b>Date:</b>	

**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:
  - o docker-compose build (if you have a Dockerfile)
  - o 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.

## 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.

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

C:\RVU\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 dockeringone                                         0.0s
=> => transferring context: 2B                                             0.0s
=> [1/2] FROM docker.io/apache/airflow:2.2.3@sha256:c3fa175d0f4f3f1e9b4ed331a69f219523925186999330d7caac827bdb 194.6s
=> =>  resolve docker.io/apache/airflow:2.2.3@sha256:c3fa175d0f4f3f1e9b4ed331a69f219523925186999330d7caac827bdbc2 0.0s
=> => sha256:dd2eec19a3cc9978a66e59b8563ala6f1296c73a54d045feef139d33d14175b 23.55kB / 23.55kB
=> => sha256:0f22b8fd766b3ca3172fe1cefa088b3fb9a9ed626e93c99b1430fb5c56ddd 9.69MB / 9.69MB
=> => sha256:76603dbd774a5e2ed14b5740b3f065059b1e4f8a5a+04f618faeb130b5ab6c1 3.15KB / 3.15KB
=> => sha256:c3fa175d0f4f3f1e9b4ed331a69f219523925186999330d7caac827bdb2ceeb 856B / 856B
=> => sha256:ffb694ff4ff9e7c61d97c2b409f3e8154e2621a5074a087d35f1849e665d0d34 27.15MB / 27.15MB
=> => sha256:2f746edec7f5a90ff637067d7b7635196acdff5dd52be3b1a1a3e2071cdaa4fed2 2.77MB / 2.77MB
=> => sha256:6290ed4804fbcdd85913bc0d83a540b7d61928581d419c7a95dcf526323c220abf 232B / 232B
=> => sha256:749d2b7f7347f97fe2073eabb632e8270d24925585b9fedea015c960a5e113f67f 2.50MB / 2.50MB
=> => sha256:733eafab37f72a5567b7f29841b9e2c46c665b2eccf01573592a2aca60a9 33.73MB / 43.73MB
=> => sha256:d412648e9c095ccb19a82b5ae10f5f11ca4295fe27701f789541896d80b184196 26.89MB / 26.89MB
=> => extracting sha256:ffb694ff4ff9e7c61d97c2b409f3e8154e2621a5074a087d35f1849e665d0d34 3.7s
=> => sha256:30d5234592d7e87754fa9d5676d9bb39ebc95ce99d66ef7d9736b9970745ce8 175.97MB / 175.97MB
=> => extracting sha256:2f746edec7f5a90ff637067d7b7635196acdff5dd52be3b1a1a3e2071cdaa4fed2 0.4s
=> => extracting sha256:0f22b8fd766b3ca3172fe1cefa088b3fb9a9ed626e93c99b1430fb5c56ddd 1.2s
=> => extracting sha256:6290ed4804fbcdd85913bc0d83a540b7d61928581d419c7a95dcf526323c220abf 0.0s
=> => extracting sha256:749d2b7f7347f97fe2073eabb632e8270d24925585b9fedea015c960a5e113f67f 0.8s
=> => sha256:ead36000791c346ee7ec129cab27c371774a8e3a25d4f4589652f464fcf04198ac 4.51KB / 4.51KB
=> => extracting sha256:733eafab37f72a5567b7f29841b9e2c46c665b2eccf01573592a2aca60a9 3.6s
=> => sha256:97a57dd5b8e02a54d50d98bebfb5b9d69810345fc1c2a5df168d2ca76cd6aeca 990B / 990B
=> => sha256:d8ab7119028eaa1b1b26c7524ed7a3e202e5e531f0d355fc89e6a512e03668 6.08KB / 6.08KB
=> => sha256:231580b9293c6b2368634ad533ada1c9bce95f3de99875a38f0e7ac6f3858866 535B / 535B
=> => extracting sha256:da5409c-f2d9e81048f151ad62857c51c7f73fd0869cf29953ad7f66eaaf84c3 0.0s
=> => sha256:1d66cb1ab6e1f91d40f61867bfb9d61d15de89e3388303d132e3b454b119f2 3.31KB / 3.31KB
=> => sha256:286f1e773be0474d3f6308780af1b3403ee060239f661cdaa4811307ed623371 1.73B / 1.73B
=> => extracting sha256:412618a9e9095cb19a82b5ae10f5f11ca4295fe27701f789541896d80b184196 1.1s
=> => extracting sha256:30d5234592d7e87754fa9d5676d9bb39ebc95ce99d66ef7d9736b9970745ce8 21.6s
=> => extracting sha256:ead36000791c346ee7ec129cab27c371774a8e3a25d4f4589652f464fcf04198ac 0.0s
=> => extracting sha256:97a57dd5b8e02a54d50d98bebfb5b9d69810345fc1c2a5df168d2ca76cd6aeca 0.0s
=> => extracting sha256:d8ab7119028eaa1b1b26c7524ed7a3e202e5e531fd0a5355cf89e6a542e03660
```