APACHE AIRFLOW – CODING ASSESMENT

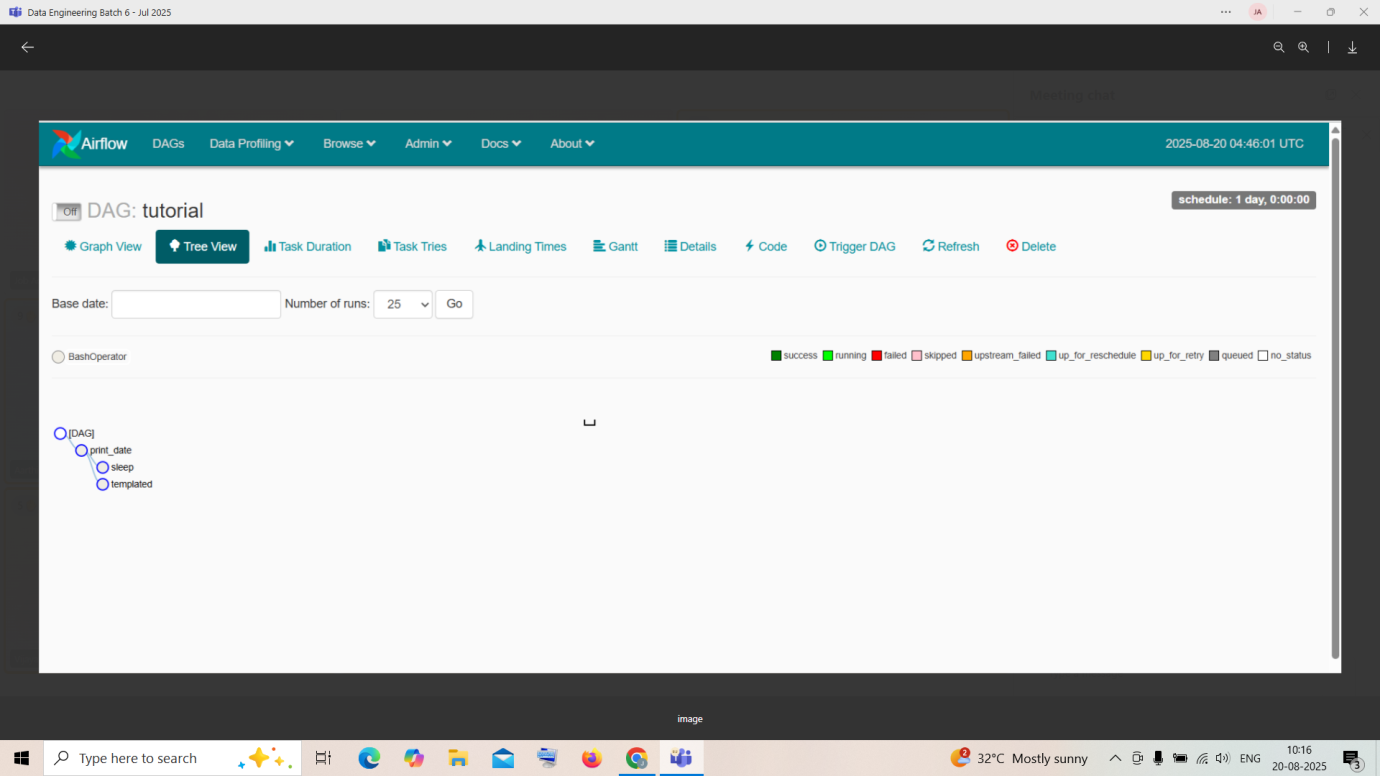
[Jobaoushadancse2021@jerusalemengg.ac.in](mailto:Jobaoushadancse2021@jerusalemengg.ac.in)

Job Aoushadan N – 20.08.2025

Start by installing Docker and Docker Compose (v2.14.0 or newer recommended), then download the official docker-compose.yaml from the Airflow documentation—this file orchestrates all necessary services like the scheduler, webserver, API server, database, and optional components such as Flower [Apache Airflow](https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html?utm_source=chatgpt.com)[Dataquest](https://www.dataquest.io/blog/setting-up-apache-airflow-with-docker-locally-part-i/?utm_source=chatgpt.com). In your project folder, make directories like dags, logs, plugins, (and config if needed), and to avoid permission issues on Linux, create a .env with AIRFLOW\_UID=$(id -u) (GID = 0) [Apache Airflow](https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html?utm_source=chatgpt.com)[Dataquest](https://www.dataquest.io/blog/setting-up-apache-airflow-with-docker-locally-part-i/?utm_source=chatgpt.com). Initialize Airflow’s metadata DB and default admin account by running docker compose up airflow-init, which typically generates credentials like airflow/airflow [Apache Airflow+1](https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html?utm_source=chatgpt.com). Finally, launch everything with docker compose up (optionally using -d for detached mode), then access the Airflow UI at http://localhost:8080 to author, schedule, and monitor your DAGs

Username – admin

Password - admin



First, make sure you have installed Docker Desktop and Visual Studio. If not, lets do it now!

Get Docker:

Get Visual Studio Code

Download the following file:

Open Visual Studio Code

create a new file .env and add the following lines

AIRFLOW\_IMAGE\_NAME=apache/airflow:2.4.2

AIRFLOW\_UID=50000

docker-compose up -d

create Admin user using below command:

docker-compose run airflow-worker airflow users create --role Admin --username admin --email admin --firstname admin --lastname admin --password admin

-----------------------------------------------------------------------------------------------------------------

Apache Airflow allows you to schedule, monitor, and manage workflows with ease, and having it on your Windows PC can be a game-changer for your data engineering and automation projects.

DAG:

To run our pipeline, we need a working Airflow environment. Docker Compose makes this easy and safe – no system-wide installs required. Just open your terminal and run the following:

*# Download the docker-compose.yaml file*

curl -LfO 'https://airflow.apache.org/docs/apache-airflow/stable/docker-compose.yaml'

*# Make expected directories and set an expected environment variable*

mkdir -p ./dags ./logs ./plugins

echo -e "AIRFLOW\_UID=**$(**id -u**)**" > .env

*# Initialize the database*

docker compose up airflow-init

*# Start up all services*

docker compose up

Once Airflow is up and running, visit the UI at http://localhost:8080.

Log in with:

* **Username:** airflow
* **Password:** airflow

You’ll land in the Airflow dashboard, where you can trigger DAGs, explore logs, and manage your environment.

## Create a Postgres Connection

Before our pipeline can write to Postgres, we need to tell Airflow how to connect to it. In the UI, open the **Admin > Connections** page and click the + button to add a new [connection](https://airflow.apache.org/docs/apache-airflow/stable/concepts/connections.html).

Fill in the following details:

* Connection ID: tutorial\_pg\_conn
* Connection Type: postgres
* Host: postgres
* Database: airflow (this is the default database in our container)
* Login: airflow
* Password: airflow
* Port: 5432

Save the connection. This tells Airflow how to reach the Postgres database running in your Docker environment.

Next, we’ll start building the pipeline that uses this connection.

## Create tables for staging and final data

Let’s begin with table creation. We’ll create two tables:

* employees\_temp: a staging table used for raw data
* employees: the cleaned and deduplicated destination

We’ll use the SQLExecuteQueryOperator to run the SQL statements needed to create these tables.

**from** **airflow.providers.common.sql.operators.sql** **import** SQLExecuteQueryOperator

create\_employees\_table = SQLExecuteQueryOperator(

task\_id="create\_employees\_table",

conn\_id="tutorial\_pg\_conn",

sql="""

CREATE TABLE IF NOT EXISTS employees (

"Serial Number" NUMERIC PRIMARY KEY,

"Company Name" TEXT,

"Employee Markme" TEXT,

"Description" TEXT,

"Leave" INTEGER

);""",

)

create\_employees\_temp\_table = SQLExecuteQueryOperator(

task\_id="create\_employees\_temp\_table",

conn\_id="tutorial\_pg\_conn",

sql="""

DROP TABLE IF EXISTS employees\_temp;

CREATE TABLE employees\_temp (

"Serial Number" NUMERIC PRIMARY KEY,

"Company Name" TEXT,

"Employee Markme" TEXT,

"Description" TEXT,

"Leave" INTEGER

);""",

)

You can optionally place these SQL statements in .sql files inside your dags/ folder and pass the file path to the sql= argument. This can be a great way to keep your DAG code clean.

## Load data into the staging table

Next, we’ll download a CSV file, save it locally, and load it into employees\_temp using the PostgresHook.

**import** **os**

**import** **requests**

**from** **airflow.sdk** **import** task

**from** **airflow.providers.postgres.hooks.postgres** **import** PostgresHook

@task

**def** get\_data():

*# NOTE: configure this as appropriate for your airflow environment*

data\_path = "/opt/airflow/dags/files/employees.csv"

os.makedirs(os.path.dirname(data\_path), exist\_ok=True)

url = "https://raw.githubusercontent.com/apache/airflow/main/airflow-core/docs/tutorial/pipeline\_example.csv"

response = requests.request("GET", url)

**with** open(data\_path, "w") **as** file:

file.write(response.text)

postgres\_hook = PostgresHook(postgres\_conn\_id="tutorial\_pg\_conn")

conn = postgres\_hook.get\_conn()

cur = conn.cursor()

**with** open(data\_path, "r") **as** file:

cur.copy\_expert(

"COPY employees\_temp FROM STDIN WITH CSV HEADER DELIMITER AS ',' QUOTE '**\"**'",

file,

)

conn.commit()

This task gives you a taste of combining Airflow with native Python and SQL hooks – a common pattern in real-world pipelines.

## Merge and clean the data

Now let’s deduplicate the data and merge it into our final table. We’ll write a task that runs a SQL INSERT … ON CONFLICT DO UPDATE.

**from** **airflow.sdk** **import** task

**from** **airflow.providers.postgres.hooks.postgres** **import** PostgresHook

@task

**def** merge\_data():

query = """

INSERT INTO employees

SELECT \*

FROM (

SELECT DISTINCT \*

FROM employees\_temp

) t

ON CONFLICT ("Serial Number") DO UPDATE

SET

"Employee Markme" = excluded."Employee Markme",

"Description" = excluded."Description",

"Leave" = excluded."Leave";

"""

**try**:

postgres\_hook = PostgresHook(postgres\_conn\_id="tutorial\_pg\_conn")

conn = postgres\_hook.get\_conn()

cur = conn.cursor()

cur.execute(query)

conn.commit()

**return** 0

**except** **Exception** **as** e:

**return** 1

## Defining the DAG

Now that we’ve defined all our tasks, it’s time to put them together into a DAG.

**import** **datetime**

**import** **pendulum**

**import** **os**

**import** **requests**

**from** **airflow.sdk** **import** dag, task

**from** **airflow.providers.postgres.hooks.postgres** **import** PostgresHook

**from** **airflow.providers.common.sql.operators.sql** **import** SQLExecuteQueryOperator

@dag(

dag\_id="process\_employees",

schedule="0 0 \* \* \*",

start\_date=pendulum.datetime(2021, 1, 1, tz="UTC"),

catchup=False,

dagrun\_timeout=datetime.timedelta(minutes=60),

)

**def** ProcessEmployees():

create\_employees\_table = SQLExecuteQueryOperator(

task\_id="create\_employees\_table",

conn\_id="tutorial\_pg\_conn",

sql="""

CREATE TABLE IF NOT EXISTS employees (

"Serial Number" NUMERIC PRIMARY KEY,

"Company Name" TEXT,

"Employee Markme" TEXT,

"Description" TEXT,

"Leave" INTEGER

);""",

)

create\_employees\_temp\_table = SQLExecuteQueryOperator(

task\_id="create\_employees\_temp\_table",

conn\_id="tutorial\_pg\_conn",

sql="""

DROP TABLE IF EXISTS employees\_temp;

CREATE TABLE employees\_temp (

"Serial Number" NUMERIC PRIMARY KEY,

"Company Name" TEXT,

"Employee Markme" TEXT,

"Description" TEXT,

"Leave" INTEGER

);""",

)

@task

**def** get\_data():

*# NOTE: configure this as appropriate for your airflow environment*

data\_path = "/opt/airflow/dags/files/employees.csv"

os.makedirs(os.path.dirname(data\_path), exist\_ok=True)

url = "https://raw.githubusercontent.com/apache/airflow/main/airflow-core/docs/tutorial/pipeline\_example.csv"

response = requests.request("GET", url)

**with** open(data\_path, "w") **as** file:

file.write(response.text)

postgres\_hook = PostgresHook(postgres\_conn\_id="tutorial\_pg\_conn")

conn = postgres\_hook.get\_conn()

cur = conn.cursor()

**with** open(data\_path, "r") **as** file:

cur.copy\_expert(

"COPY employees\_temp FROM STDIN WITH CSV HEADER DELIMITER AS ',' QUOTE '**\"**'",

file,

)

conn.commit()

@task

**def** merge\_data():

query = """

INSERT INTO employees

SELECT \*

FROM (

SELECT DISTINCT \*

FROM employees\_temp

) t

ON CONFLICT ("Serial Number") DO UPDATE

SET

"Employee Markme" = excluded."Employee Markme",

"Description" = excluded."Description",

"Leave" = excluded."Leave";

"""

**try**:

postgres\_hook = PostgresHook(postgres\_conn\_id="tutorial\_pg\_conn")

conn = postgres\_hook.get\_conn()

cur = conn.cursor()

cur.execute(query)

conn.commit()

**return** 0

**except** **Exception** **as** e:

**return** 1

[create\_employees\_table, create\_employees\_temp\_table] >> get\_data() >> merge\_data()

dag = ProcessEmployees()