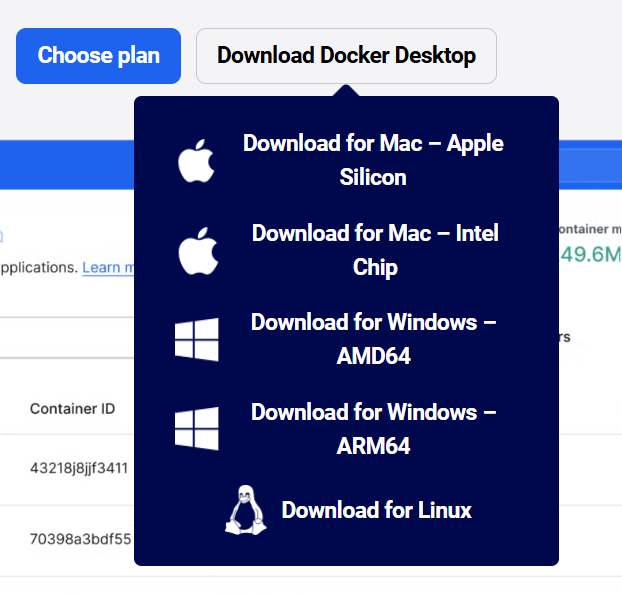
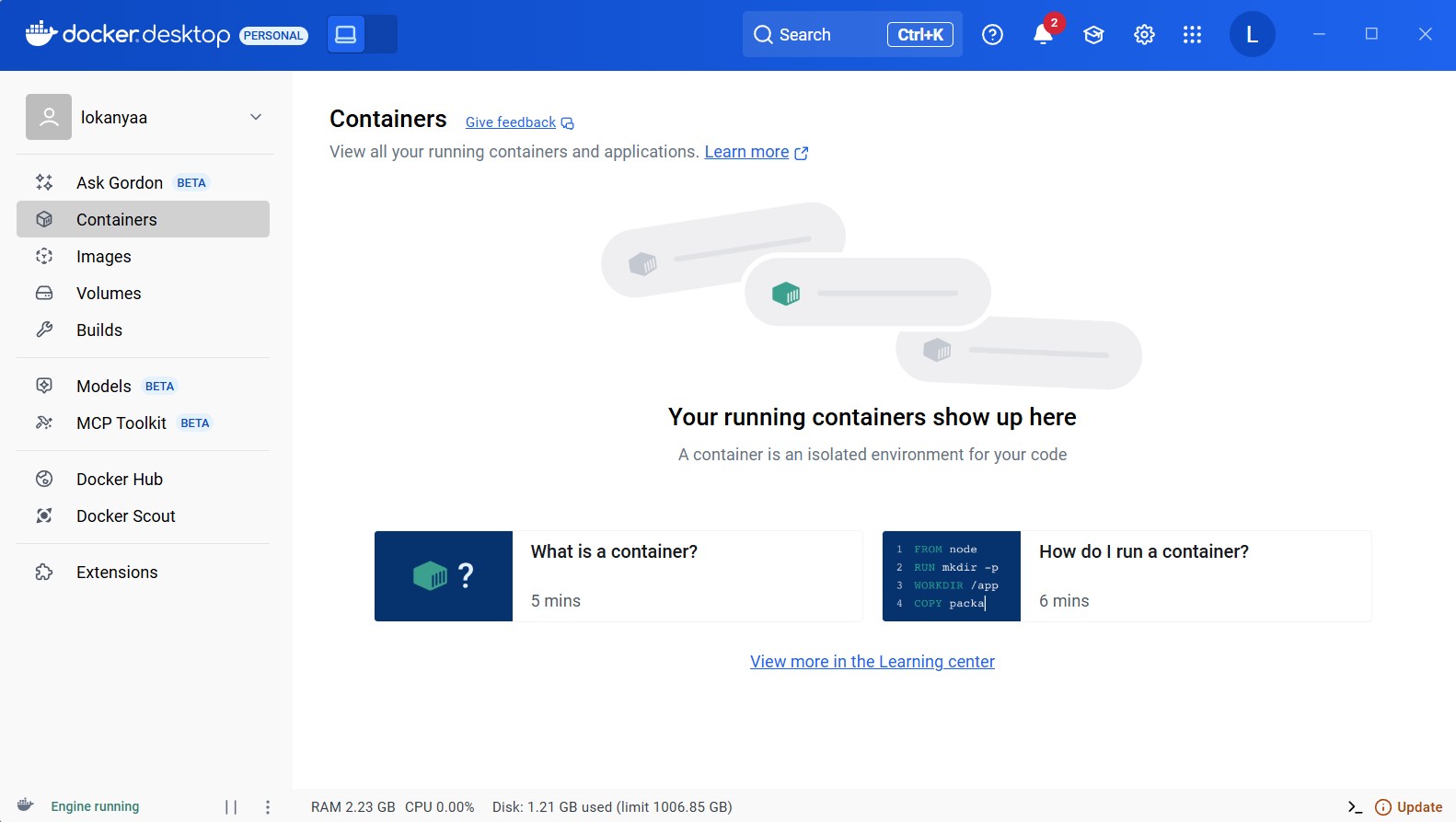
**Apache Airflow – Assignment - Lokanya G**

**Step 1**: Install docker exe from google, choose Download for Windows-AMD64



**Step 2**: Double click on the exe file you downloaded and after the process gets completed restart you system.



**Step 3**: Open this location “C:\Users\user name\materials” and paste the ‘docker-compose.yaml’ file in it.

**Step 4**: Now right click on the file and open with vs code(open a folder), in the vs code folder create ‘.env’ file and paste the below command .

AIRFLOW\_IMAGE\_NAME=apache/airflow:2.4.2

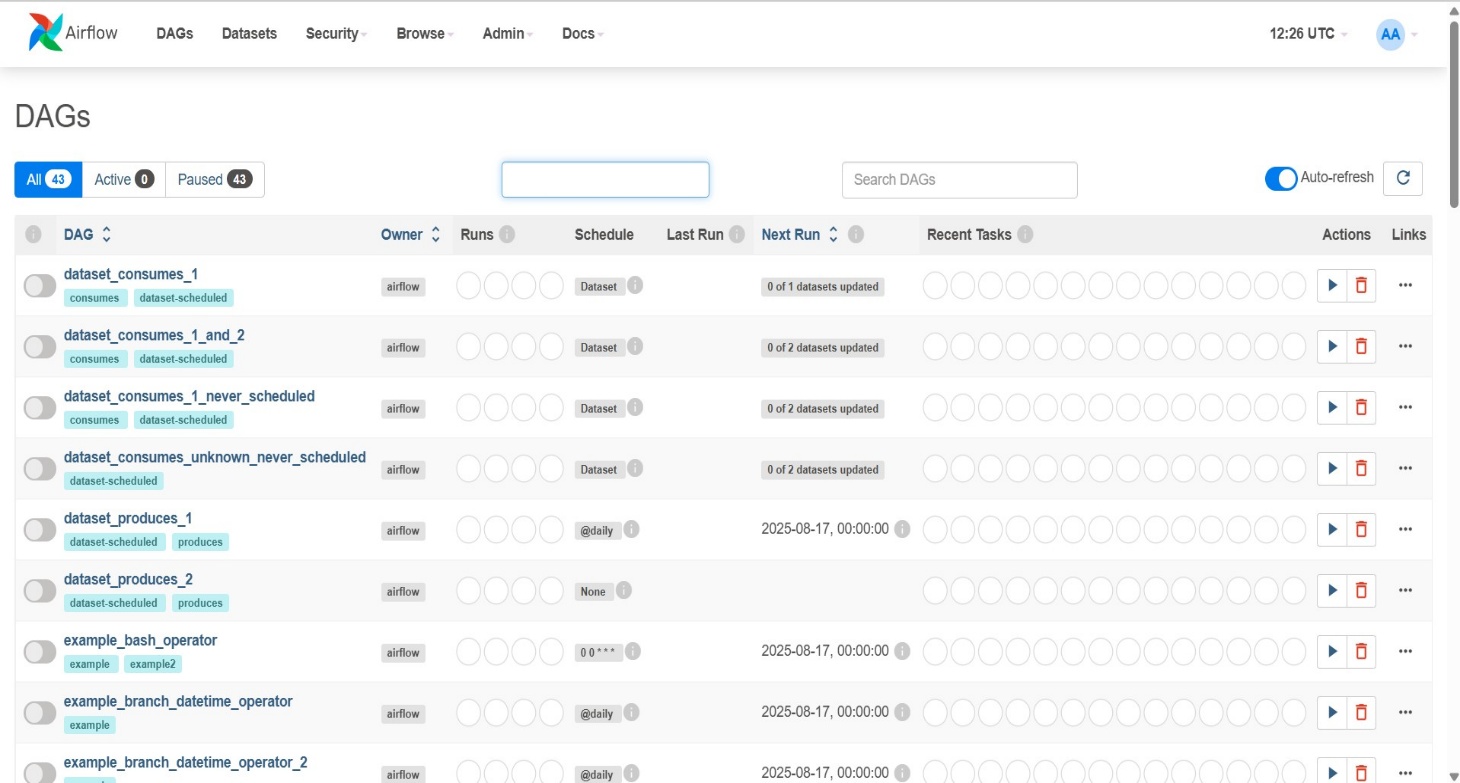
AIRFLOW\_UID=50000

**Step 5**: Again in vs code on the same location open a new terminal and run the below command.

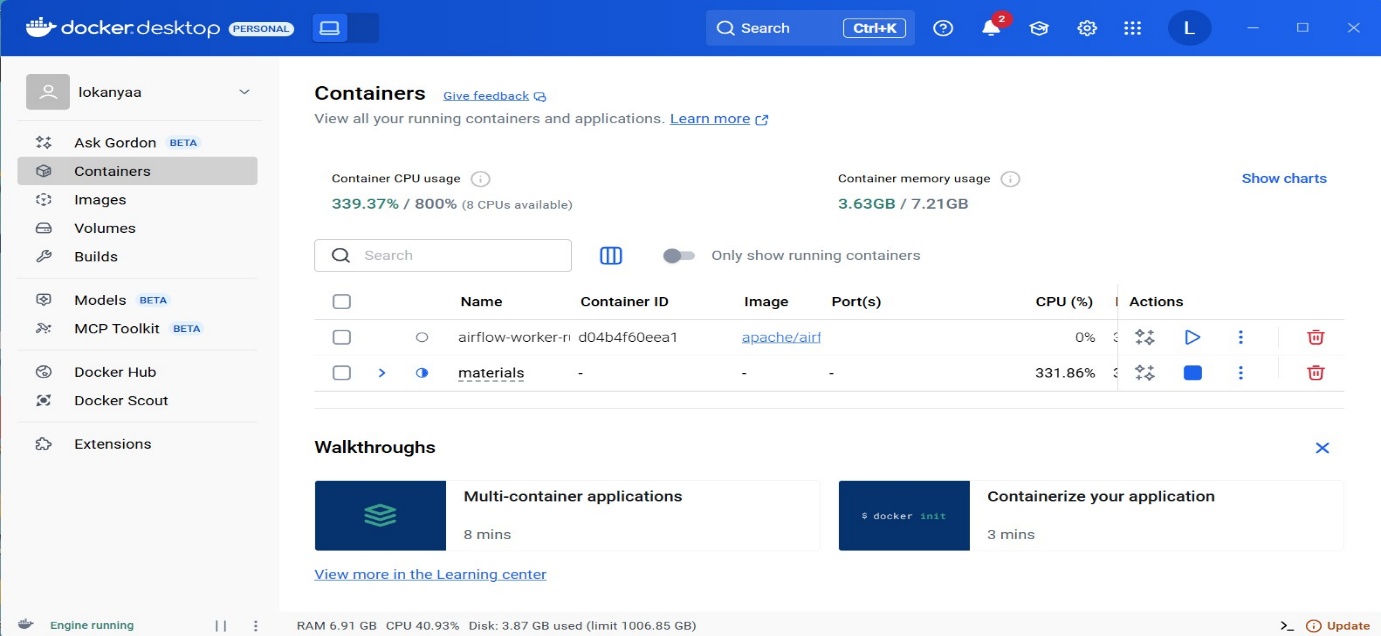
docker-compose up -d

**Step 6**: Open chrome and type ‘localhost:8080’ ,Docker UI window will open then you have to enter the password and username and login .’

**Step 7**: Now the final Docker workpage gets opened.



Docker page after setup done in chrome.



**Building a Simple Data Pipeline**

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.

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.

You have to save the code in the ‘materials’ location under dags and name it as process\_employees.py.

**CODE:**

from airflow import DAG

from airflow.providers.sqlite.operators.sqlite import SqliteOperator

from datetime import datetime

with DAG(

dag\_id="process\_employees\_dag",

start\_date=datetime(2023, 1, 1),

schedule\_interval=None,

catchup=False,

) as dag:

create\_employees\_table = SqliteOperator(

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

sqlite\_conn\_id="my\_sqlite\_conn",

sql="""

CREATE TABLE IF NOT EXISTS employees (

id INTEGER PRIMARY KEY,

name TEXT,

department TEXT

)

""",

)

create\_employees\_temp\_table = SqliteOperator(

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

sqlite\_conn\_id="my\_sqlite\_conn",

sql="""

CREATE TABLE IF NOT EXISTS employees\_temp (

id INTEGER,

name TEXT,

department TEXT

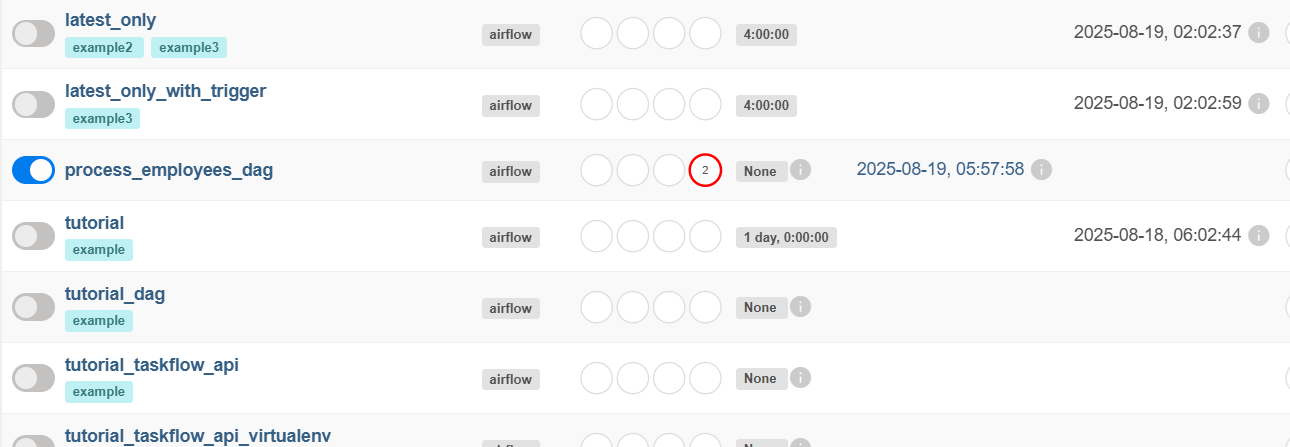
)

""",

)

create\_employees\_table >> create\_employees\_temp\_table

After this step you will see the process\_employees dag name in the DAGs



**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.

**Then modify the code in process\_employees.py like below for Loading data:**

from datetime import datetime

from airflow import DAG

from airflow.providers.postgres.operators.postgres import PostgresOperator

from airflow.decorators import task

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

import os

import requests

# Task to download CSV and load into Postgres

@task

def get\_data():

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

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

    # Download CSV

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

    response = requests.get(url)

    with open(data\_path, "w", encoding="utf-8") as file:

        file.write(response.text)

    # Load into Postgres

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

    conn = postgres\_hook.get\_conn()

    cur = conn.cursor()

    with open(data\_path, "r", encoding="utf-8") as file:

        cur.copy\_expert("""

            COPY employees\_temp FROM STDIN WITH CSV HEADER

            DELIMITER ',' QUOTE '"'

        """, file)

    conn.commit()

    cur.close()

    conn.close()

with DAG(

    dag\_id="process\_employees\_dag",

    start\_date=datetime(2023, 1, 1),

    schedule\_interval=None,

    catchup=False,

) as dag:

    # Create employees main table

    create\_employees\_table = PostgresOperator(

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

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

        sql="""

        CREATE TABLE IF NOT EXISTS employees (

            id INTEGER PRIMARY KEY,

            name TEXT,

            department TEXT

        )

        """,

    )

    # Create employees temp table

    create\_employees\_temp\_table = PostgresOperator(

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

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

        sql="""

        CREATE TABLE IF NOT EXISTS employees\_temp (

            id INTEGER,

            name TEXT,

            department TEXT

        )

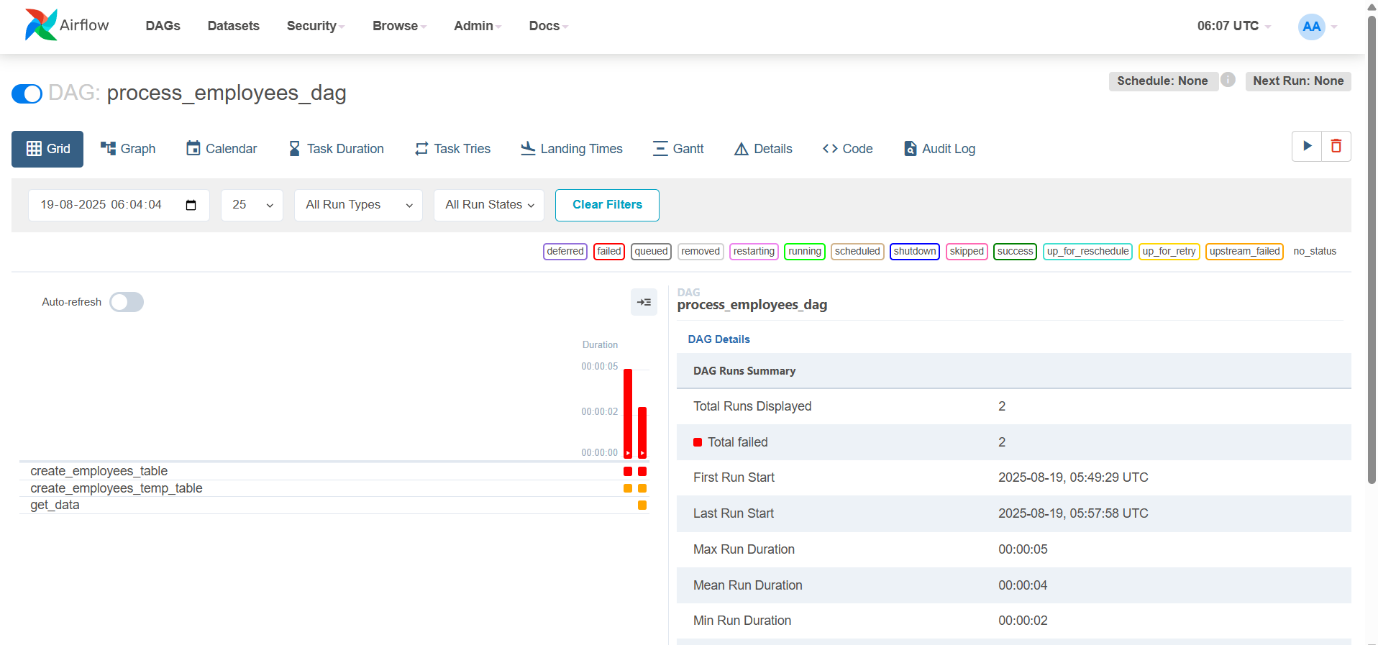
        """,

    )

    # Run tasks in sequence

    create\_employees\_table >> create\_employees\_temp\_table >> get\_data()

Then trigger the dag again to see the updation done via code .



Now create an employees.csv file in the location ~/materials/dags/data/employees.csv

**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*.

Modify the code in process\_employees.py file like below:

from datetime import datetime

from airflow import DAG

from airflow.providers.postgres.operators.postgres import PostgresOperator

from airflow.decorators import task

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

import os

# Task to load CSV into Postgres

@task

def get\_data():

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

    if not os.path.exists(data\_path):

        raise FileNotFoundError(f"{data\_path} not found. Please place employees.csv there.")

    # Load into Postgres

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

    conn = postgres\_hook.get\_conn()

    cur = conn.cursor()

    with open(data\_path, "r", encoding="utf-8") as file:

        cur.copy\_expert("""

            COPY employees\_temp FROM STDIN WITH CSV HEADER

            DELIMITER ',' QUOTE '"'

        """, file)

    conn.commit()

    cur.close()

    conn.close()

with DAG(

    dag\_id="process\_employees\_dag",

    start\_date=datetime(2023, 1, 1),

    schedule\_interval=None,

    catchup=False,

) as dag:

    # Create employees main table

    create\_employees\_table = PostgresOperator(

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

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

        sql="""

        CREATE TABLE IF NOT EXISTS employees (

            id INTEGER PRIMARY KEY,

            name TEXT,

            department TEXT

        )

        """,

    )

    # Create employees temp table

    create\_employees\_temp\_table = PostgresOperator(

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

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

        sql="""

        CREATE TABLE IF NOT EXISTS employees\_temp (

            id INTEGER,

            name TEXT,

            department TEXT

        )

        """,

    )

    # Merge data from temp into main table

    merge\_data = PostgresOperator(

        task\_id="merge\_data",

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

        sql="""

        INSERT INTO employees (id, name, department)

        SELECT DISTINCT id, name, department

        FROM employees\_temp

        ON CONFLICT (id) DO UPDATE

        SET

            name = EXCLUDED.name,

            department = EXCLUDED.department;

        """,

    )

    # Run tasks in sequence

    create\_employees\_table >> create\_employees\_temp\_table >> get\_data() >> merge\_data

