


# Apache Airflow Assignment: Building a Simple Data Pipeline

By ESAQ A

## 1 . Open your Airflow UI


- Open this website in browser <http://localhost:8080>
- Enter your username and password

03:23 IST (+05:30) → Log In


Sign In

Enter your login and password below:

Username:


 airflow

Password:

 .....

Sign In

- You will land on the Airflow Dashboard


DAGsCluster ActivityDatasetsSecurityBrowseAdminDocs03:25 IST (+05:30) AA







## DAGs

All 54Active 2Paused 52Running 0Failed 0

Filter DAGs by tag

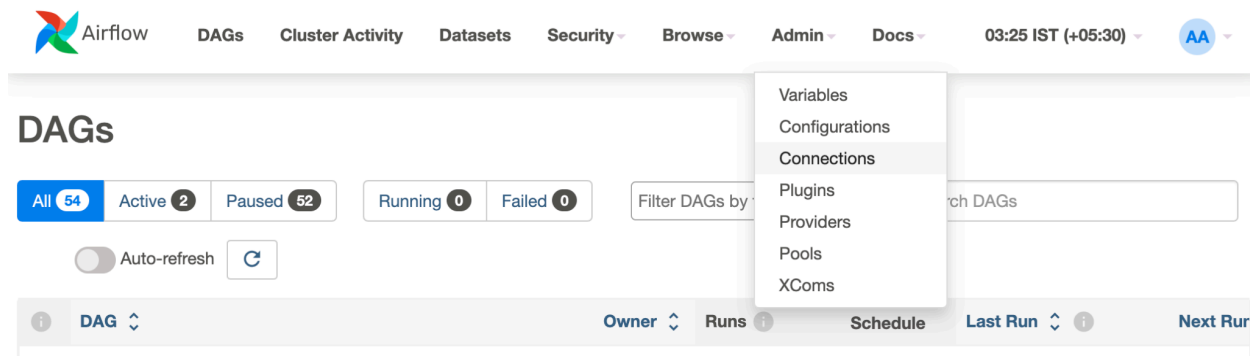
Search DAGs

☐ Auto-refresh 

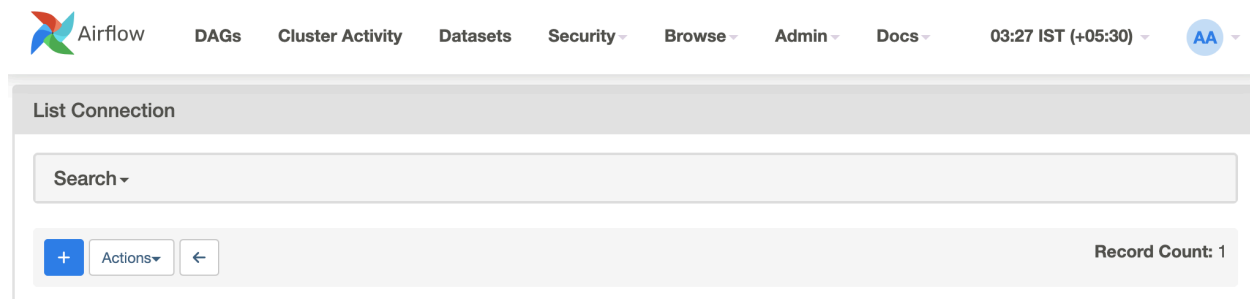
DAG	Owner	Runs	Schedule	Last Run	Next Run
 dataset_consumes_1 consumes dataset-scheduled	airflow		Dataset	2025-08-18, 17:11:53	On s3://da
 dataset_consumes_1_and_2 consumes dataset-scheduled	airflow		Dataset		0 of 2 data
 dataset_consumes_1_never_scheduled consumes dataset-scheduled	airflow		Dataset		0 of 2 data

## 2 . Creating Postgres Connection:

- In the navigation bar, select Admin and Connections.



- Click on the + icon and add a new connection.



- Fill these details in the new connection and save the new connection..

- 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**

## 3 . Installing required Python packages

- Open terminal, in VS Code or default terminal
- Install the “apache-airflow” package using `pip install apache-airflow`

```
iamesaq@iamesaqsmac airflow % pip3 install apache-airflow
Collecting apache-airflow
  Using cached apache_airflow-3.0.4-py3-none-any.whl.metadata (32 kB)
Collecting apache-airflow-core==3.0.4 (from apache-airflow)
  Using cached apache_airflow_core-3.0.4-py3-none-any.whl.metadata (7.4 kB)
Collecting apache-airflow-task-sdk<1.1.0,>=1.0.4 (from apache-airflow)
  Using cached apache_airflow_task_sdk-1.0.4-py3-none-any.whl.metadata (3.8 kB)
Collecting a2wsgi>=1.10.8 (from apache-airflow-core==3.0.4->apache-airflow)
  Using cached a2wsgi-1.10.10-py3-none-any.whl.metadata (4.0 kB)
Collecting aiosqlite>=0.20.0 (from apache-airflow-core==3.0.4->apache-airflow)
  Using cached aiosqlite-0.21.0-py3-none-any.whl.metadata (4.3 kB)
Collecting alembic<2.0,>=1.13.1 (from apache-airflow-core==3.0.4->apache-airflow)
  Using cached alembic-1.16.4-py3-none-any.whl.metadata (7.3 kB)
Collecting apache-airflow-providers-common-compat>=1.6.0 (from apache-airflow-core==3.0.4->apache-airflow)
  Using cached apache_airflow_providers_common_compat-1.7.3-py3-none-any.whl.metadata (5.3 kB)
Collecting apache-airflow-providers-common-io>=1.5.3 (from apache-airflow-core==3.0.4->apache-airflow)
  Using cached apache_airflow_providers_common_io-1.6.2-py3-none-any.whl.metadata (5.3 kB)
Collecting apache-airflow-providers-common-sql>=1.26.0 (from apache-airflow-core==3.0.4->apache-airflow)
```

## 4. Defining the DAG:

- Now navigate to the folder where docker-console.yaml file is present and create a “dags” folder
- create a python file named “process\_employees.py” in this location “../airflow/dags/files/” with the following code.

process\_employees.py

```
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()

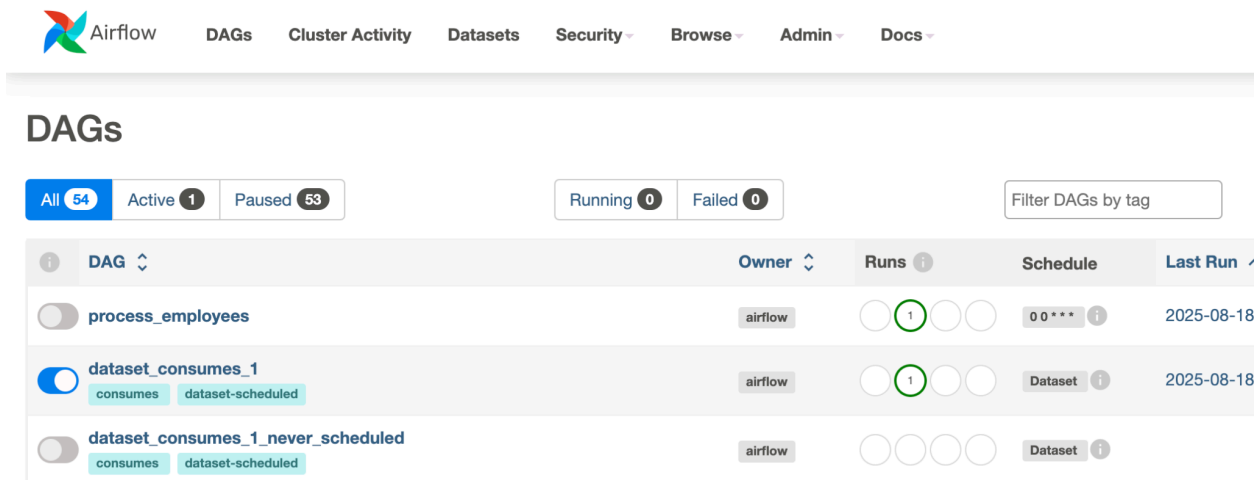
dag = ProcessEmployees()

```

- Once the file is saved in the above said directory, we will be able to see our DAG on the Airflow UI.

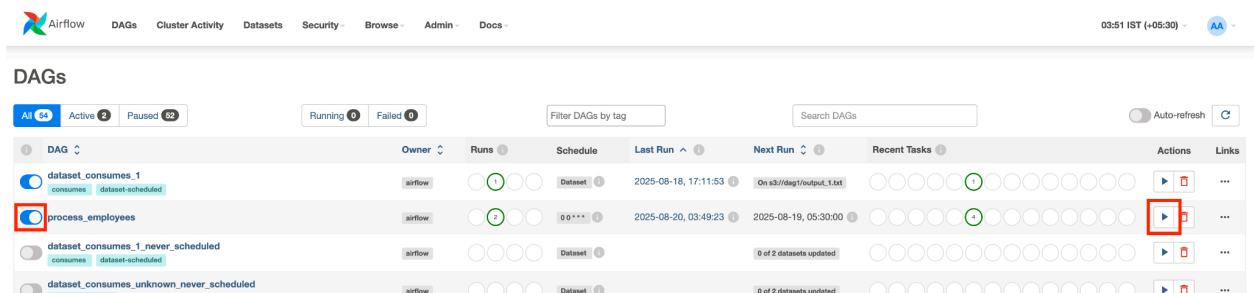
## 4 . Trigger and Explore your DAG

- Now open the Airflow UI, and look for your DAG's name.



The screenshot shows the Airflow DAGs list page. The top navigation bar includes links for DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs. Below the navigation bar, there are filters for DAG status: All (54), Active (1), Paused (53), Running (0), and Failed (0). A search bar for filtering DAGs by tag is also present. The main table lists DAGs with columns for DAG name, Owner, Runs, Schedule, and Last Run. The 'process\_employees' DAG is shown with its slider turned off. The 'dataset\_consumes\_1' DAG is shown with its slider turned on and a green circle around the '1' in the 'Runs' column.

- Toggle the slider near your DAG “ON” and click on the play button to trigger a run.



The screenshot shows the Airflow DAGs list page. The top navigation bar includes links for DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs. Below the navigation bar, there are filters for DAG status: All (54), Active (2), Paused (62), Running (0), and Failed (0). A search bar for filtering DAGs by tag is also present. The main table lists DAGs with columns for DAG name, Owner, Runs, Schedule, Last Run, Next Run, Recent Tasks, Actions, and Links. The 'process\_employees' DAG is shown with its slider turned on. The 'dataset\_consumes\_1' DAG is shown with its slider turned on and a green circle around the '1' in the 'Runs' column. The 'process\_employees' DAG is highlighted with a red box, and the play button in the 'Actions' column is also highlighted with a red box.

- Then we can view each task in GRID and explore by viewing the logs for each step.

The image displays the Apache Airflow web interface for a DAG named 'process\_employees'. The top navigation bar includes links for DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs. The interface is divided into several sections:

- Grid View:** A table showing task instances with columns for Duration, Task Name, and Status. The tasks listed are 'create\_employees\_table', 'create\_employees\_temp\_table', 'get\_data', and 'merge\_data'. All tasks show a duration of 00:00:09 and a status of 'success'.
- DAG Runs Summary:** A table providing summary statistics for the DAG runs.

DAG Runs Summary	
Total Runs Displayed	2
Total success	2
First Run Start	2025-08-19, 12:10:16 IST
Last Run Start	2025-08-20, 03:49:24 IST
Max Run Duration	00:00:09
Mean Run Duration	00:00:08
Min Run Duration	00:00:07
DAG Summary	
Total Tasks	4
- DAG Graph:** A visual representation of the DAG workflow. It shows four tasks: 'create\_employees\_table' (SQL ExecuteQueryOperator), 'create\_employees\_temp\_table' (SQL ExecuteQueryOperator), 'get\_data' (task), and 'merge\_data' (task). The tasks are connected in a sequence, with 'create\_employees\_table' and 'create\_employees\_temp\_table' both leading to 'get\_data', which then leads to 'merge\_data'.

The interface also includes a search bar, filters for run types and states, and a 'Clear Filters' button. The bottom right corner shows the current time as 03:55 IST (+05:30) and the next run scheduled for 2025-08-19, 05:30:00.