

## **CODING CHALLENGE (Apache Airflow)**

**TRAINEE NAME: Swathi Baskaran**

Apache Airflow is an open-source workflow orchestration tool. It helps us to define, schedule and monitor workflows as code. Pipelines are defined as DAGs using Python code, tasks are created with operators and dependencies are set to define workflow execution. Using Airflow, we can automate repetitive tasks, orchestrate data movement, and manage dependencies between processes. This was first developed by Airbnb. It later became a part of Apache.

### **Features of Apache Airflow:**

1. **Dynamic Workflow Creation:** Pipelines are defined as Python code, making them dynamic, extensible, and easy to modify.
2. **DAGs (Directed Acyclic Graphs):** Workflows are represented as a graph of tasks with clear dependencies.
3. **Task Scheduling:** Workflows can be scheduled based on time, events or external triggers.
4. **Scalability:** Supports execution across distributed environments using Celery, Kubernetes or local Executors
5. **Monitoring and Logging:** Apache Airflow provides an extensive UI to monitor running tasks, view logs, and retry failed tasks.
6. **Extensibility:** Offers integrations with multiple data sources, cloud platforms, and third-party systems.
7. **Version Control:** Workflows can be managed under Git or any versioning system.
8. **Data Lineage:** Helps in tracking dependencies and flow of data between tasks.

### **Building a pipeline in Apache Airflow:**

1. Install and set up Airflow
  - Install Apache Airflow using pip or Docker.
  - Initialize the Airflow metadata database (docker compose up airflow-init)

- Start Airflow components: webserver, scheduler, and workers.  
(docker compose up)

```

.env
1 AIRFLOW_IMAGE_NAME=apache/airflow:2.4.2
2 AIRFLOW_UID=50000
3

PS C:\Users\swath\Materials> docker compose up airflow-init
time="2025-08-20T09:29:23+05:30" level=warning msg="C:\\Users\\swath\\Materials\\docker-compose.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 2/2
  ✓ Container materials-redis-1      Running      0.0s
  ✓ Container materials-postgres-1   Running      0.0s
Attaching to airflow-init-1
airflow-init-1 | WARNING!!!!: Not enough memory available for Docker.
airflow-init-1 | At least 4GB of memory required. You have 3.5G
airflow-init-1 |
airflow-init-1 | WARNING!!!!: You have not enough resources to run Airflow (see above)!
airflow-init-1 | Please follow the instructions to increase amount of resources available:
airflow-init-1 | https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html#before-you-begin
airflow-init-1 |

PS C:\Users\swath\Materials> docker compose up
time="2025-08-20T09:31:20+05:30" level=warning msg="C:\\Users\\swath\\Materials\\docker-compose.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 6/6
  ✓ Container materials-redis-1      Running      0.0s
  ✓ Container materials-postgres-1   Running      0.0s
  ✓ Container materials-airflow-webserver-1 Running      0.0s
  ✓ Container materials-airflow-scheduler-1 Running      0.0s
  ✓ Container materials-airflow-triggerer-1 Running      0.0s
  ✓ Container materials-airflow-worker-1 Running      0.0s
Attaching to airflow-init-1, airflow-scheduler-1, airflow-triggerer-1, airflow-webserver-1, airflow-worker-1, postgres-1, redis-1
airflow-worker-1 | BACKEND=redis
airflow-worker-1 | DB_HOST=redis
airflow-worker-1 | DB_PORT=6379
airflow-init-1 |
airflow-init-1 | WARNING!!!!: Not enough memory available for Docker.

```

## 2. Creating a connection with Apache UI

- In the Apache UI, a connection is created which mentions the port, username, password, database and other details that are required.

**Airflow** DAGs Datasets Security Browse Admin Docs 04:04 UTC

### Edit Connection

Connection Id \* tutorial\_pg\_conn

Connection Type \* Postgres  
Connection Type missing? Make sure you've installed the corresponding Airflow Provider Package.

Description

Host postgres

Schema

Login airflow

Password

Port 5432

---

**Airflow** DAGs Datasets Security Browse Admin Docs 04:05 UTC

### List Connection

Search

+ Actions -

Record Count: 1

	Conn Id	Conn Type	Description	Host	Port	Is Encrypted	Is Extra Encrypted
<input type="checkbox"/>	tutorial_pg_conn	postgres		postgres	5432	False	False

### 3. Define a DAG (Directed Acyclic Graph)

- Create a Python script inside the dags folder
- Define default arguments (owner, retries, retry\_delay, start\_date)
- Define the DAG with DAG() function and a schedule interval.

```
.env process_employees.py 4 X
dags > process_employees.py > ProcessEmployees
1 import datetime
2 import pendulum
3 import os
4 import requests
5
6 from airflow.decorators import dag, task
7 from airflow.providers.postgres.hooks.postgres import PostgresHook
8 from airflow.providers.postgres.operators.postgres import PostgresOperator
9
10
11
12 @dag(
13     dag_id="process_employees",
14     schedule="0 0 * * *", # daily at midnight
15     start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
16     catchup=False,
17     dagrun_timeout=datetime.timedelta(minutes=60),
18 )
19 def ProcessEmployees():
20     # Step 1: Create main employees table
21     create_employees_table = PostgresOperator(
22         task_id="create_employees_table",
23         postgres_conn_id="tutorial_pg_conn",
24         sql="""
25             CREATE TABLE IF NOT EXISTS employees (
26                 "Serial Number" NUMERIC PRIMARY KEY,
27                 "Company Name" TEXT,
28                 "Employee Markme" TEXT,
29                 "Description" TEXT,
30                 "Leave" INTEGER
31             );""",
32     )
33
34     # Step 2: Create staging table
35     create_employees_temp_table = PostgresOperator(
36         task_id="create_employees_temp_table",
37         postgres_conn_id="tutorial_pg_conn",
```

Ln 19, Col 24 Spaces: 4 UTF-8

#### 4. Define Tasks

- Use Operators such as PostGresOperator, BashOperator, PythonOperator to define tasks.
- The operations that we choose to perform are defined as tasks

```
.env load_data_dag.py 2 X
dags > load_data_dag.py > get_data
1 import os
2 import requests
3 from airflow.decorators import task
4 from airflow.providers.postgres.hooks.postgres import PostgresHook
5
6
7 @task
8 def get_data():
9     # NOTE: configure this as appropriate for your airflow environment
10    data_path = "/opt/airflow/dags/files/employees.csv"
11    os.makedirs(os.path.dirname(data_path), exist_ok=True)
12
13    url = "https://raw.githubusercontent.com/apache/airflow/main/airflow-core/docs/tutorial/pipeline_example.csv"
14
15    response = requests.request("GET", url)
16
17    with open(data_path, "w") as file:
18        file.write(response.text)
19
20    postgres_hook = PostgresHook(postgres_conn_id="tutorial_pg_conn")
21    conn = postgres_hook.get_conn()
22    cur = conn.cursor()
23    with open(data_path, "r") as file:
24        cur.copy_expert(
25            "COPY employees_temp FROM STDIN WITH CSV HEADER DELIMITER AS ',' QUOTE '\"'",
26            file,
27        )
28    conn.commit()
```

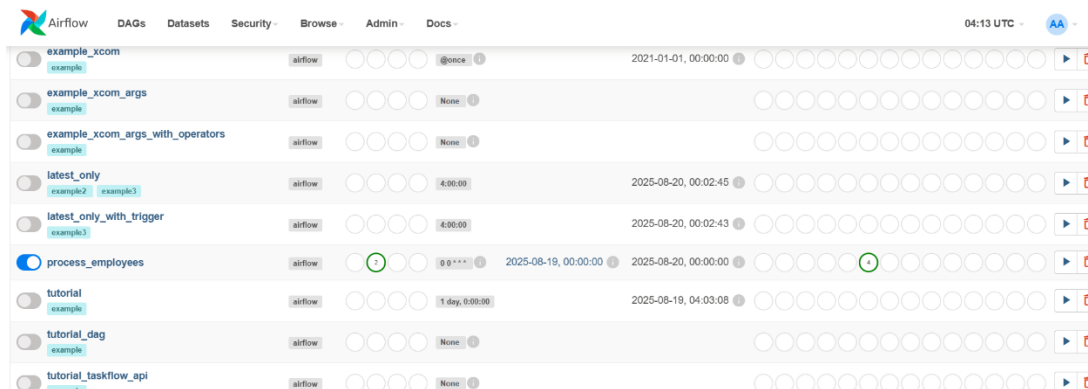
```
.env merge_task.py 2 X
dags > merge_task.py > merge_data
1 from airflow.decorators import task
2 from airflow.providers.postgres.hooks.postgres import PostgresHook
3
4
5 @task
6 def merge_data():
7     query = """
8         INSERT INTO employees
9         SELECT *
10        FROM (
11            SELECT DISTINCT *
12            FROM employees_temp
13        ) t
14        ON CONFLICT ("Serial Number") DO UPDATE
15        SET
16            "Employee Markme" = excluded."Employee Markme",
17            "Description" = excluded."Description",
18            "Leave" = excluded."Leave";
19    """
20    try:
21        postgres_hook = PostgresHook(postgres_conn_id="tutorial_pg_conn")
22        conn = postgres_hook.get_conn()
23        cur = conn.cursor()
24        cur.execute(query)
25        conn.commit()
26        return 0
27    except Exception as e:
28        return 1
```

## 5. Set Task Dependencies

- Use >> (Right shift) and << (Left shift) to set execution order.

## 6. Deploying the DAG

- The DAG file is placed in the dags directory
- Scheduler automatically picks it up
- The DAG is either triggered manually or we have to wait for its schedule



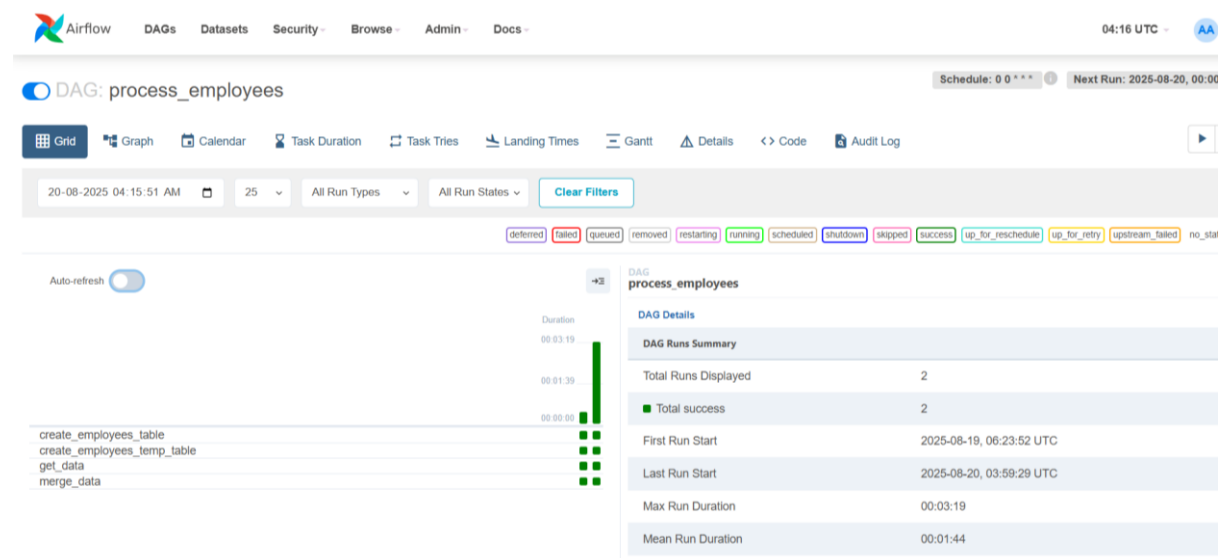
DAG ID	Owner	Next Run	Status
example_xcom	airflow	2021-01-01, 00:00:00	Failed
example_xcom_args	airflow	None	Failed
example_xcom_args_with_operators	airflow	None	Failed
latest_only	airflow	2025-08-20, 00:02:45	Failed
latest_only_with_trigger	airflow	2025-08-20, 00:02:43	Failed
process_employees	airflow	2025-08-19, 00:00:00	Running
tutorial	airflow	2025-08-19, 04:03:08	Failed
tutorial_dag	airflow	None	Failed
tutorial_taskflow_api	airflow	None	Failed

## 7. Monitor execution of DAG

- Open Airflow UI (<http://localhost:8080>)
- Check DAG runs, task statuses and logs.
- Retry failed tasks if needed.

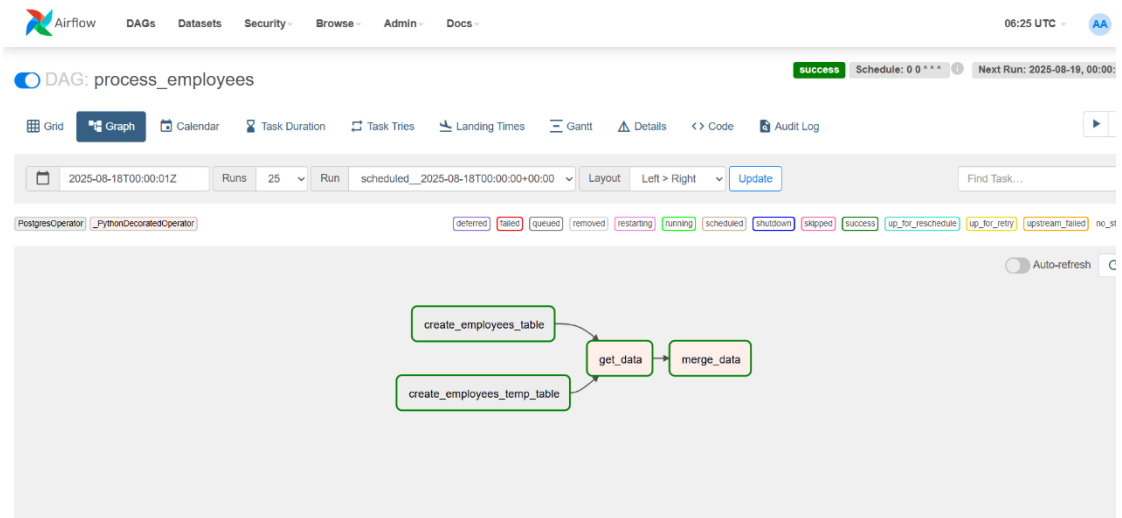
## Viewing Pipelines in Airflow:

- **Grid View:** Shows DAG runs in a tabular format with task statuses, making it easy to spot failures and successes over multiple runs.

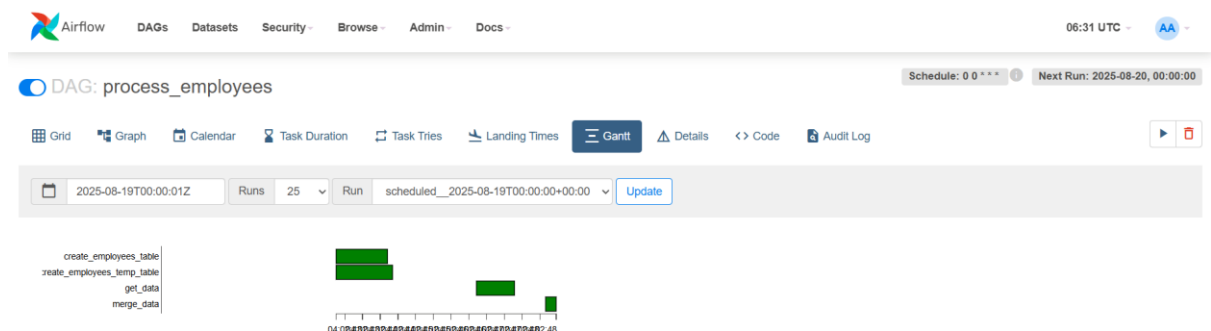


DAG: process_employees	
Schedule: 0 0 * * * * Next Run: 2025-08-20, 00:00	
Grid   Graph   Calendar   Task Duration   Task Times   Landing Times   Gantt   Details   Code   Audit Log	
20-08-2025 04:15:51 AM 25 All Run Types All Run States Clear Filters	
Auto-refresh	
Duration: 00:03:19	
create_employees_table	
create_employees_temp_table	
get_data	
merge_data	
DAG Details	
DAG Runs Summary	
Total Runs Displayed	2
Total success	2
First Run Start	2025-08-19, 06:23:52 UTC
Last Run Start	2025-08-20, 03:59:29 UTC
Max Run Duration	00:03:19
Mean Run Duration	00:01:44

- **Graph View:** It is a visual representation of the workflow DAG showing task dependencies.



- **Tree View:** Displays task statuses across multiple DAG runs.
- **Gantt Chart:** Shows task execution timeline for performance monitoring.



- **Code View:** Displays DAG's Python code directly in the UI.

The screenshot shows the Airflow web interface for the DAG 'process\_employees'. The top navigation bar includes links for Airflow, DAGs, Datasets, Security, Browse, Admin, and Docs. The current time is 06:32 UTC. The DAG's schedule is '0 0 \* \* \*' and the next run is on 2025-08-20 at 00:00. The interface has tabs for Grid, Graph, Calendar, Task Duration, Task Tries, Landing Times, Gantt, Details, Code, and Audit Log. The 'Code' tab is selected, displaying the Python code for the DAG. The code defines a DAG with a daily schedule at midnight, starting from 2021, and a task 'create\_employees\_table' using a 'PostgresOperator' to create a table in a PostgreSQL database. A 'Toggle Wrap' button is visible on the right side of the code editor.

```
1 import datetime
2 import pendulum
3 import os
4 import requests
5
6 from airflow.decorators import dag, task
7 from airflow.providers.postgres.hooks.postgres import PostgresHook
8 from airflow.providers.postgres.operators.postgres import PostgresOperator
9
10
11
12 @dag(
13     dag_id="process_employees",
14     schedule="0 0 * * *", # daily at midnight
15     start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
16     catchup=False,
17     dagrun_timeout=datetime.timedelta(minutes=60),
18 )
19 def ProcessEmployees():
20     # Step 1: Create main employees table
21     create_employees_table = PostgresOperator(
22         task_id="create_employees_table",
23         postgres_conn_id="tutorial_pg_conn",
24         sql="""
25         CREATE TABLE IF NOT EXISTS employees (
```

- **Task Instance Logs:** Each task has execution logs viewable in the UI.

The screenshot shows the Airflow web interface for the Task Instance 'create\_employees\_table' at 2025-08-19, 00:00:00. The top navigation bar is the same as the previous screenshot. The current time is 06:34 UTC. The DAG's schedule is '0 0 \* \* \*'. The interface has tabs for Grid, Graph, Calendar, Task Duration, Task Tries, Landing Times, Gantt, Details, Code, and Audit Log. The 'Details' tab is selected, showing the task instance details. Below the details, there are buttons for 'Task Instance Details', 'Rendered Template', 'Log', and 'XCom'. The 'Log' button is selected, displaying the execution logs. The logs show the task instance starting at 2025-08-20, 04:02:42 UTC, with dependencies met, and the task running successfully on 2025-08-20, 04:02:43 UTC. The logs are displayed in a scrollable area with a 'Jump To End' button, a 'Toggle Wrap' button, and a 'Download' button.

Task Instance: create\_employees\_table at 2025-08-19, 00:00:00

Task Instance Details <> Rendered Template Log XCom

Log by attempts

1

Jump To End Toggle Wrap Download

```
*** Reading local file: /opt/airflow/logs/dag_id=process_employees/run_id=scheduled__2025-08-19T00:00:00:00/task_id=create_employees_table/attempt=1.log
[2025-08-20, 04:02:42 UTC] (taskinstance.py:1165) INFO - Dependencies all met for <TaskInstance: process_employees.create_employees_table scheduled__2025-08-19T00:00:00:00 [queued]>
[2025-08-20, 04:02:42 UTC] (taskinstance.py:1165) INFO - Dependencies all met for <TaskInstance: process_employees.create_employees_table scheduled__2025-08-19T00:00:00:00 [queued]>
[2025-08-20, 04:02:42 UTC] (taskinstance.py:1362) INFO -
-----
[2025-08-20, 04:02:42 UTC] (taskinstance.py:1363) INFO - Starting attempt 1 of 1
[2025-08-20, 04:02:42 UTC] (taskinstance.py:1364) INFO -
-----
[2025-08-20, 04:02:42 UTC] (taskinstance.py:1383) INFO - Executing <Task(PostgresOperator): create_employees_table> on 2025-08-19 00:00:00:00
[2025-08-20, 04:02:42 UTC] (standard_task_runner.py:55) INFO - Started process 71 to run task
[2025-08-20, 04:02:42 UTC] (standard_task_runner.py:82) INFO - Running: ['***', 'tasks', 'run', 'process_employees', 'create_employees_table', 'scheduled__2025-08-19T00:00:00:00', '--job-id', '28', '--
[2025-08-20, 04:02:42 UTC] (standard_task_runner.py:83) INFO - Job 28: Subtask create_employees_table
[2025-08-20, 04:02:43 UTC] (task_command.py:376) INFO - Running <TaskInstance: process_employees.create_employees_table scheduled__2025-08-19T00:00:00:00 [running]> on host ebbf924e6830
[2025-08-20, 04:02:44 UTC] (taskinstance.py:1592) INFO - Exporting the following env vars:
```