

Coding Challenge

Apache Airflow: Features, Installation, and Pipeline Building Guide

Key Features of Apache Airflow

1. Dynamic Pipeline Generation

Workflows are written as Python code, allowing for dynamic DAG (Directed Acyclic Graph) creation.

2. Scalability

Supports scaling horizontally across multiple workers.

3. Extensibility

Provides a rich set of operators, hooks, sensors, and allows custom plugin development.

4. Monitoring and UI

Web-based UI to visualize, monitor, and manage workflows with detailed logs.

5. Task Dependency Management

Explicitly defines dependencies between tasks for better control.

6. Integration

Integrates with a wide range of systems (AWS, GCP, Azure, Hadoop, Databases, etc.).

7. Resiliency

Automatic retrying of failed tasks, SLA monitoring, and alerting.

8. Versioning and Modularity

DAGs are modular and version-controlled like standard code.

Installing Apache Airflow with Docker (Using YAML)

1. Pre-requisites

Install **Docker** and **Docker Compose** on your system.

Verify installation:

```
docker --version
```

```
docker compose version
```

2. Create a yaml File

Download the YAML file and store it in the folder(materials).

3. Access Airflow UI

Start all services:

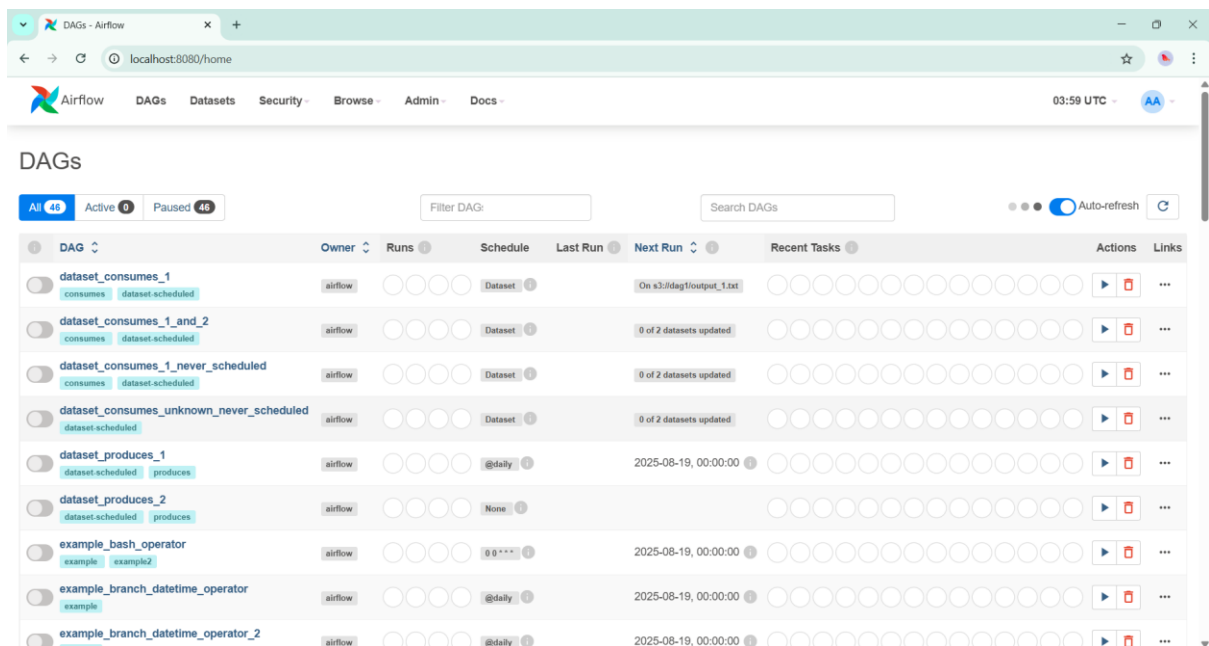
`docker compose up -d`

Open browser: <http://localhost:8080>

Steps to Build a Pipeline in Airflow

Step 1: Open Airflow Web UI

- Navigate to <http://localhost:8080>
- The Airflow home page will open.



Step 2: Create Connection

- Go to **Admin → Connections**.
- Add a new Postgres connection:
 - **Connection ID:** tutorial_pg_conn
 - **Connection Type:** Postgres
 - **Host:** postgres
 - **Database:** airflow

- **Login:** airflow
- **Password:** airflow
- **Port:** 5432
- Save the connection.

The screenshot shows the Airflow web interface at localhost:8080/home. The 'DAGs' tab is selected. A dropdown menu is open, showing options: Variables, Configurations, Connections, Plugins, Providers, Pools, and XComs. The 'Connections' option is highlighted. Below the menu, a table lists DAGs with columns: DAG, Owner, File, Last Run, Next Run, Recent Tasks, Actions, and Links. The table contains several DAGs, including 'dataset_consumes_1', 'dataset_consumes_1_and_2', 'dataset_consumes_1_never_scheduled', 'dataset_consumes_unknown_never_scheduled', 'dataset_produces_1', 'dataset_produces_2', 'example_branch_operator', and 'example_branch_datetime_operator'.

The screenshot shows the 'List Connection' page in the Airflow web interface at localhost:8080/connection/list/. The page has a search bar and a table with columns: Name, Type, Description, and Actions. The table is empty, and the message 'No records found' is displayed. The 'Record Count' is 0.

The screenshot shows the 'Add Connection' page in the Airflow web interface at localhost:8080/connection/add. The page has a form with the following fields:

- Connection Id ***: A text input field.
- Connection Type ***: A dropdown menu with 'Amazon Elastic MapReduce' selected. Below it, a message says: 'Connection Type missing? Make sure you've installed the corresponding Airflow Provider Package.'
- Description**: A text area.
- Run Job Flow Configuration**: A text area containing a JSON configuration:


```
{
  "Name": "MyClusterName",
  "ReleaseLabel": "emr-5.36.0",
  ...
}
```

 At the bottom, there are buttons for 'Save', 'Test', and a back arrow.

Connection Type: Postgres

Description:

Host: postgres

Schema:

Login: airflow

Password:

Port: 5432

Extra:

Buttons: Save, Test, Cancel

Added Row

List Connection

Search:

Actions:

Record Count: 1

Conn Id	Conn Type	Description	Host	Port	Is Encrypted	Is Extra Encrypted
tutorial_pg_conn	postgres		postgres	5432	False	False

Step 3: Create SQL File

- Go to dags/sql/
- Create tables.sql with SQL queries for creating required tables.

Step 4: Create Python DAG File

- Go to dags/
- Create process_employee.py:

```

1 import datetime
2 import pendulum
3 import os
4
5 import requests
6 from airflow.decorators import dag, task
7
8 from airflow.providers.postgres.hooks.postgres import PostgresHook
9 from airflow.providers.postgres.operators.postgres import PostgresOperator
10
11
12
13 @dag
14     dag_id="process_employees",
15     schedule="0 0 * * *",
16     start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
17     catchup=False,
18     dagrun_timeout=datetime.timedelta(minutes=60),
19 )
20
21 def process_employees():
22     create_employees_table = PostgresOperator(
23         task_id="create_employees_table",
24         postgres_conn_id="tutorial_pg_conn",
25         sql="""
26         CREATE TABLE IF NOT EXISTS employees (
27             "Serial Number" NUMERIC PRIMARY KEY,
28             "Company Name" TEXT,
29             "Employee Marksm" TEXT,
30             "Description" TEXT,
31             "Leave" INTEGER
32         )
33     """
34     )
35
36     create_employees_temp_table = PostgresOperator(
37         task_id="create_employees_temp_table",
38         postgres_conn_id="tutorial_pg_conn",
39         sql="""
40         DROP TABLE IF EXISTS employees_temp;
41         CREATE TABLE employees_temp (
42             "Serial Number" NUMERIC ,
43             "Company Name" TEXT,
44             "Employee Marksm" TEXT,
45             "Description" TEXT,
46             "Leave" INTEGER
47         )
48     """
49     )

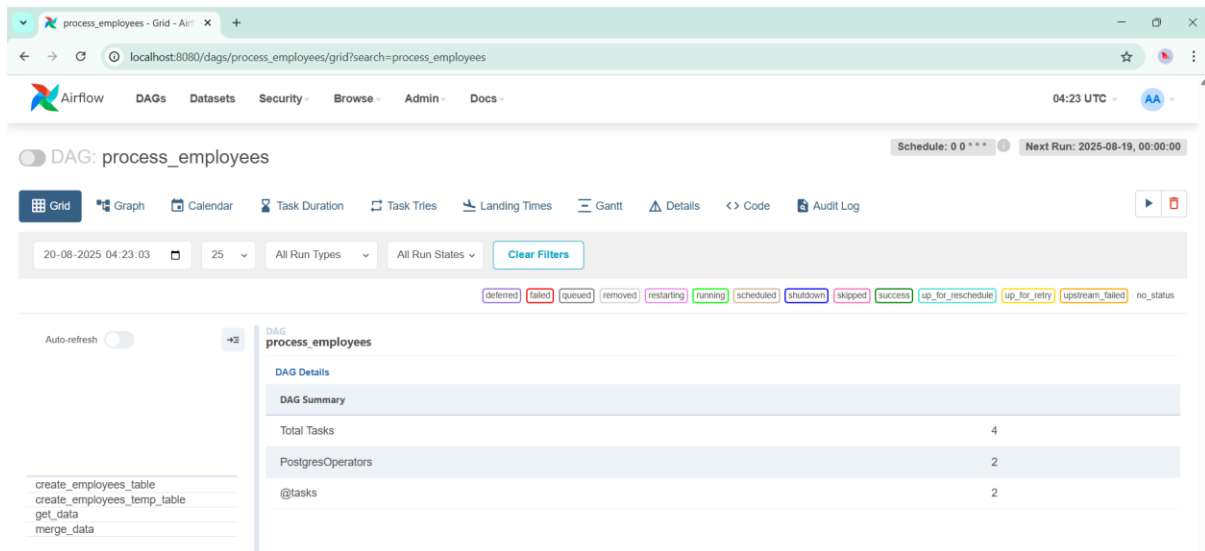
```

Step 5: Deploy the DAG

- Place process_employee.py in the dags/ directory.
- Airflow will detect it.

Step 6: Run the DAG

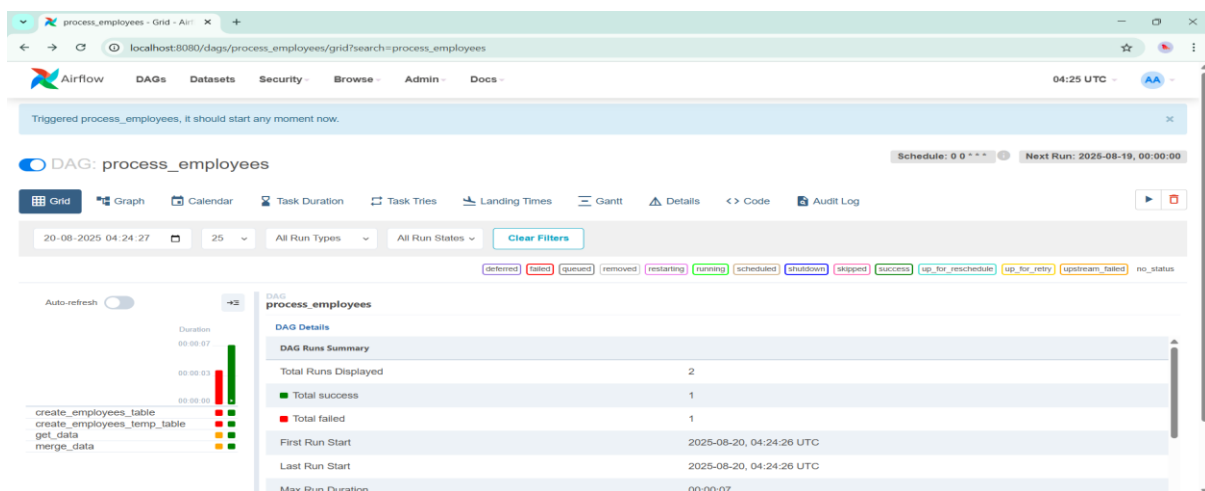
- Trigger the DAG from the web UI.
- Monitor task execution in **Graph View** or **Tree View**.



Viewing Pipelines in Airflow

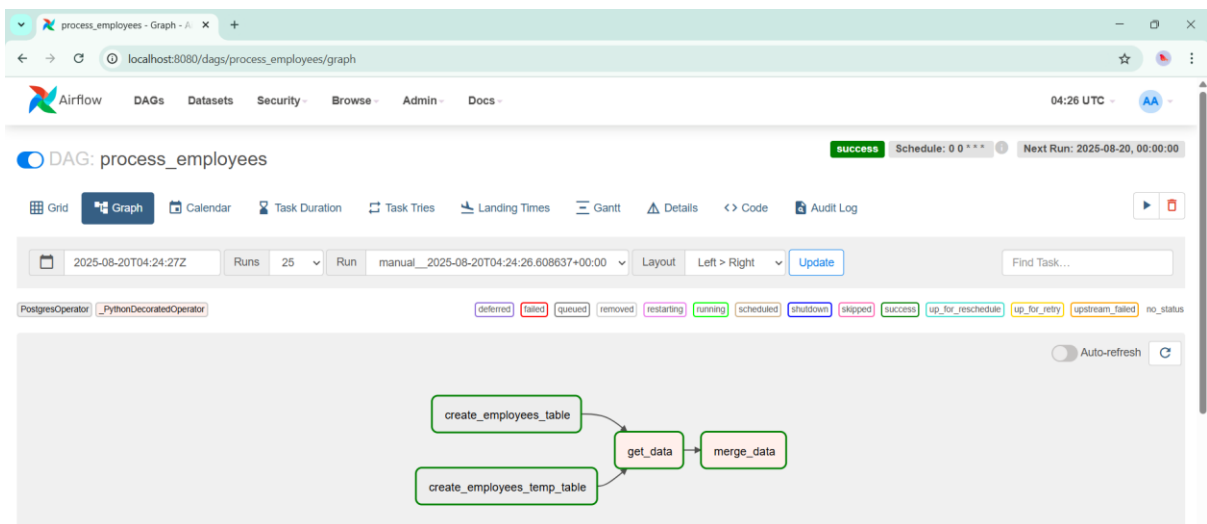
Grid View

- Displays DAG runs and tasks in a timeline-like grid.
- Each column represents a DAG run, and each row represents a task.
- Provides quick insights into task status across multiple runs.



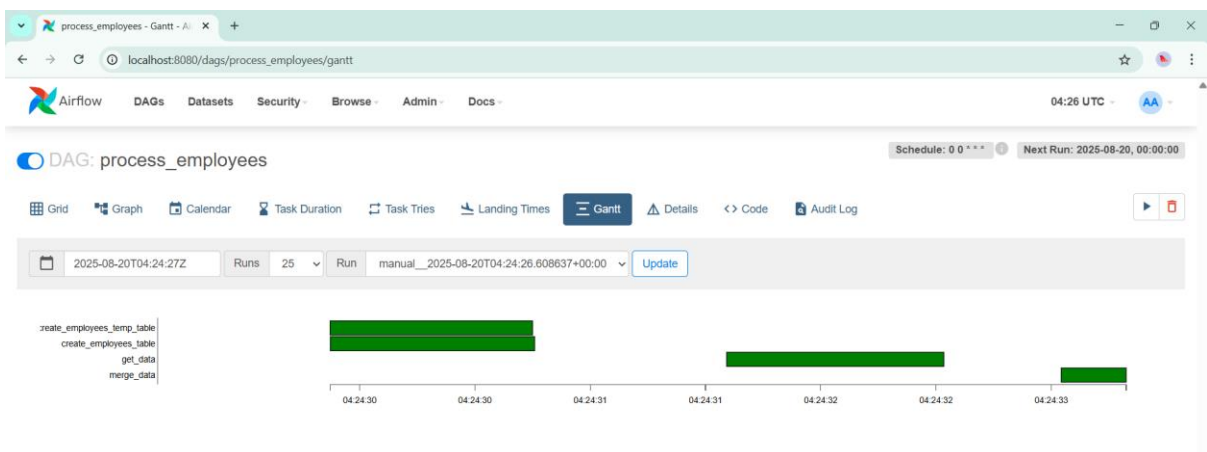
Graph View

- Shows DAG structure as a directed acyclic graph.
- Visualizes task dependencies and execution flow.
- Helpful for understanding pipeline logic.



Gantt View

- Visualizes task execution duration.
- Helps identify bottlenecks and overlapping tasks.
- Useful for performance monitoring.



DAG Details

- Shows overall information about a DAG.
- Includes schedule, description, start date, owners, and configuration.
- Links to logs, task instances, and code view.

The screenshot shows the Apache Airflow web interface at localhost:8080/dags/process_employees/details. The page title is 'DAG: process_employees'. The top navigation bar includes links for Grid, Graph, Calendar, Task Duration, Task Tries, Landing Times, Gantt, Details (active), Code, and Audit Log. The 'DAG Details' section shows a status bar with 'failed' (2), 'success' (4), and 'upstream_failed' (2) indicators. Below this is a table of DAG configuration:

Schedule Interval	0 0 * * *
Catchup	False
Started	True
End Date	None
Max Active Runs	0 / 16
Concurrency	16
Default Args	{}
Tasks Count	4
Task IDs	['create_employees_table', 'create_employees_temp_table', 'get_data', 'merge_data']
Relative file location	process_employees.py
Owner	airflow
Owner Links	None

Summary

Using **Docker Compose with YAML**, Apache Airflow can be set up quickly. A sample **Employee Processing Pipeline** demonstrates how to configure connections, create SQL scripts, define DAGs, and execute them. The Airflow UI provides powerful tools like **Grid View, Graph View, Gantt View, and DAG Details** for comprehensive monitoring and debugging.