

Airflow

Name: Priyeshwar

Mail: priyesh2664@gmail.com

What is Airflow?

Apache Airflow is an open-source workflow orchestration tool used by data engineers to programmatically author, schedule, and monitor workflows (pipelines). Workflows are defined in Python code and represented as Directed Acyclic Graphs (DAGs).

Key Features of Apache Airflow

Workflow Management

- **DAGs (Directed Acyclic Graphs):** Workflows are represented as DAGs ensuring tasks run in the correct sequence without cycles.
- **Task Dependencies:** Define upstream and downstream relationships clearly.

Scheduling & Execution

- **Scheduler & Executors:** Tasks are scheduled and distributed across executors (Local, Celery, Kubernetes, etc.).
- **Flexible Scheduling:** Cron-like expressions or time intervals; supports backfilling.

Monitoring & Management

- **Web UI:** Visual interface for monitoring DAG runs, logs, and task states.
- **Logging:** Centralized logs stored locally or in external systems (S3, GCS, Elasticsearch).
- **Alerts & Retries:** Automatic retries, failure callbacks, and alerts on errors.

Scalability & Integration

- **Horizontal Scalability:** Distribute workloads across multiple workers.
- **Integrations:** Works with AWS, GCP, Azure, Spark, Hive, Presto, Kubernetes, and many other tools.
- **Event-based Triggers:** Sensors for external jobs, file arrivals, and system events.

Security

- **Role-Based Access Control (RBAC):** Manage permissions by roles.
 - **Authentication:** Integration with LDAP, OAuth, and other authentication systems.
-

Steps to Build and Run an Airflow Pipeline

Step 1: Install and Initialize Airflow

- Install Airflow (e.g., with `pip install apache-airflow`).
 - Initialize the Airflow database:
 - `airflow db init`
-

Step 2: Create the DAG File

- Write your DAG definition in a `.py` file.
- Save it inside the Airflow `dags/` folder (default: `~/airflow/dags/`).

Example DAG file: `~/airflow/dags/process_employees.py`

```
import datetime
import pendulum
import os
import requests

from airflow.decorators import task, dag
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.providers.postgres.operators.postgres import PostgresOperator

@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 = PostgresOperator(

 task_id="create_employees_table",

 postgres_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 = PostgresOperator(

 task_id="create_employees_temp_table",

 postgres_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

);""",

)

Step 3: Create the task

```
@task
def get_data():
    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.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()

```

Step 4: Start Airflow Services

- Start the scheduler:
 - airflow scheduler
 - Start the webserver:
 - airflow webserver -p 8080
-

Step 5: Access the Airflow UI

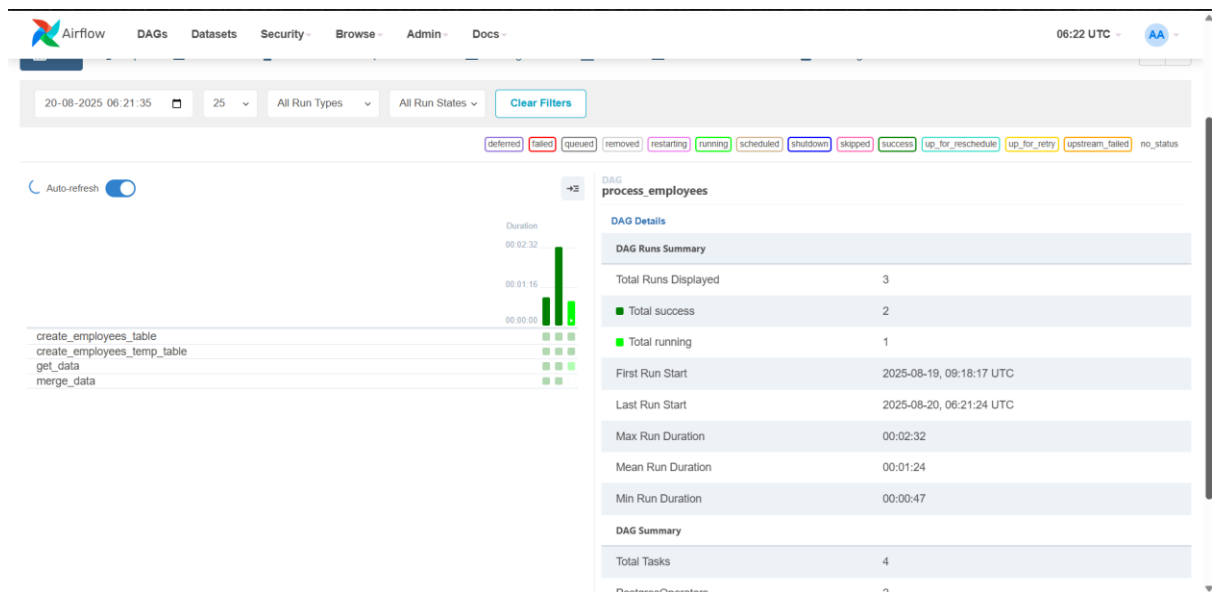
- Open your browser and go to: <http://localhost:8080>.
- Log in with the user created earlier.

The screenshot shows the Apache Airflow web interface. At the top, there's a navigation bar with links for Airflow, DAGs, Datasets, Security, Browse, Admin, and Docs. The current time is 06:19 UTC. Below the navigation bar, the 'DAGs' section is active. It shows a list of DAGs with columns for DAG name, Owner, Runs, Schedule, Last Run, Next Run, Recent Tasks, Actions, and Links. The DAGs listed are 'postgres_pipeline_dag', 'postgres_setup_dag', and 'process_employees'. The 'process_employees' DAG is highlighted, showing its recent tasks as 'create_employees_table', 'create_employees_temp_table', 'get_data', and 'merge_data'. The bottom of the page shows the version information: Version: v2.4.2 and Git Version: .release:2.4.2+febf35500d5de172e25280e5f5492257f898fd5.

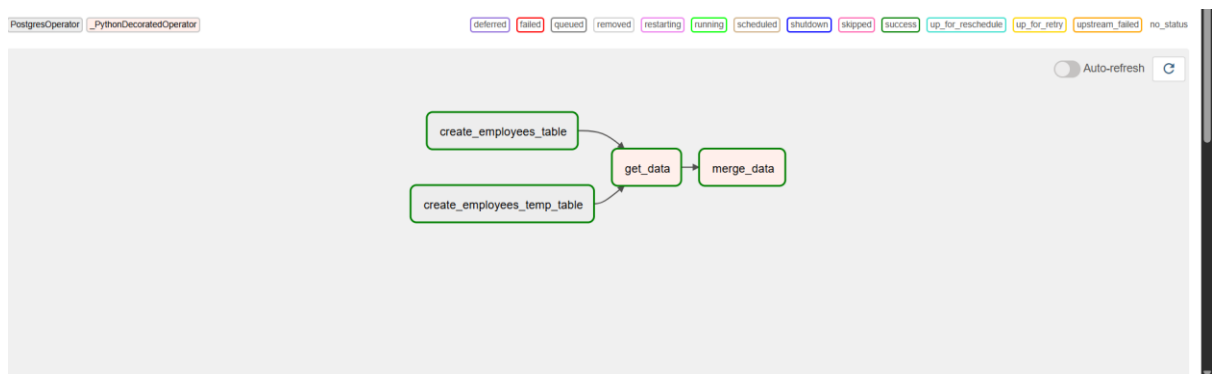
Step 6: Trigger and Monitor the DAG

- Enable the DAG in the UI.
- Trigger a run manually or wait for the schedule.

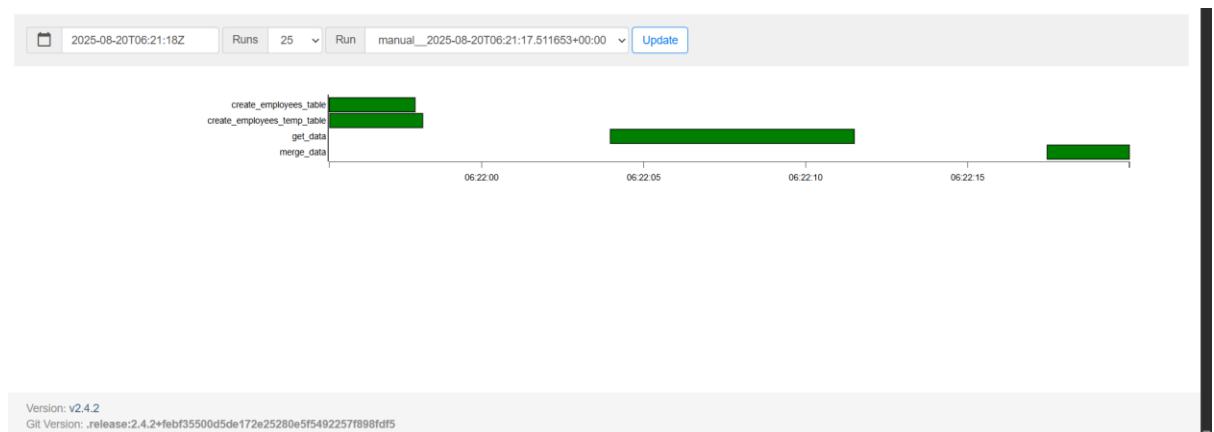
The screenshot shows the Apache Airflow web interface for the 'process_employees' DAG. The top navigation bar is the same as in Step 5. The current time is 06:21 UTC. Below the navigation bar, the 'DAG: process_employees' section is active. It shows the DAG's schedule as '0 0 * * *' and the next run as '2025-08-20, 00:00:00'. The DAG is currently in a 'scheduled' state. Below the DAG details, there's a 'DAG Summary' section with a table showing the DAG's performance metrics. The table has two columns: 'Metric' and 'Value'. The metrics listed are 'Total Runs Displayed', 'Total success', 'First Run Start', 'Last Run Start', 'Max Run Duration', 'Mean Run Duration', and 'Min Run Duration'. The values are: 'Total Runs Displayed: 2', 'Total success: 2', 'First Run Start: 2025-08-19, 09:18:17 UTC', 'Last Run Start: 2025-08-20, 04:49:36 UTC', 'Max Run Duration: 00:02:32', 'Mean Run Duration: 00:01:43', and 'Min Run Duration: 00:00:54'. The bottom of the page shows the version information: Version: v2.4.2 and Git Version: .release:2.4.2+febf35500d5de172e25280e5f5492257f898fd5.



- **Graph View:** Visual DAG representation showing task dependencies.



- **Gantt View:** Duration and overlap of tasks.



- View logs and retries for each task.

