

# Airflow Coding Assessment

By Esaq A

Apache Airflow is an open-source framework which is designed for monitoring and scheduling workflows. It excels at orchestrating complex data pipeline activities. We will now be seeing some features of Airflow, the creation of a simple pipeline, and how to visualize it in the Airflow UI.

## Features:


1. *Dynamic Pipeline Generation*: Every pipeline is described in native Python, enabling you to craft workflows that can adapt at runtime by embedding loops, conditionals, or external configuration.
2. *Extensibility*: It allows users to create custom operators, sensors, and plugins to integrate with various systems and services. It also integrates with various cloud platforms and databases.
3. *Scalability*: It can scale a single machine to a cluster of machines to handle increasing workload and parallel execution.
4. *Monitoring and Logging*: It has a user-friendly interface for monitoring workflow progress, task status, logs, and gives alert for failed tasks
5. *Scheduling and Task Execution*: It supports cron-like scheduling, allowing for flexible task triggering based on time intervals and external events.

## Building a simple pipeline:

### 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:

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

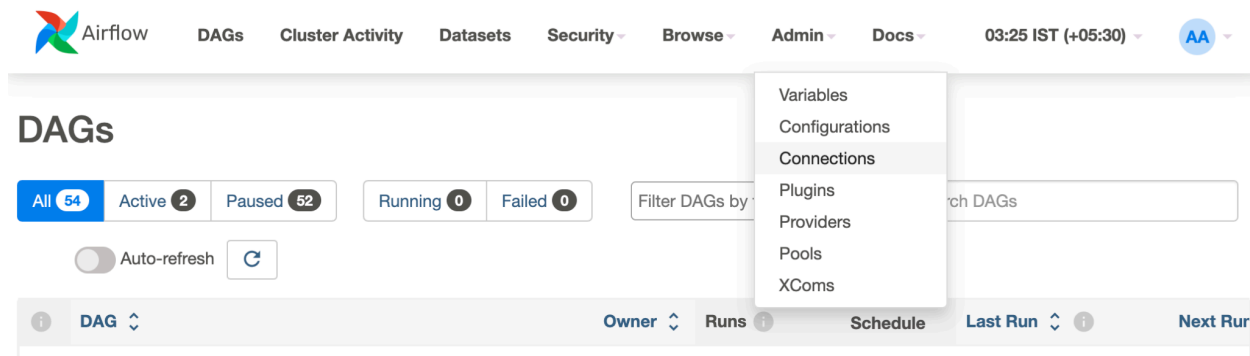
Search DAGs

☐ Auto-refresh 

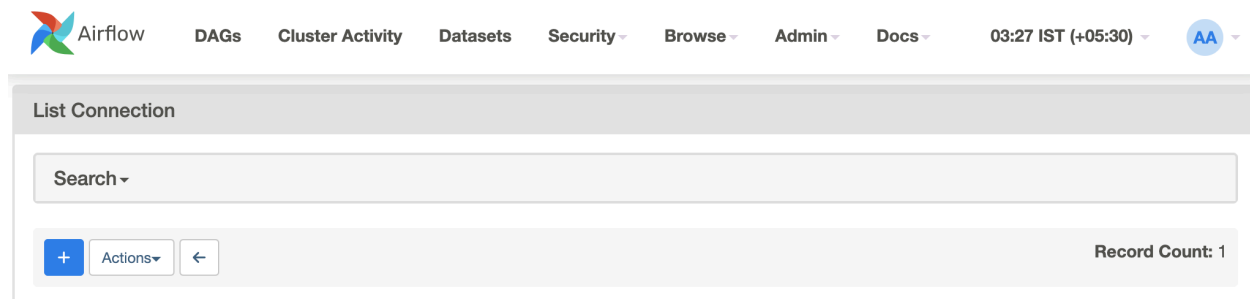
DAG	Owner	Runs	Schedule	Last Run	Next Run
 dataset_consumes_1 consumes dataset-scheduled	airflow	   	Dataset	2025-08-18, 17:11:53	On s3://da
 dataset_consumes_1_and_2 consumes dataset-scheduled	airflow	   	Dataset		0 of 2 data
 dataset_consumes_1_never_scheduled consumes dataset-scheduled	airflow	  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

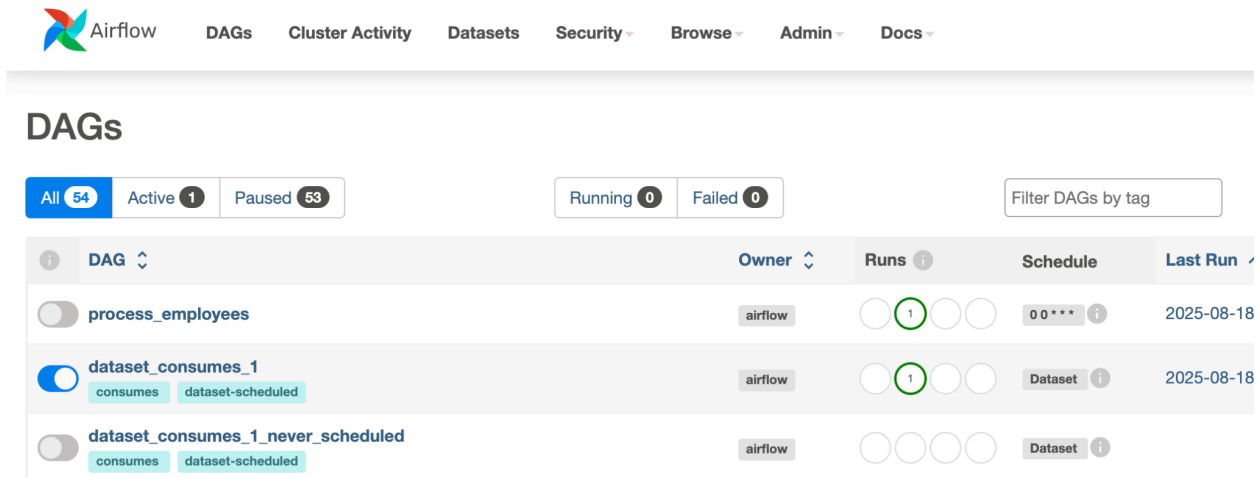
dag = ProcessEmployees()

```

- 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

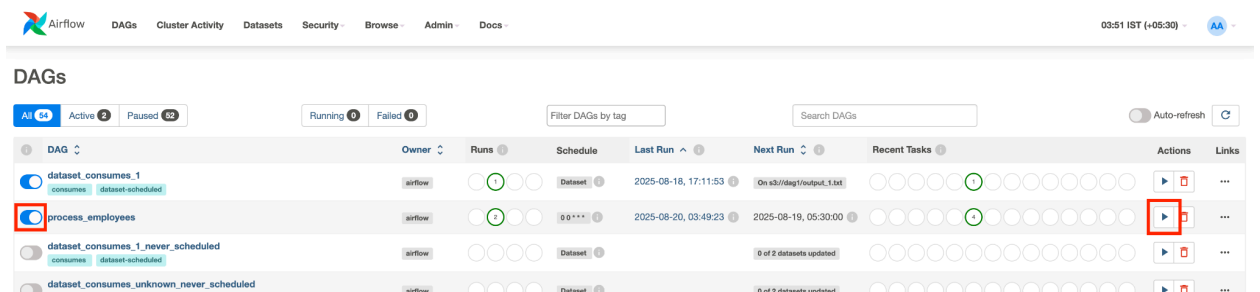
- Now open the Airflow UI, and look for your DAG's name.











The screenshot shows the Airflow DAGs list page. The top navigation bar includes links for DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs. The main section is titled 'DAGs' and features a filter bar with buttons for 'All' (54), 'Active' (1), and 'Paused' (53). There are also buttons for 'Running' (0) and 'Failed' (0), and a 'Filter DAGs by tag' input field. The table below lists the DAGs:

DAG	Owner	Runs	Schedule	Last Run
<input type="checkbox"/> process_employees	airflow	1	0 0 * * *	2025-08-18
<input checked="" type="checkbox"/> dataset_consumes_1 consumes dataset-scheduled	airflow	1	Dataset	2025-08-18
<input type="checkbox"/> dataset_consumes_1_never_scheduled consumes dataset-scheduled	airflow		Dataset	

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



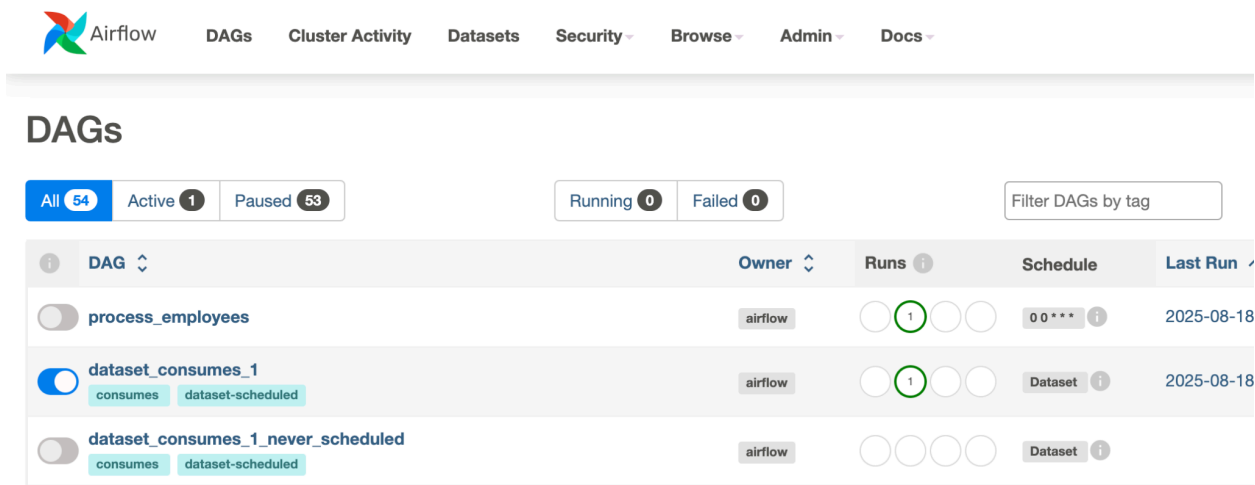
The screenshot shows the Airflow DAGs list page with the 'process\_employees' DAG selected. The 'process\_employees' DAG is highlighted, and its slider is toggled 'ON'. The 'Actions' column shows a play button (trigger) and a stop button (cancel). The table below lists the DAGs:

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
<input checked="" type="checkbox"/> dataset_consumes_1 consumes dataset-scheduled	airflow	1	Dataset	2025-08-18, 17:11:53	On s3://dag/output_1.txt		 	...
<input checked="" type="checkbox"/> process_employees	airflow	1	0 0 * * *	2025-08-20, 03:49:23	2025-08-19, 05:30:00		 	...
<input type="checkbox"/> dataset_consumes_1_never_scheduled consumes dataset-scheduled	airflow		Dataset		0 of 2 datasets updated		 	...
<input type="checkbox"/> dataset_consumes_unknown_never_scheduled	airflow		Dataset		0 of 2 datasets updated		 	...

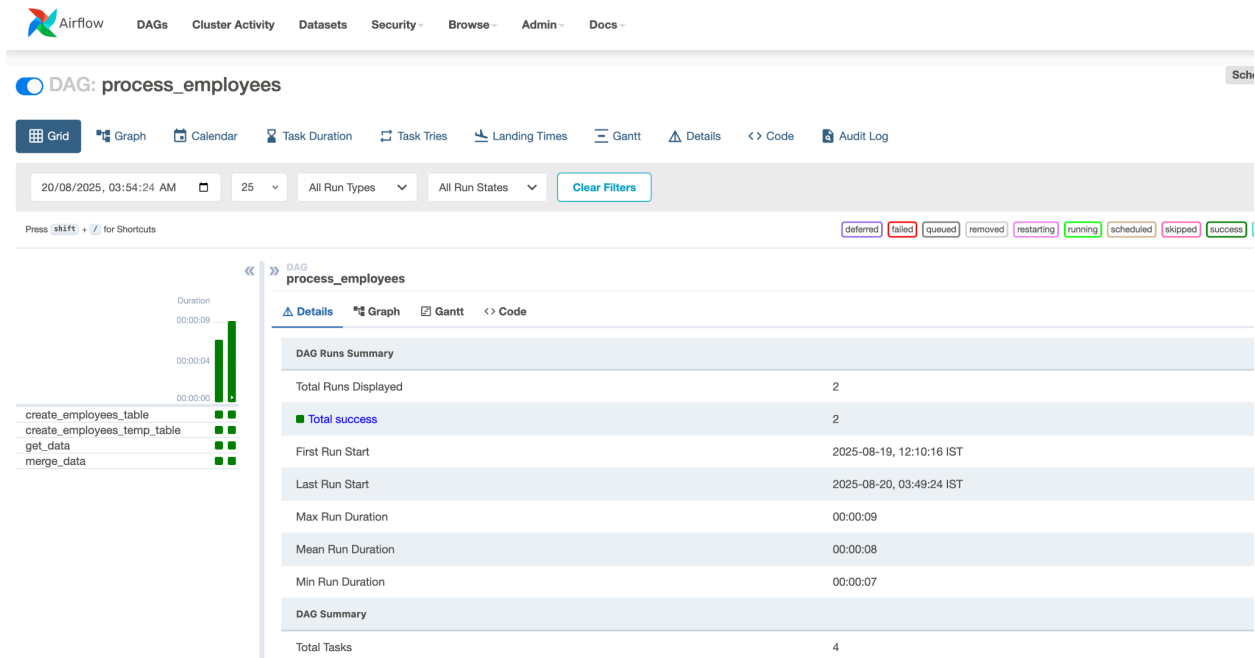
# Viewing our pipeline in Airflow:

It enables us to view our pipeline in various perspectives of our workflows.

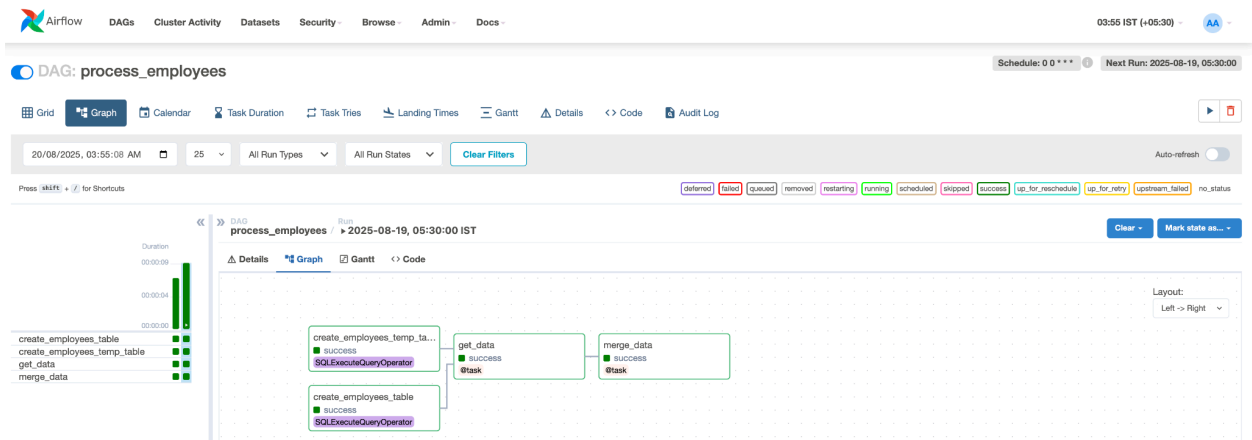
## DAGs View: The main dashboard listing all our DAGs



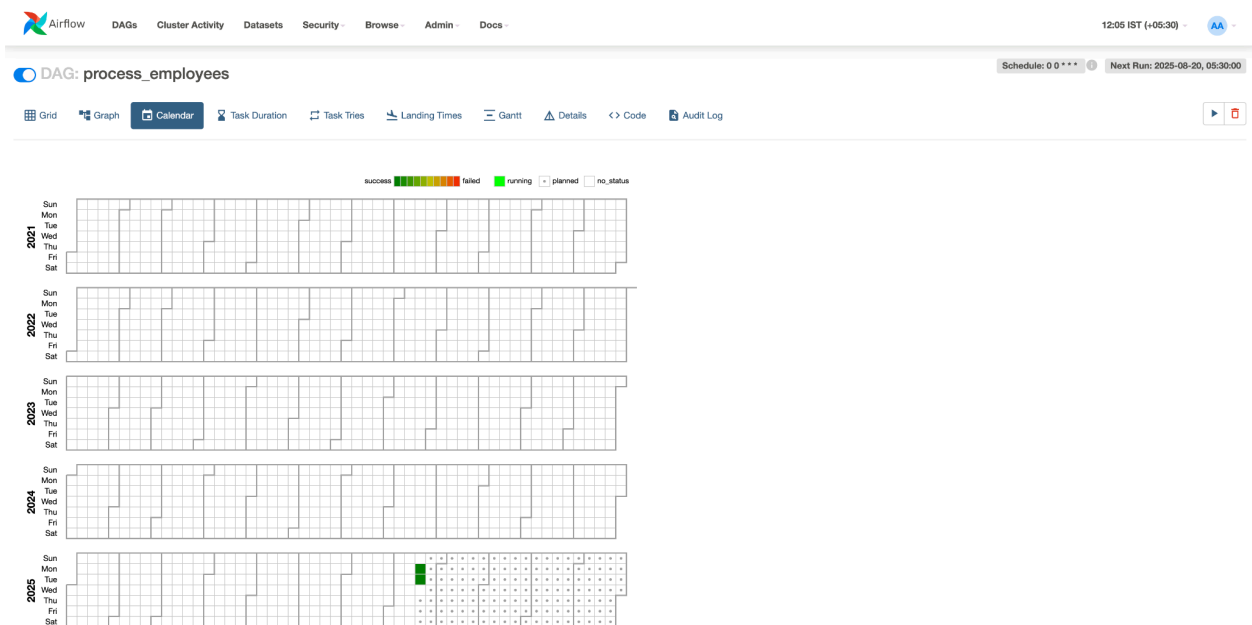
## Grid View: It shows the status of our DAG runs over time in grid format



**Graph View:** It provides a visual representation of our DAG's structure.



**Calendar View:** Displays the state of our DAG runs on a calendar.





## Code View: Allows us to view our python code of DAG directly in UI.

The screenshot displays the Apache Airflow web interface for a DAG named 'process\_employees'. The top navigation bar includes links for Airflow, DAGs, Cluster Activity, Datasets, Security, Browse, Admin, and Docs. The current time is 12:06 IST (+05:30). The DAG's schedule is '0 0 \* \* \*' and the next run is scheduled for 2025-08-20, 05:30:00.

Below the navigation bar, there are tabs for Grid, Graph, Calendar, Task Duration, Task Times, Landing Times, Gantt, Details, Code, and Audit Log. The 'Code' tab is currently selected. A filter bar shows the date '20/08/2025, 12:05:57 PM', a dropdown for '25', and filters for 'All Run Types' and 'All Run States'. A 'Clear Filters' button is also present. An 'Auto-refresh' toggle is on the right.

A status bar at the top of the code view lists various task states: deferred, failed, queued, removed, restarting, running, scheduled, skipped, success, up\_for\_reschedule, up\_for\_retry, upstream\_failed, and no\_status.

On the left side, a 'Task Duration' bar chart shows the duration of tasks. Below it, a table lists tasks and their durations:

Task	Duration
create_employees_table	00:00:00
create_employees_temp_table	00:00:00
get_data	00:00:00
merge_data	00:00:00

The main area shows the Python code for the DAG, parsed at 2025-08-20, 12:05:47 IST. The code is as follows:

```
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.common.sql.operators.sql import SQLExecuteQueryOperator
9
10
11 @dag(
12     dag_id="process_employees",
13     schedule="0 0 * * *",
14     start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
15     catchup=False,
16     default_timeout=datetime.timedelta(minutes=60),
```

A 'Toggle Wrap' button is located in the top right corner of the code editor.