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:

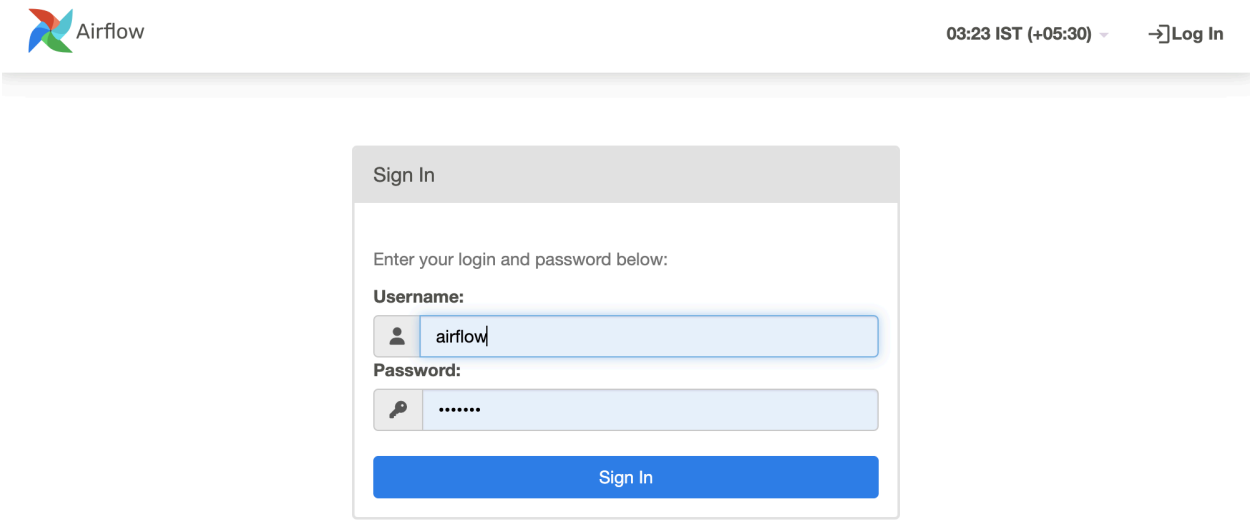
1. *Dynamic Pipeline Generation:* Every pipeline is described in native Python, enabling you to craft workflows that can adapt at runtime by embedding loops, conditionals, or external configuration.
2. *Extensibility:* It allows users to create custom operators, sensors, and plugins to integrate with 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:

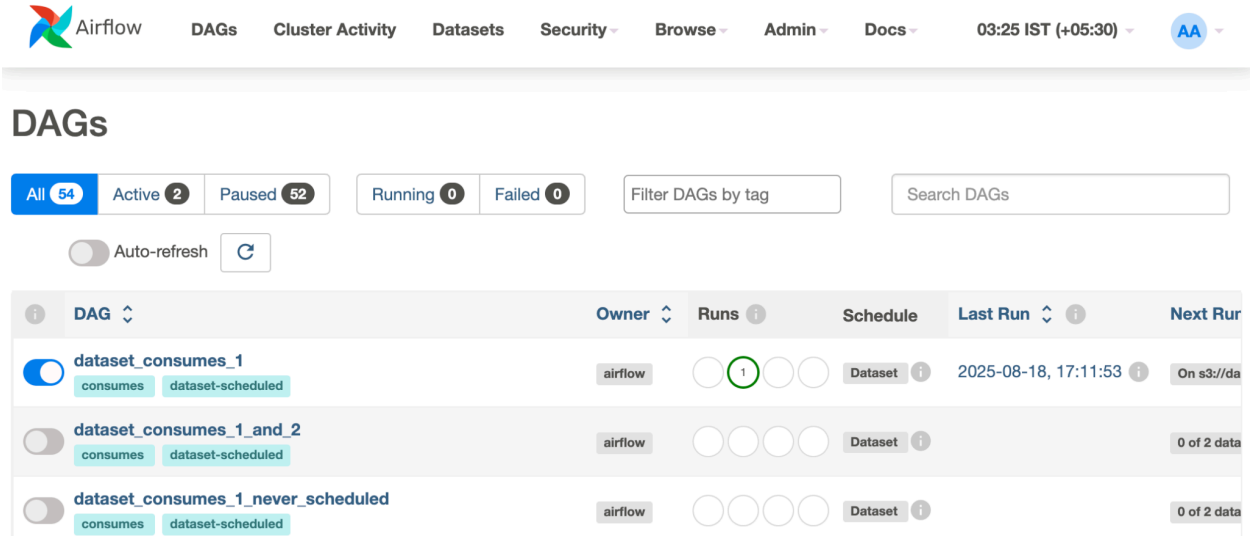
1 . Open your Airflow UI

- Open this website in browser http://localhost:8080

- Enter your username and password

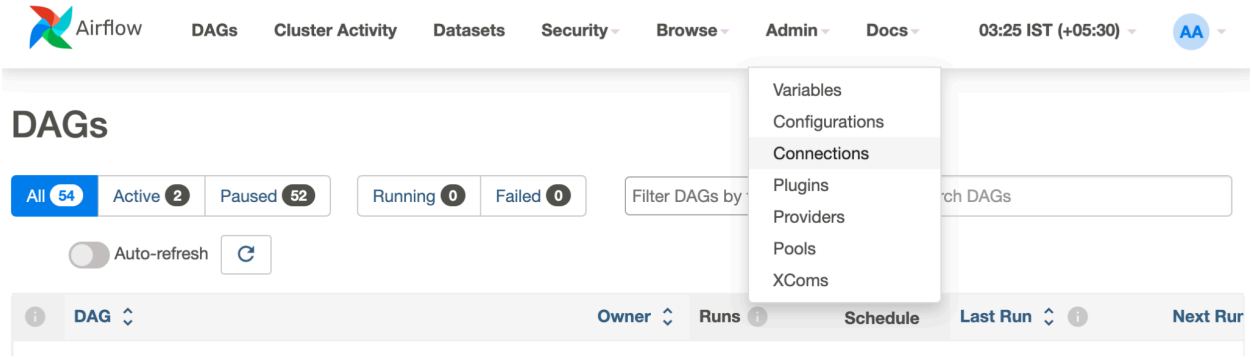


- You will land on the Airflow Dashboard



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",
```

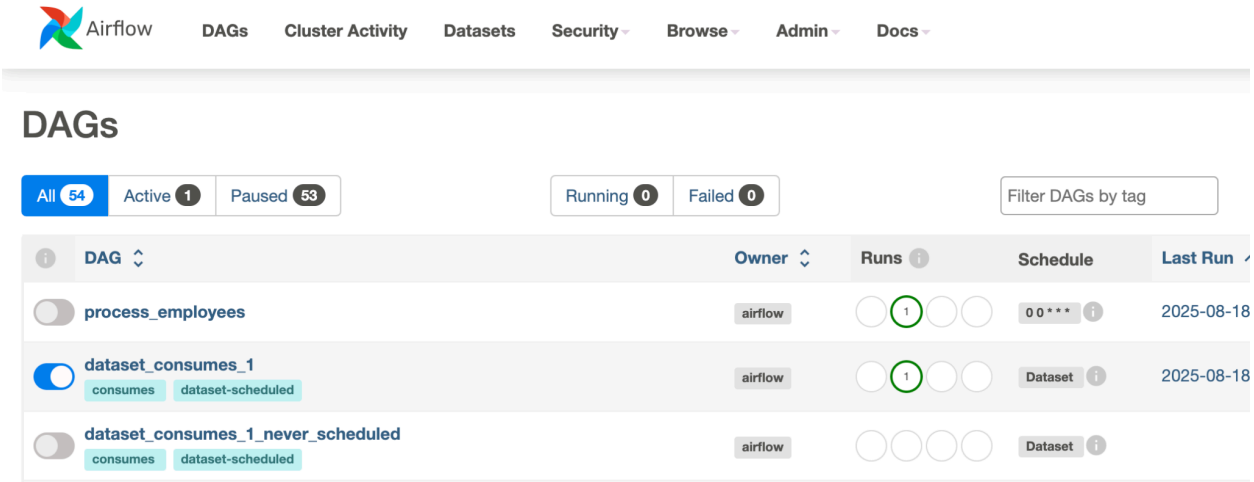
```
        sql="""
            DROP TABLE IF EXISTS employees_temp;
            CREATE TABLE employees_temp (
                "Serial Number" NUMERIC PRIMARY KEY,
                "Company Name" TEXT,
                "Employee Markme" TEXT,
                "Description" TEXT,
                "Leave" 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()
```

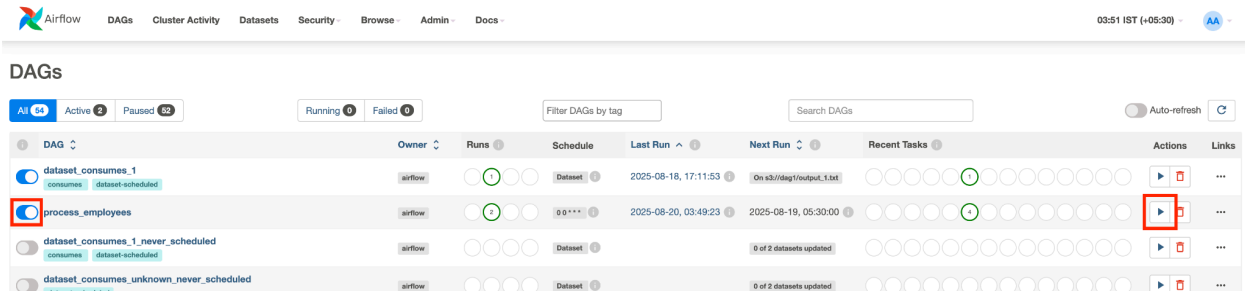
- Once the file is saved in the above said directory, we will be able to see our DAG on the Airflow UI.

4 . Trigger and Explore your DAG

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



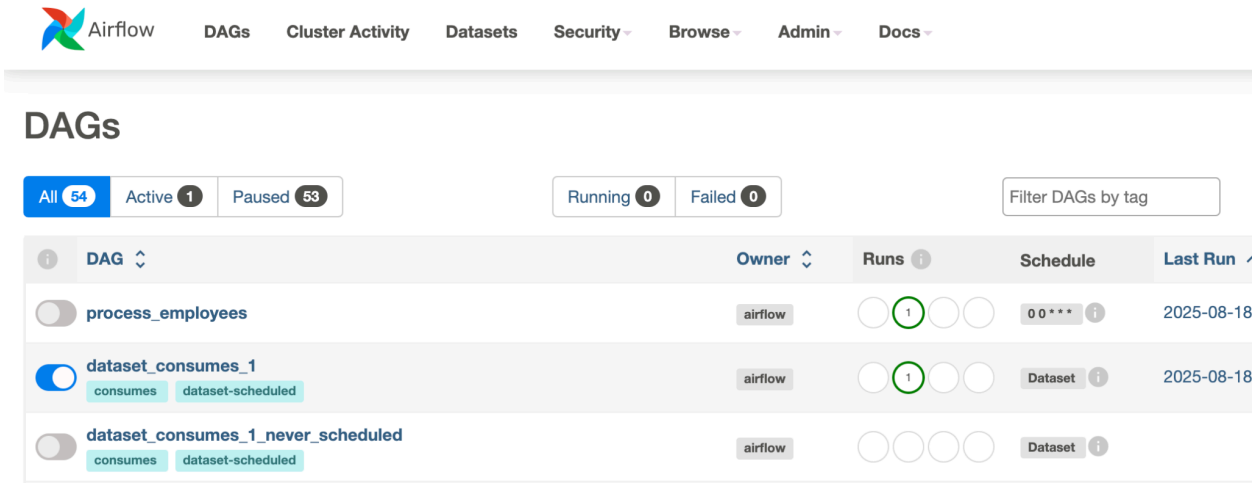
- Toggle the slider near your DAG "ON" and click on the play button to trigger a run.



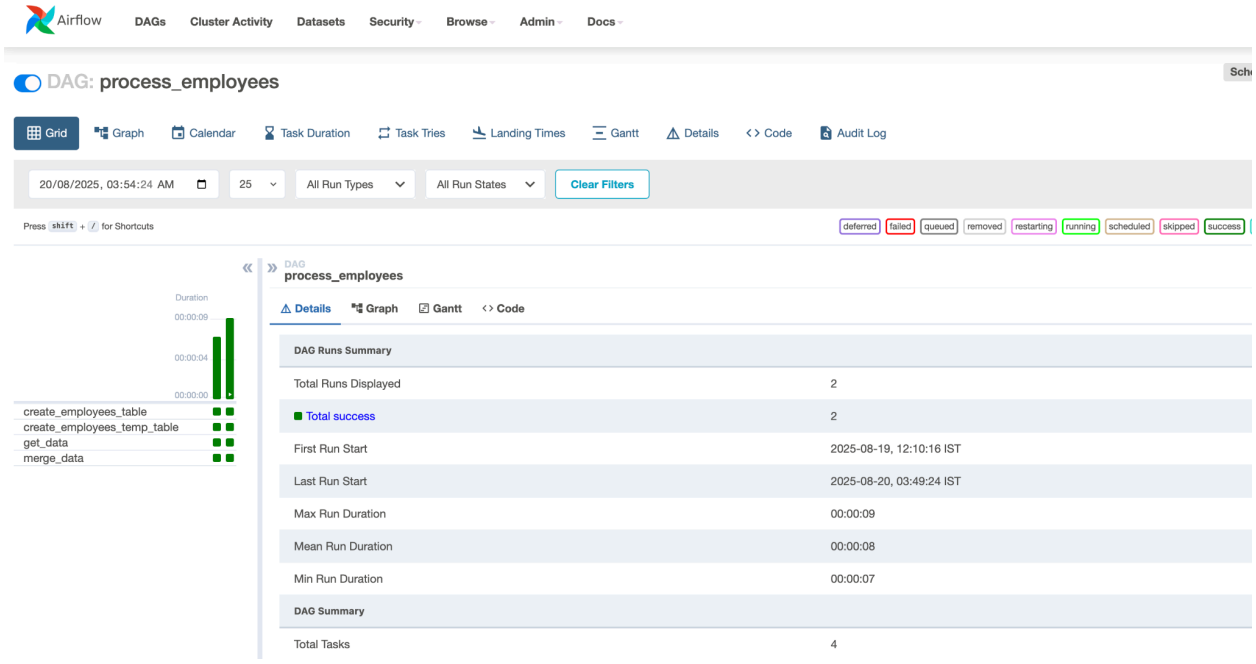
Viewing our pipeline in Airflow:

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

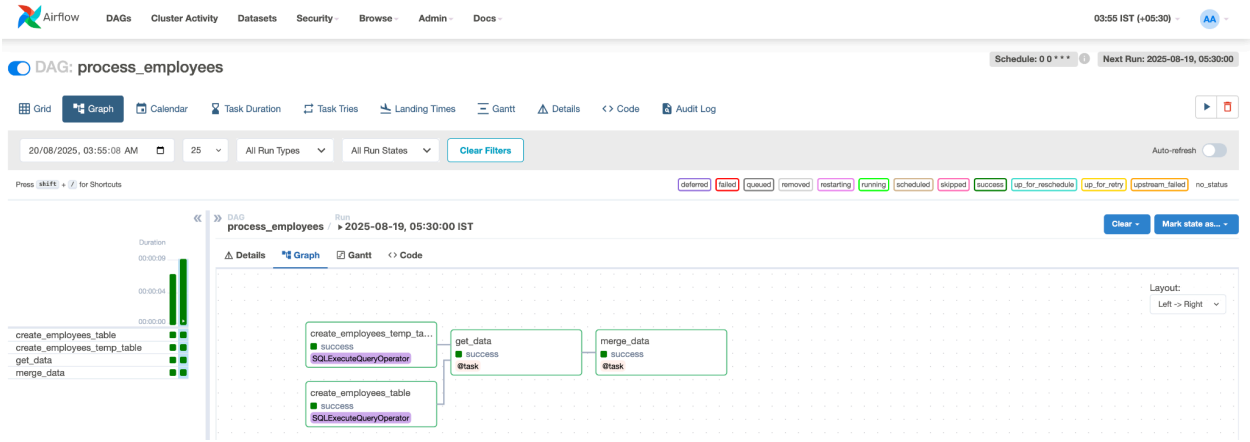
DAGs View: The main dashboard listing all our DAGs



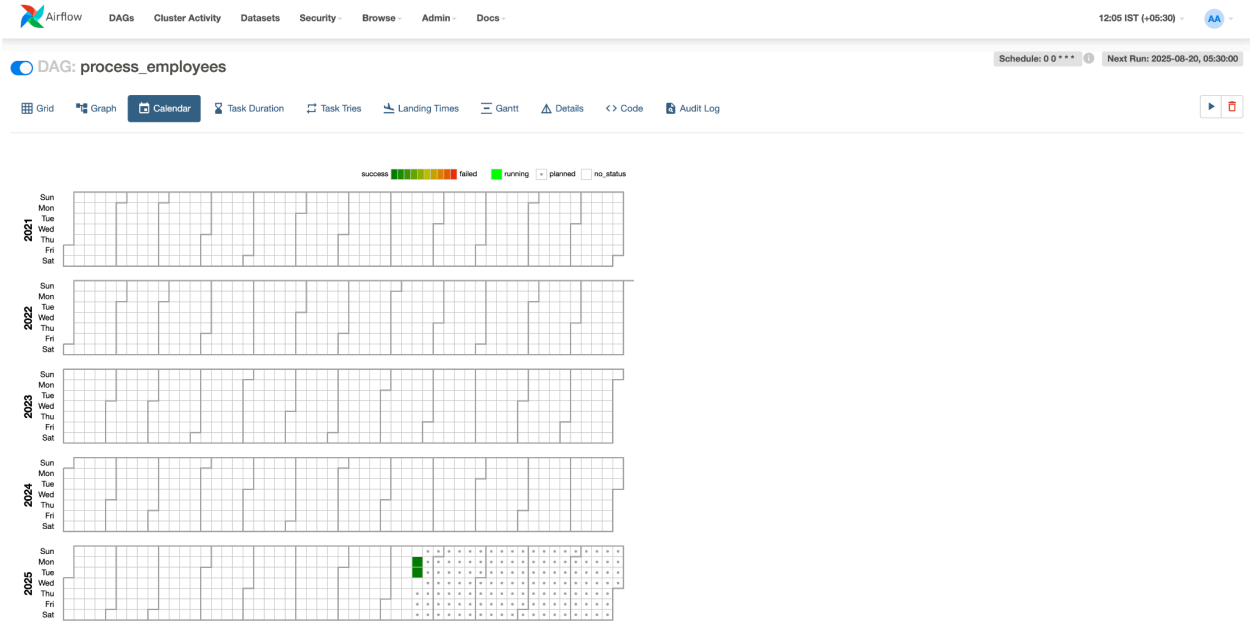
Grid View: It shows the status of our DAG runs over time in grid format



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.