Nombre: Dilan Zurita

NRC: 10049

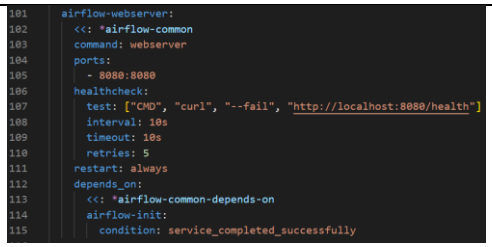
Implementación Apache Airflow

Objetivo General

Implementar un ejemplo que demuestre la capacidad de Apache Airflow para conectarse a una base de datos, realizar operaciones de creación y manipulación de datos, y automatizar el flujo de trabajo de inserción y eliminación de registros en intervalos regulares.

Servicios a utilizar

Servicio	Rol a desempeñar	Configuracion en archivo yaml
Docker compose	Crea, define y configura los contenedores que se van a usar mediante el archivo yaml	Indefinido
Postgres	Base de datos donde se escribirán y almacenarán los registros que deseemos.	

Airflow – web server	La interfaz web de Airflow proporciona una forma visual de administrar y monitorear los flujos de trabajo.	
Airflow – scheduler	El Scheduler es responsable de la planificación y ejecución de las tareas definidas en los flujos de trabajo de Airflow.	
Airflow – worker	Es un componente que se encarga de ejecutar las tareas de un flujo de trabajo de Airflow.	
Redis	Capa de caché para facilitar la carga de datos.	

Proceso

1. **Creacion de archivo .yaml**
   La creación de este archivo nos permitirá levantar nuestros contenedores con las especificaciones que necesitemos, en este caso se han creado los especificados en el cuadro de servicios, en dicho cuadro podemos ver la configuración de cada uno de nuestros contenedores.
   Adicional a esto se ha configurado de la siguiente manera las variables de entorno de apache airflow:

```yaml
image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.3.3}
# build: .
environment:
  &airflow-common-env
  AIRFLOW__CORE__EXECUTOR: CeleryExecutor
  AIRFLOW__DATABASE__SQL_ALCHEMY_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow
  # For backward compatibility, with Airflow <2.3
  AIRFLOW__CORE__SQL_ALCHEMY_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow
  AIRFLOW__CELERY__RESULT_BACKEND: db+postgresql://airflow:airflow@postgres/airflow
  AIRFLOW__CELERY__BROKER_URL: redis://:@redis:6379/0
  AIRFLOW__CORE__FERNET_KEY: ''
  AIRFLOW__CORE__DAGS_ARE_PAUSED_AT_CREATION: 'true'
  AIRFLOW__CORE__LOAD_EXAMPLES: 'false'
  AIRFLOW__SCHEDULER__DAG_DIR_LIST_INTERVAL: 5
  AIRFLOW__API__AUTH_BACKENDS: 'airflow.api.auth.backend.basic_auth'
  _PIP_ADDITIONAL_REQUIREMENTS: ${_PIP_ADDITIONAL_REQUIREMENTS:-}
volumes:
  - ./dags:/opt/airflow/dags
  - ./logs:/opt/airflow/logs
  - ./plugins:/opt/airflow/plugins
user: "${AIRFLOW_UID:-50000}:0"
```

2. **Creación del DAG**

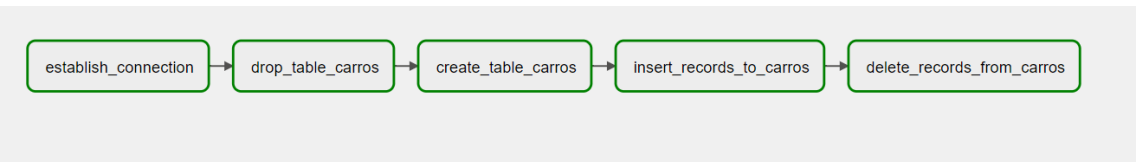
Un DAG en pocas palabras es la representación de un flujo de trabajo de manera acíclica o no repetitiva, para la creación de este procedemos a crear nuestro documento dag_carga.py en el cual asignaremos varias tareas, pero antes presentamos la configuración general del DAG:

```python
default_args = {
    'owner': 'Dilan_Zurita',
    'start_date': datetime(2023, 7, 6),
    'retries': 1,
    'retry_delay': timedelta(minutes=1),
}
```

En esta especificamos el dueño del proceso (owner), la fecha de inicio (start_date), el numero de reintentos en caso de que la tarea falle (retries) y el intervalo de tiempo entre los reintentos de una tarea en caso de fallo (retry_delay).

Una vez especificada la configuración general del DAG veamos la estructura lógica que seguirán nuestros trabajos.

```python
# Definir las dependencias entre las tareas
establish_connection_task >> drop_table_task >> create_table_task >> insert_records_task >> delete_records_task
```



Una vez establecido nuestro flujo de trabajo haciendo uso de las dependencias procedemos a crear las acciones que realizará cada trabajo.

- Establecer conexión a postgres

```
establish_connection_task = PostgresOperator(
    task_id='establish_connection',
    postgres_conn_id='postgres_default',  # Debes configurar una conexión a PostgreSQL en Airflow
    sql='SELECT 1;',
    dag=dag
)
```

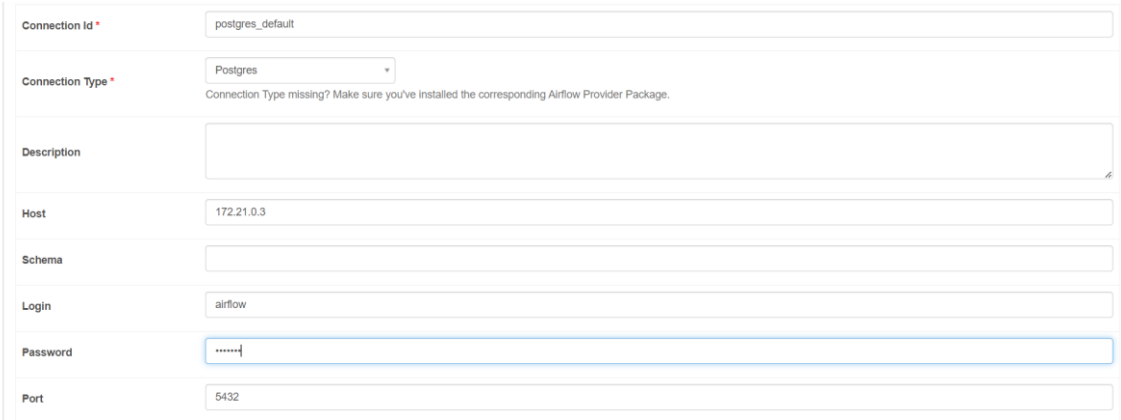
Para establecer la conexión a postgres debemos tomar en cuenta que se debe crear una conexión previa en el ambiente de Airflow para ellos seguimos los siguientes pasos:

1. Buscamos si la conexión existe, de no existir la creamos

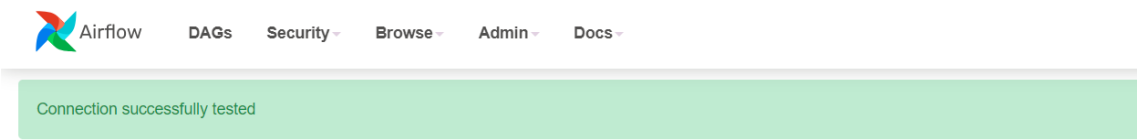
2. Configuramos los campos de la conexión, en el campos host debemos utilizar la IP del contenedor de nuestro servicio postgres el cual lo podemos obtener de la siguiente manera:

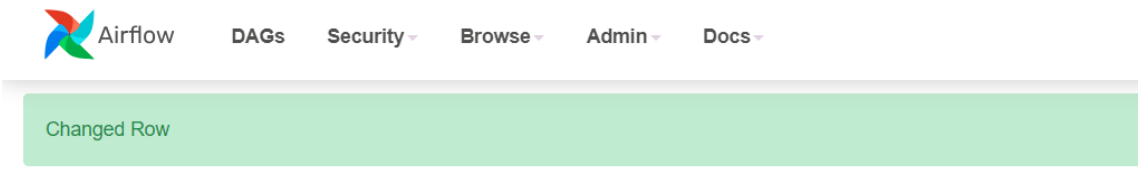
```
PS C:\Users\dilan> docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS                   PORTS
          NAMES
f39ed684ced6   apache/airflow:2.3.3   "/usr/bin/dumb-init …"   4 minutes ago   Up 4 minutes (healthy)   8080/tcp
          proyecto_apache-airflow-scheduler-1
750f812c0641   apache/airflow:2.3.3   "/usr/bin/dumb-init …"   4 minutes ago   Up 4 minutes (healthy)   8080/tcp
          proyecto_apache-airflow-worker-1
ce88f781e1d3   apache/airflow:2.3.3   "/usr/bin/dumb-init …"   4 minutes ago   Up 4 minutes (healthy)   0.0.0.0:8080->80
80/tcp    proyecto_apache-airflow-webserver-1
8763da3dd690   apache/airflow:2.3.3   "/usr/bin/dumb-init …"   4 minutes ago   Up 4 minutes (healthy)   8080/tcp
          proyecto_apache-airflow-triggerer-1
6c3623a01a5b   redis:latest           "docker-entrypoint.s…"   4 minutes ago   Up 4 minutes (healthy)   6379/tcp
          proyecto_apache-redis-1
a647e1826538   postgres:13            "docker-entrypoint.s…"   4 minutes ago   Up 4 minutes (healthy)   5432/tcp
          proyecto_apache-postgres-1
PS C:\Users\dilan> docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' a647e1826538
172.20.0.2
PS C:\Users\dilan>
```

Una vez obtenemos la IP procedemos a completar el resto de los campos.



3. Testeamos y guardamos la conexión





- Borrar tabla

```
drop_table_task = PostgresOperator(
    task_id='drop_table_carros',
    postgres_conn_id='postgres_default',  # Debes configurar una conexión a PostgreSQL en Airflow
    sql='DROP TABLE IF EXISTS carros;',
    dag=dag
)
```

Este proceso es necesario ya que si existiese una tabla previamente creada con el mismo nombre el proceso entraría en conflicto, esto dependerá mucho de la base de datos que se esté usando.

- Crear tabla Carros

```
create_table_task = PostgresOperator(
    task_id='create_table_carros',
    postgres_conn_id='postgres_default',  # Debes configurar una conexión a PostgreSQL en Airflow
    sql='''
    CREATE TABLE carros (
        Marca VARCHAR(255),
        Placa VARCHAR(255),
        precio VARCHAR(255)
    );
    ''',
    dag=dag
)
```

A manera de ejemplo se ha creado una tabla con 3 columnas especificando el tipo de variable que aceptará en cada registro.

- Insertar 200 registros de manera secuencial

```
insert_records_task = PostgresOperator(
    task_id='insert_records_to_carros',
    postgres_conn_id='postgres_default',  # Debes configurar una conexión a PostgreSQL en Airflow
    sql='''
    INSERT INTO carros (Marca, Placa, precio)
    SELECT
        'Marca ' || id,
        'Placa ' || id,
        'Precio ' || id
    FROM generate_series(1, 200) AS id;
    ''',
    dag=dag
)
```

Una vez conocemos la estructura de nuestra tabla insertamos 200 registros en nuestra tabla para esto especificamos la conexión antes creada y la base de datos.

- Eliminar 50 registros de la tabla antes creada

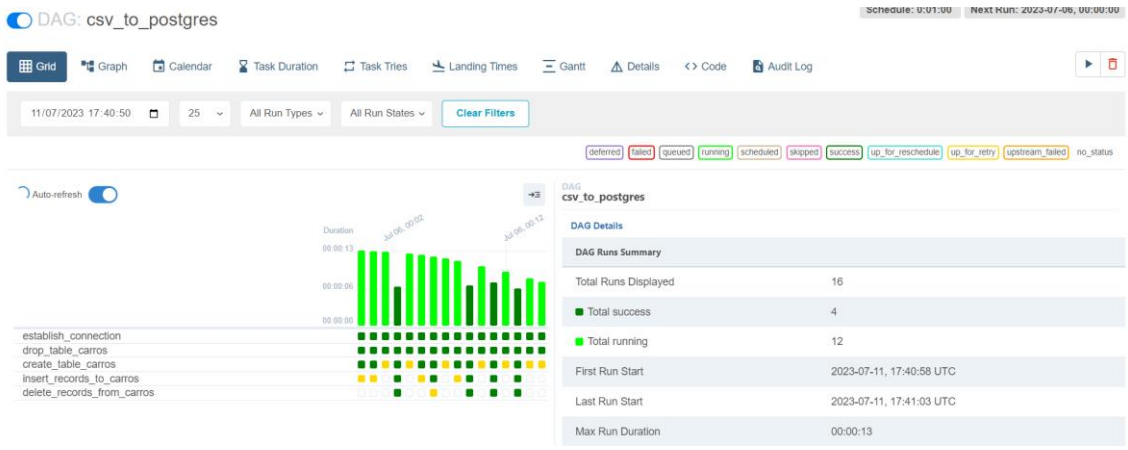
```
delete_records_task = PostgresOperator(
    task_id='delete_records_from_carros',
    postgres_conn_id='postgres_default',  # Debes configurar una conexión a PostgreSQL en Airflow
    sql='DELETE FROM carros WHERE ctid IN (SELECT ctid FROM carros LIMIT 50);',
    dag=dag
)
```

En este trabajo eliminamos 50 registros de forma aleatoria de los 200 previamente agregados.

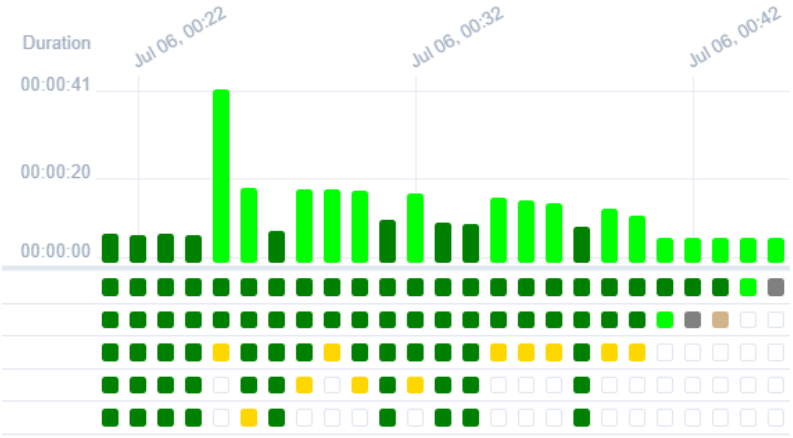
3. Probar el DAG

Una vez nos hayamos asegurado que nuestra configuración y conexión se encuentran configuradas de manera correcta procedemos a testear nuestro DAG en el servidor web de apache airflow, para esto podemos hacer uso de la pestaña GRID la cual nos indica la ejecución de cada tarea y su respectivo estatus.

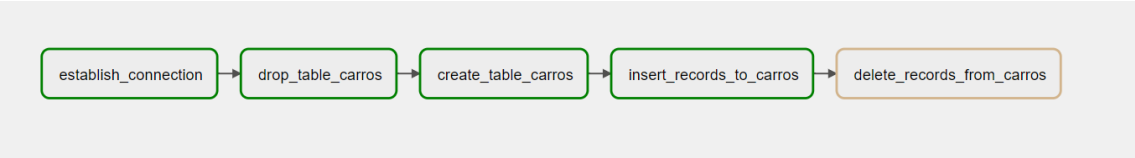
- Prueba en GRID



- Ejecución de trabajos exitosa



- Prueba del proceso en Grafico de procesos



- Comprobación de inserción de registros en tabla

```
electricos=# SELECT * FROM carros;
   marca   |   placa   |   precio
-----------+-----------+------------
 Marca 1   | Placa 1   | Precio 1
 Marca 2   | Placa 2   | Precio 2
 Marca 3   | Placa 3   | Precio 3
 Marca 4   | Placa 4   | Precio 4
 Marca 5   | Placa 5   | Precio 5
 Marca 6   | Placa 6   | Precio 6
 Marca 7   | Placa 7   | Precio 7
 Marca 8   | Placa 8   | Precio 8
 Marca 9   | Placa 9   | Precio 9
 Marca 10  | Placa 10  | Precio 10
 Marca 11  | Placa 11  | Precio 11
 Marca 12  | Placa 12  | Precio 12
 Marca 13  | Placa 13  | Precio 13
 Marca 14  | Placa 14  | Precio 14
 Marca 15  | Placa 15  | Precio 15
 Marca 16  | Placa 16  | Precio 16
 Marca 17  | Placa 17  | Precio 17
 Marca 18  | Placa 18  | Precio 18
 Marca 19  | Placa 19  | Precio 19
 Marca 20  | Placa 20  | Precio 20
 Marca 21  | Placa 21  | Precio 21
 Marca 22  | Placa 22  | Precio 22
 Marca 23  | Placa 23  | Precio 23
 Marca 24  | Placa 24  | Precio 24
 Marca 25  | Placa 25  | Precio 25
 Marca 26  | Placa 26  | Precio 26
 Marca 27  | Placa 27  | Precio 27
```