Exercise 7: Connecting Astronomer Cloud with PostgreSQL

Introduction

PostgreSQL is a popular open-source relational database, and connecting it to Airflow allows for automating database operations. This lab will guide you through configuring a connection to a PostgreSQL database hosted on Render.com, a free managed service, and executing basic operations like creating and dropping tables using the provided Airflow DAG.

Objectives

- Set up a free PostgreSQL database on Render.com.
- Configure a PostgreSQL connection in Astronomer Cloud.
- Deploy and run an Airflow DAG to create and drop a table in the PostgreSQL database.
- Understand the use of the `PostgresOperator` in Airflow.

Preparation

Prerequisites

- Familiarity with Airflow DAGs and task dependencies.
- Basic understanding of PostgreSQL and SQL commands.
- Docker installed and running.
- Python (3.7 or above) and pip installed.
- Active Astronomer Cloud account and workspace.
- Astro CLI installed on your local machine
    - Verify using bash command astro version

Create a Free PostgreSQL Database on Render.com

- Sign up or log in to Render.com.
- Go to New + > PostgreSQL.
- Fill in the details:
- Name: Choose a name for your database.
- Region: Select a region closest to you.
- Free Plan: Ensure you select the free plan.
- Save the database and copy the connection details (host, port, database name, username, password).

Let's Get Started

Part 1: Environment Setup

Step 1. Authenticate Astro CLI:

- Authenticate your CLI with Astronomer Cloud:
  i. Use command astro login
  ii. Press Enter

Step 3. Project setup

- Ensure you have an initialized Airflow project. If not, initialize a new project using:
  i. Create a new directory and navigate into it:
     1. `mkdir astro_cloud`
     2. `cd astro_cloud`
     3. `astro dev init`

Part 2: Creating Postgres Connection in Astronomer Cloud

Step 1. Set Up PostgreSQL Connection in Astronomer Cloud

- Log in to your Astronomer Cloud workspace.
- Navigate to your Deployment and go to the Environment Variables or Connections tab.
- Get connection details form render.com
    i. Go to dashboard
        1. Top right click on connect dropdown
        2. Select External
        3. Copy External Database URL available there in below format

           `postgresql://USER:PASSWORD@EXTERNAL_HOST:PORT/DATABASE`

        4. Use these parament later while creating code
- Add a new connection with the following details:
    - Connection ID: postgress_render
    - Connection Type: Postgres
    - Host: `EXTERNAL_HOST` from Render.com
    - Schema: `DATABASE` name from Render.com
    - Login: `USER` from Render.com
    - Password: `PASSWORD` from Render.com
    - Port: Default is 5432 unless specified otherwise by Render.

  Save the connection.

Install dbeaver on your machine

- Download and install dbeaver exe from https://dbeaver.io/download/
- Open dbeaver
- Go to File -> New > DBeaver -> Database connection ->PostgreSQL.
- Fill in the details: under Main
- Host: `EXTERNAL_HOST` from Render.com
- Database: `DATABASE` name from Render.com
- Login: `USER` from Render.com
- Password: `PASSWORD` from Render.com
- Then Test connection and save

Part 3: Deploying to Astronomer Cloud

Step 1. Add a new Dag

- Navigate to the `dags/` directory and create a file named Postgres_Dag.py

```
from airflow import DAG
from airflow.providers.postgres.operators.postgres import PostgresOperator
from airflow.utils.dates import days_ago
from airflow.utils.dates import timedelta


# Define the default arguments for the DAG
default_args = {
    'owner': 'airflow',
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}


# Define the DAG
with DAG(
    'create_drop_postgres_table',
    default_args=default_args,
    description='A simple DAG to create and drop PostgreSQL table',
    schedule_interval=None,  # Trigger manually or set cron schedule
```

```
    start_date=days_ago(1),
    catchup=False,
) as dag:


    # SQL query to create the table
    create_table_sql = """
    CREATE TABLE IF NOT EXISTS test_table (
        id SERIAL PRIMARY KEY,
        name VARCHAR(100),
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );
    """


    # SQL query to drop the table
    drop_table_sql = "DROP TABLE IF EXISTS test_table;"


    # Task to create the table
    create_table = PostgresOperator(
        task_id='create_table',
        postgres_conn_id='postgress_render',  # Connection ID set in Airflow UI
        sql=create_table_sql,
    )


    # Task to drop the table
```

```
drop_table = PostgresOperator(

    task_id='drop_table',

    postgres_conn_id='postgress_render',  # Connection ID set in Airflow UI

    sql=drop_table_sql,

)



# Set the task dependencies

create_table >> drop_table
```

- Save the file and navigate to the project root.

Step 2. Create deployment

- Use below command to create astor deployment
    i. astro deploy
    ii. It will ask to create a deployment if no deployment is available as of now Provide details like
        1. name of deployment
        2. Cloud region where it need to be deployed
    iii. if deployment is available then select that deployment and proceed with deploy

Step 3. Verify deployment

- Log in to the Astronomer Cloud UI and navigate to your deployment.
- Ensure your DAG appears under the DAGs tab.

Step 4. Monitor and Manage Workflows

- Trigger the `create_drop_postgres_table` DAG from the Astronomer Cloud UI.
- Monitor task execution:
  - **Task 1 (`create_table`):** Creates the `test_table` in the PostgreSQL database.
  - **Task 2 (`drop_table`):** Drops the `test_table`.

  Check the Render.com database logs

  Verify using SQL editor on dbeaver

Step 5. Update deployment

- Modify the DAG to include error for SQL execution.
- Log detailed error messages using Airflow's logging capabilities.
- Check failure logs on Airflow UI

Conclusion

In this lab, you:

- Connected Astronomer Cloud to a PostgreSQL database hosted on Render.com.
- Configured and tested a DAG that performed basic database operations using the `PostgresOperator`.
- Explored key Airflow concepts like connections and task dependencies.

This exercise demonstrates how Airflow can be used to manage database operations programmatically, a critical skill for data engineers.