ELT Pipeline using Snowflake, DBT & Airflow

Tools Used:
DBT - For transformation. DBT(Data Build Tool) is a command line tool that helps in transformation of data in warehouses. DBT does the T(transformation) in the ELT process. It does not extract or load data, but is designed to be performant at transforming data already inside of a warehouse
Snowflake - For Data Warehousing
Airflow - For orchestration

Dataset - TPC-H [Link]

Development (In windows env):

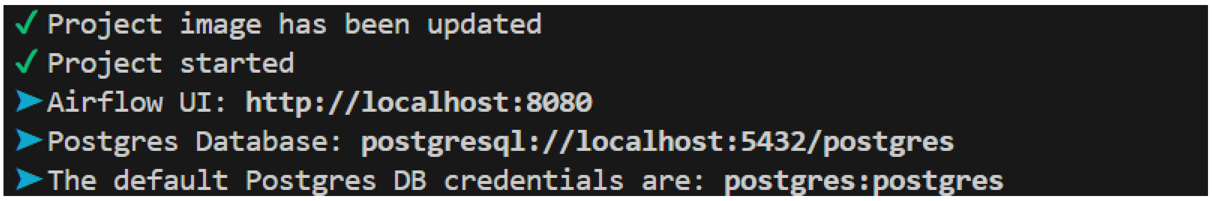
1. Install dbt using pip command [Link]
   ```
   python3 -m venv de_venv
   de_venv\Scripts\activate.bat
   ```
2. Install dbt-snowflake
   pip install dbt-snowflake
   pip install dbt-core
3. Create a warehouse, a database, schema , dbt tables & a role and assign that role to our superuser in snowflake.
4. Setup snowflake environment [ELT Pipeline file]
   Use the comments in file to create DWH, DB & role
5. Initialize dbt project: dbt init
   Choose snowflake , login with your username and password(or with other authenticator as you wish)
   For role, warehouse, db, schema enter the names that we created -(e.g. dbt_role in this case we created in ELT pipeline file)
   For threads say 10
6. Edit the yml file(this yml file tells dbt where to look for your models) with models(here we write our sql logic, source dataset), we added our warehouse , marts(tables) and created folders for the same in modes dir.
7. Create a new yml file(packages.yml) with dbt_utils & latest version
8. Run dbt deps [The `dbt deps` command **installs external dbt packages** listed in our project's `packages.yml` file.]
   Folders descriptions
   dbt-packages: For third party libraries
   seeds: For static files (for example some csv files that is not going to change)
9. Setup source and staging tables

tpch_sources.yml file, stg_tpch_orders.sql, stg_tpch_line_items.sql
After that run command: dbt run -s stg_tpch_line_items (Here s in -s stands for short)

10. Transformed Models(Fact tables, data marts) : Plan is to aggregate some data in line_items table here and create fact tables(that stores quantitative data (facts) and foreign keys that reference dimension tables) as a result.
    Create init_order_items.sql file
    >> dbt run -s init_order_items
11. Create a macro function(these are good way to reuse business logic across multiple models) - pricing.sql
    And other tables inside marts folder, then:
    >> dbt run
12. Test Codes:
    Two types: 1. Singular Tests - Where we write SQL queries that returns failing rows -
    2. Generic Tests - generic_tests.yml, fact_orders_date_valid(This query is designed to **return records with "impossible" or suspicious dates** — either in the **future** or **before 1990**.)
13. Deploy models using Airflow:
    Below is specific for windows platform:
    Download Astro CLI latest version[Link]. Download the exe file , change its name to astro.exe. Add the path to env variable in your system.
    Then run below commands:
    mkdir dbt-dag
    cd dbt-dag
    Astro dev init # It will initialize empty Astro project in your dbt-dag dir
14. Update the Dockerfile with

```
# Creating a new venv for Docker image , not using the local
de_venv
RUN python -m venv dbt_venv && source dbt_venv/bin/activate && \
    pip install --no-cache-dir dbt-snowflake && deactivate
```

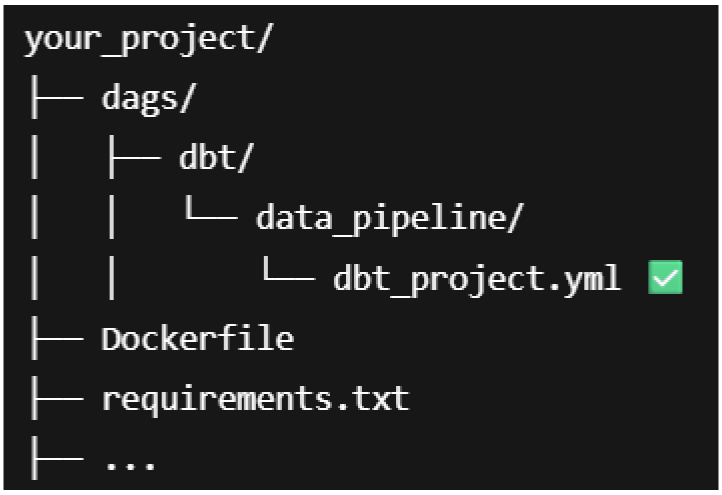
    And requirements.txt file
15. Start Docker desktop and run below command in terminal
    astro dev start
    If it shows below:



```
✓ Project image has been updated
✓ Project started
➤ Airflow UI: http://localhost:8080
➤ Postgres Database: postgresql://localhost:5432/postgres
➤ The default Postgres DB credentials are: postgres:postgres
```

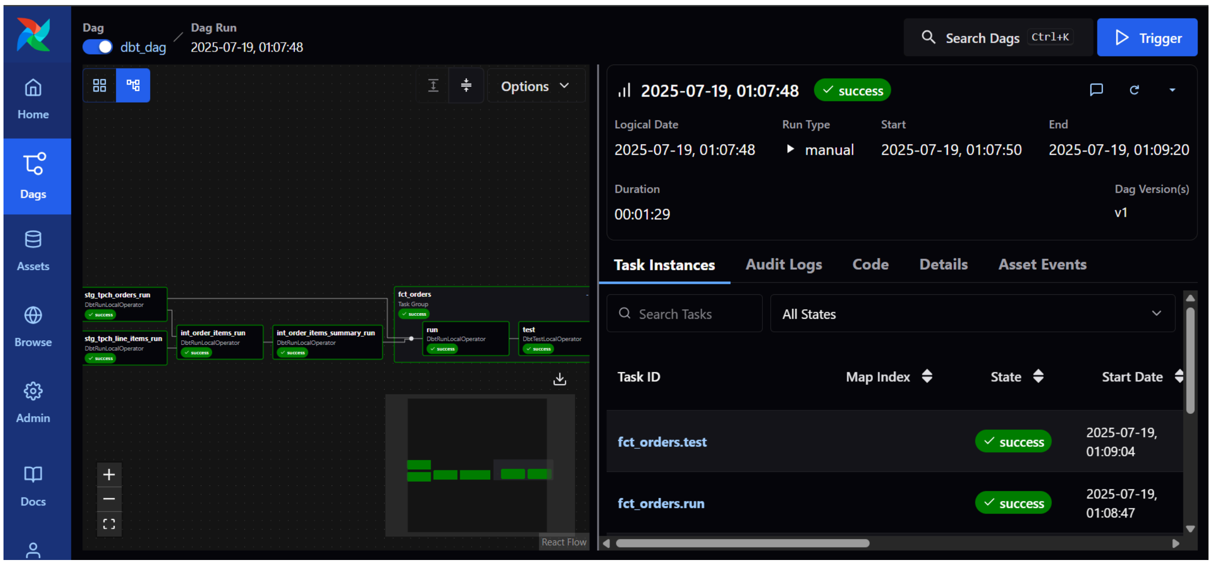
    Hey Man! You Are on right path 🙂 **BELIEVE**

16. Now in order to run dbt on airflow, create a dbt folder inside dbt-dag dir and copy paste the data-pipeline folder to dags/dbt folder since we need to place files in the correct place relative to Astro's expected structure
By default, Astro expects this structure:



```
your_project/
├── dags/
│   ├── dbt/
│   │   └── data_pipeline/
│   │       └── dbt_project.yml ✅
├── Dockerfile
├── requirements.txt
├── ...
```

17. Now DAG should be visible in Airflow
18. Create a new connection with name(snowflake_conn) same as we used in our DAG, use your hostname(e.g. In your browser address bar : https://<account_locator>.<region>.<cloud>.snowflakecomputing.com, The part after `https://` is your **host** e.g. xy12345.ap-south-1.aws.snowflakecomputing.com)
19. Run your DAG



Dag: dbt_dag / Dag Run: 2025-07-19, 01:07:48

2025-07-19, 01:07:48 success

Logical Date	Run Type	Start	End
2025-07-19, 01:07:48	manual	2025-07-19, 01:07:50	2025-07-19, 01:09:20

Duration: 00:01:29
Dag Version(s): v1

Task ID	Map Index	State	Start Date
fct_orders.test		success	2025-07-19, 01:09:04
fct_orders.run		success	2025-07-19, 01:08:47

So it ran, initially with some errors but fixed them.
We have completed the orchestration part as well using Airflow.

Thanks for reading this document 🙂

[Debugging tip - Command:>> dbt debug]