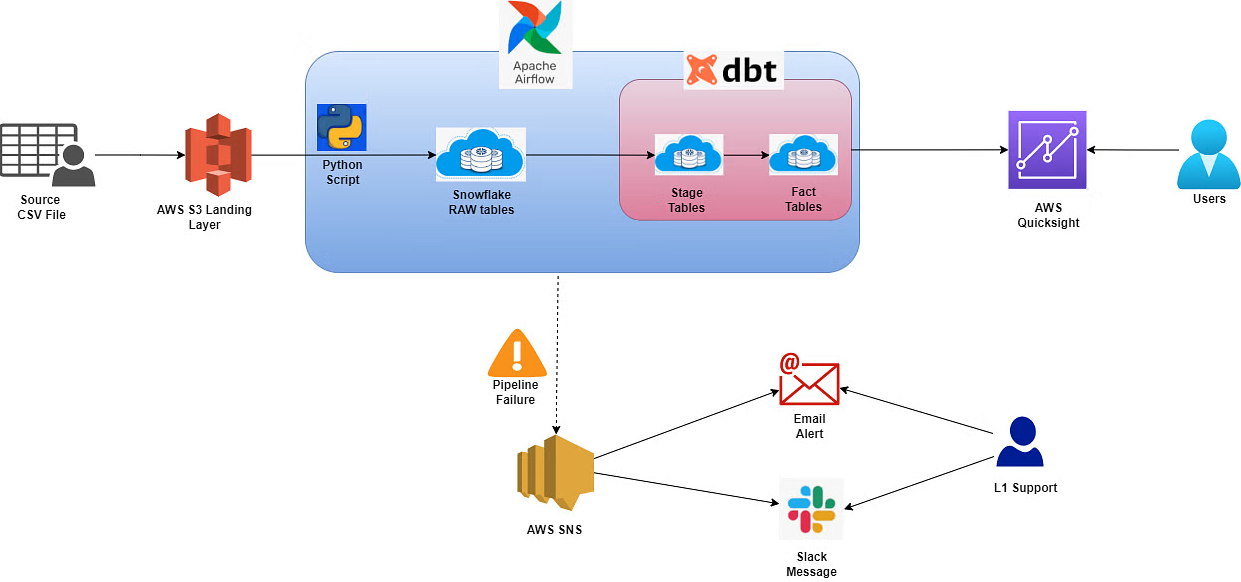
# Netflix Data Pipeline

# Project Overview



**Aim:**

* The main objective of this project is to offer a complete comprehension of the Data Build Tool (DBT). We will explore the fundamental elements of DBT, such as the tool's significance and its role within the Extract, Transform, Load (ETL) process. We will also delve into key aspects like data modeling and logging, which play vital roles in DBT.
* Idea is to construct a robust ETL pipeline utilizing various technologies such as dbt, Snowflake, and Airflow with Container services with docker. By integrating these tools, we aim to create a seamless flow for extracting, transforming, and loading data.
* Furthermore, to ensure efficient monitoring of each pipeline run, we will incorporate Slack and email notifications using SNS (Simple Notification Service). Overall, this project series is designed to provide a comprehensive understanding of DBT, covering its essential concepts and practical implementation through the development of an ETL pipeline that leverages the capabilities of dbt, Snowflake, Airflow, Slack, and email notifications via SNS.

**Tech Stack**

➔ Language: Python, SQL

➔ Tool: DBT

➔ Database: Snowflake

➔ Services: Docker, Airflow

**Step – by – Step Execution**

# Step1: Setting up Environment in Windows PC

We can Execute the Data pipeline in a Linux based system hosted on cloud services like AWS and we have discussed this in separate project. Here we will focus on hosting the Pipeline in our Local Windows PC thus encountering the challenges and building a Robust Data Pipeline.

To start with we need to setup our Arsenal i.e. setting up all the environments and then on top of that we can write the code logics for successful executions

* Install Python
* Create a Project directory and create necessary Modules
* Create a Python virtual environment [ppro\_DBT3]
* Create a requirements.txt file and encompass all the libraries needed inside it
* Setup Airflow with Docker as we can use WSL and have seamless execution
* Create Snowflake and DBT accounts

# Step2: Setting up Airflow with Docker

Very Good documentation to setup this: <https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html>

So to start with we need to install Docker Desktop and have to integrate with WSL. Then make sure if the Docker desktop engine is up and running. Give sufficient space for the CPU consumption as it tends to consume lot of memory

Now, setting up Docker container with Airflow Image the documentation has great explanation however there will be some issues with the setup and will help down with addressing them

* Create a folder airflow\_docker and we will perform all the setps on setting up airflow inside this dir
* We can get the docker-compose.yaml file from

curl -LfO 'https://airflow.apache.org/docs/apache-airflow/2.9.2/docker-compose.yaml'

* With the docker-compose file we can create the containers and verify if it
* The below are the services hosted with Docker for Airflow. We can add any or remove any services based on our requirements of project. Wen we add or remove we should be careful with the configuration in the yaml file
  + 1. **airflow-scheduler** - The scheduler monitors all tasks and DAGs, then triggers the task instances once their dependencies are complete.
    2. **airflow-webserver** - The webserver is available at http://localhost:8080.airflow-worker - The worker that executes the tasks given by the scheduler.
    3. **airflow-triggerer** - The triggerer runs an event loop for deferrable tasks.
    4. **airflow-init** - The initialization service.
    5. **postgres** - The database. We will setup the Mota data for Airflow in Postgres
* The changes I made in the docker-compose.yml file for this project are as follows:
* Removed services like redis and flower as they are not needed
* Made changes to the Executor and SQ\_Alchemy\_Conn

AIRFLOW\_\_CORE\_\_EXECUTOR: LocalExecutor

AIRFLOW\_\_DATABASE\_\_SQL\_ALCHEMY\_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow

AIRFLOW\_\_CORE\_\_SQL\_ALCHEMY\_CONN: postgresql+psycopg2://airflow:airflow@postgres/airflow

* Setup :

AIRFLOW\_\_CORE\_\_LOAD\_EXAMPLES: 'false'

* Mounted a directory for my data files as I will be working with data from local PC and docker and Airflow should be able to identify it

volumes:

./dags:/opt/airflow/dags

./logs:/opt/airflow/logs

./plugins:/opt/airflow/plugins

C:/Users/Akilesh/OneDrive/Desktop/Data\_Projects/Data\_Files:/opt/airflow/data\_files

Note: So docker has its file system which is windows based so here we are having our data Local PC but even if you have data coming from S3 it has to be defined inside the docker file system so that docker can recognize it and work with them. Example in the above compose file if you see the data directory is defined to be (:/opt/airflow/data files). Let us see another example if data is coming in from AWS S3 and you want to house the data you can do like (df.to\_csv(‘/opt/airflow/dags/data.csv”)

* We have defined directories for logs and volumes and we will have to create them along with the dags folder as defined in the compose file

mkdir -p ./dags ./logs ./plugins ./config

* We can test the Docker creation first and then make more changes. We can go to docker desktop and see if the container is up and running also we can go to the webserver and see if it is connecting well and se if we can access the dags: http://localhost:8080/

docker-compose up airflow-init

docker-compose up -d

* We will now stop and bring the container down with the volumes as we are going to make more changes and implement them. Since we are running the DAG with python and **we are adding dependencies via requirements.txt file** like the python libraries installations for this we will **create a new docker image** by making changes to the image version we have copied from the apache site. This can be achieved by creating a Dockerfile thus we will define the changes in the docker file give name for the Docker image and update the image in the docker-compose.yml file. Now we need to build the new image and run up the docker-compose again for all to sync in
* Dockerfile

FROM apache/airflow:2.5.1

COPY requirement.txt /requirement.txt

RUN pip install --user --upgrade pip

RUN pip install --no-cache-dir --user -r /requirement.txt

* Docker build the image

docker build . --tag extend\_airflow:latest

* We need to update this Docker image in place of our old docker image in the compose file

version: '3'

x-airflow-common:

  &airflow-common

  image: ${AIRFLOW\_IMAGE\_NAME:-extended\_airflow:latest}

  environment:

    &airflow-common-env

* Now we can use this docker-compose.yml file to initiate the new containers [Note: whenever we make changes to the Dockerfile and the Compose file we need to bring the containers down with the volumes down and the again initiate them up]

docker-compose up airflow-init

docker-compose up -d

Now you should be all set with Docker-Airflow and you can create and run the DAGs from the webserver (<http://localhost:8080>)

# Step3: Setting up DBT-Snowflake

* Install **DBT-<Data warehouse>** with the pip. So we are using Snowflake so I am installing

pip install dbt-snowflake

* Now in we are using **DBT core** instead of **DBT cloud** as it is open source. So to setup DBT core make sure to install Python and DBT power user extensions for ease of work
* Create your **snowflake account** if you don’t have one and note down the credentials
* Now there are Two things you can do. If you have a Code repository already you can Clone it (ex: Cloning a Repo from Github) and the Second option Setup the whole DBT code lineage from start with a pure Vanilla setup. We will do the setup from start
* So the first step is to setup a **profiles.yml** file. It is recommended to setup this in the root directory else you have to make extra efforts to point to the profiles.yml file at various parts of the code. So we are creating profiles.yml file inside root directory **(~/.dbt/profiles.yml).** you can setup any number of profiles in this. Like you can setup any number of accounts with their credentials – Mapped to the project you are working. So in our case we have setup our **Snowflake Credentials – Project Name (ppro\_netflix\_project).** This is the environment var setup so any project created anywhere would point to this profiles.yml file
* ppro\_netflix\_project:
* target: dev
* outputs:
* dev:
* type: snowflake
* account: MOHHJKJ-EOB44343
* user: mohanraj
* password: \*\*\*\*
* role: ACCOUNTADMIN
* database: ppro\_airflow\_db
* warehouse: COMPUTE\_WH
* schema: ppro\_airflow\_schema
* threads: 4
* Now on the directory where we are running the project we have to Initialize DBT. This will create all the basic folder and files needed to run DBT. It will also create dbt\_project.yml file when we initialize DBT and ensure it is correctly configured. Dbt\_project.yml file is also the master configuration for setting up the project, models, lineage, etc of DBT project

cd C:\Users\Akilesh\OneDrive\Desktop\Project\_Pro\Big\_Data\DBT\P3\_DBT\_Snowflake\_Airflow\_Netflix\_Project\airflow\_docker\ppro\_netflix\_project

dbt init ppro\_netflix\_project

To make sure if connection is fine and the dbt\_project.yml and profiles.yml file is correctly configured and are in sync run

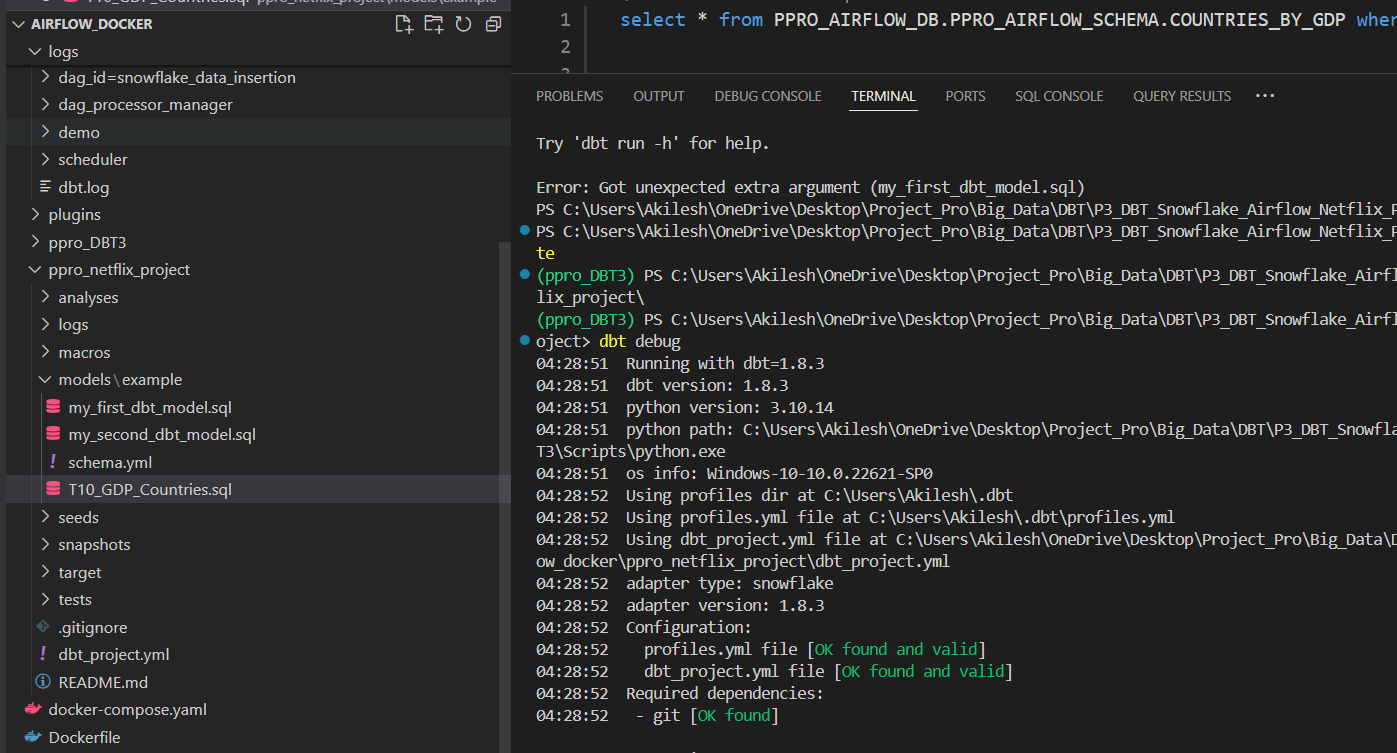
dbt debug

if all are fine we will get

04:28:53    Connection test: [OK connection ok]

04:28:53  All checks passed!

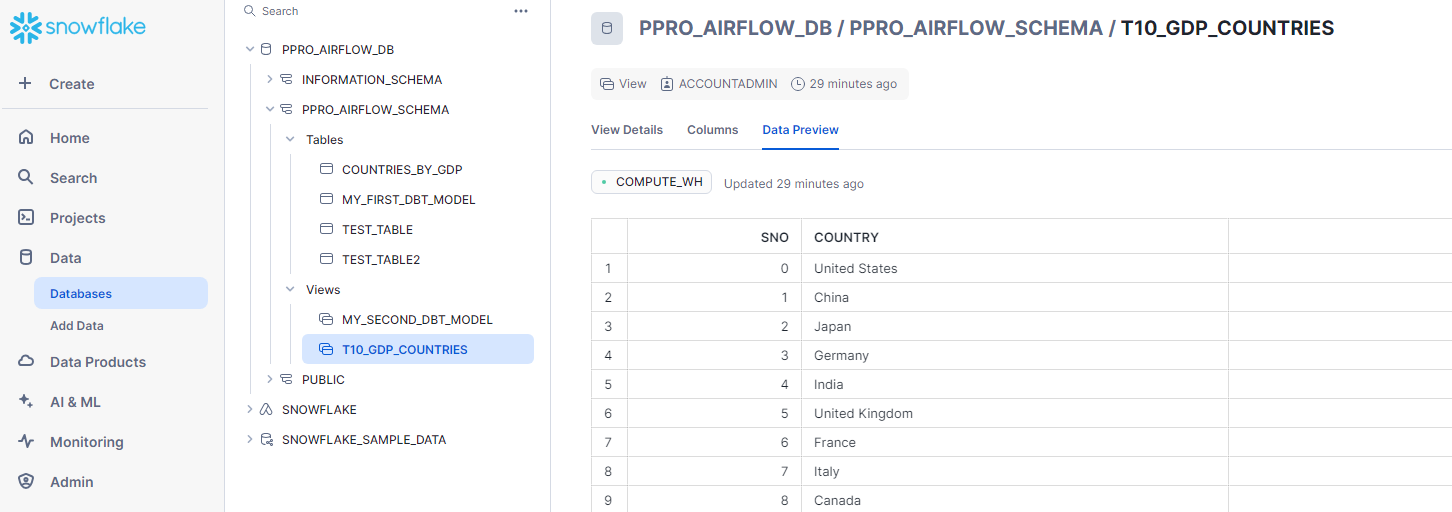
You can also see all the Folders created and setup when we Initialized the DBT project



* So now connection is successful and we can Create and run the DBT models as per our Business requirements.

dbt run --model T10\_GDP\_Countries.sql

The Tables, Views will be materialized in Snowflake



# Step4: DBT: Data Modeling

Now we will solve Business cases with the various Data modelling in addressing the below Questions by creating Tables, views that the downstream teams will use for Analysis.

During the process of doing this we will build the DBT project like setting up Seeds, Source, Python Jinja templates, Tags….etc

**Business Requirement:**

**1. Popularity Dimension (popularity\_dim.sql)**

**Business Requirement:**

**Objective:** To create a comprehensive dimension that combines detailed information about shows with their popularity metrics from various sources (IMDB, TMDB). This will enable the downstream data analysis team to perform an in-depth analysis of the popularity trends and ratings of different shows and movies.

**Purpose:**

* **Trend Analysis:** Analyze trends in show popularity across different platforms.
* **Correlation Analysis:** Determine the correlation between IMDB scores and TMDB popularity.
* **Market Analysis:** Assess which shows are most popular and identify potential factors contributing to their popularity.

**2. Actors Dominating Fact (actors\_dominating\_fact.sql)**

**Business Requirement:**

**Objective:** To create a fact table that captures the performance metrics of actors in various genres, allowing the downstream data analysis team to identify actors dominating specific genres.

**Purpose:**

* **Performance Metrics:** Track and analyze the number of performances by actors across different genres.
* **Actor Analysis:** Identify actors who are most active and successful in specific genres.
* **Genre Popularity:** Understand which genres have the most actor performances, helping in talent acquisition and casting decisions.

**3. Movies vs. Series Share (movies\_series\_share.sql)**

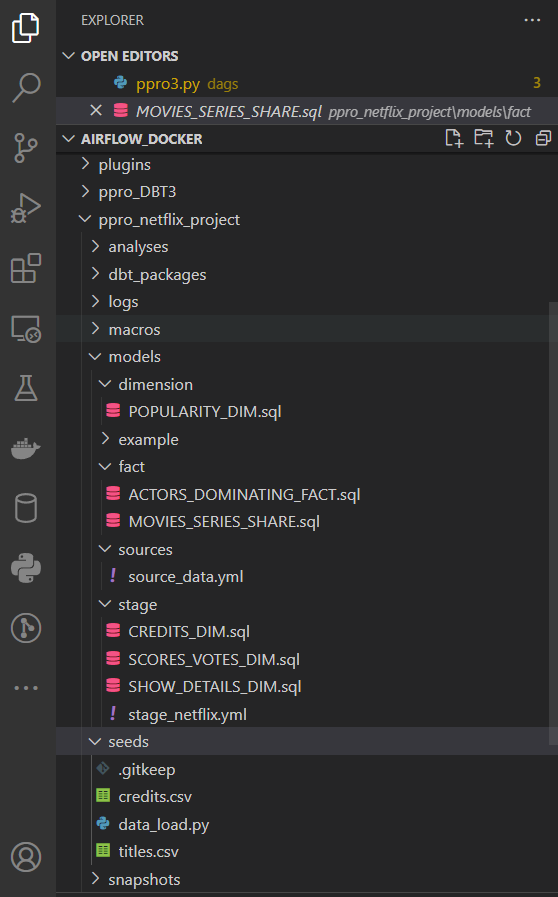
**Business Requirement:**

**Objective:** To calculate and provide the share percentage of movies and series within each genre, enabling the downstream data analysis team to understand the distribution of content types across genres.

**Purpose:**

* **Content Distribution:** Analyze the distribution of movies and series across different genres.
* **Strategic Decisions:** Make informed decisions regarding content production and acquisition strategies based on the genre-wise share of movies and series.
* **Audience Preferences:** Understand audience preferences for movies versus series within specific genres, helping to tailor marketing and promotional efforts.

**Models folder structure:**



# Step5: Adding DBT Models to Airflow DAG

We have successfully Run and Tested our DBT models and now we will have to add the DBT task to our Airflow DAG. Since we are using Airflow in Docker which is used using WSL it follows a Linux directory system and so we will have to first mount the DBT folders.

We have to define that in the docker-compose.yml file and rerun.

  volumes:

    - ./dags:/opt/airflow/dags

    - ./logs:/opt/airflow/logs

    - ./plugins:/opt/airflow/plugins

    - C:/Users/Akilesh/OneDrive/Desktop/Data\_Projects/Data\_Files:/opt/airflow/data\_files

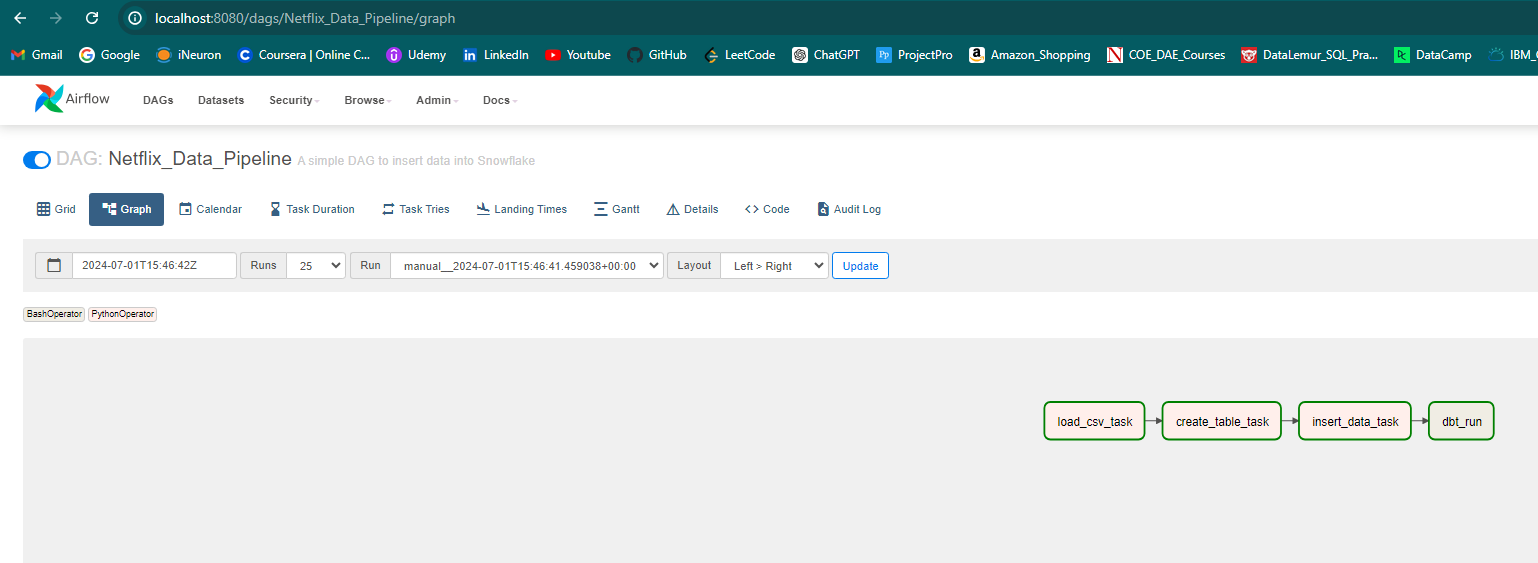
    - C:/Users/Akilesh/OneDrive/Desktop/Project\_Pro/Big\_Data/DBT/P3\_DBT\_Snowflake\_Airflow\_Netflix\_Project/airflow\_docker/ppro\_netflix\_project:/opt/airflow/dbt/netflix\_project

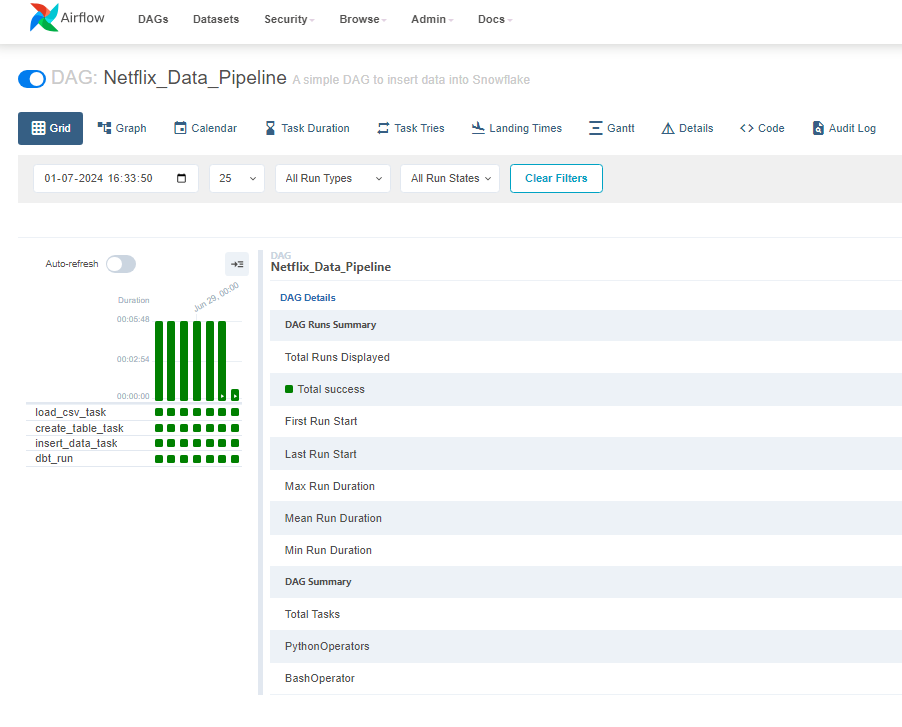
    - C:/Users/Akilesh/.dbt:/home/airflow/.dbt  # Assuming profiles.yml is here

Now we will use this Path (/opt/airflow/dbt/Netflix\_project) in our DAG for DBT task as all the files and dependencies will be mounted there

The DAG we are using is **ppro3.py** with Name: **Netflix\_Data\_Pipeline.**

Let us Now Trigger the DAG and see if all the tasks are executing properly





Thus we can see that all the Tasks worked perfectly fine

# Step6: Error and Success handling

* Once the Airflow DAG is successfully executed, enhancing the project by implementing error and success handling can add significant value. This step involves setting up notifications to alert you about the status of the DAG executions, providing immediate feedback on the workflow's health.
* To achieve this, you can integrate communication tools such as Slack or email. By configuring Airflow to send messages, you can ensure that both errors and successful runs are promptly reported. This involves setting up Airflow hooks and operators for your chosen communication medium.
* For error handling, you can define tasks within the DAG that trigger alerts if a task fails. These alerts can include detailed error messages, helping you quickly identify and resolve issues. Similarly, for success handling, you can set up notifications that inform you when the entire DAG or critical tasks within it complete successfully.
* This proactive approach to monitoring not only keeps you informed but also helps maintain the reliability and efficiency of your data pipelines. By leveraging Airflow's notification capabilities, you can create a more responsive and robust data processing environment, enhancing the overall effectiveness of your project.

Let us see the code for this:

pip install slack\_sdk

from airflow import DAG

from airflow.operators.python\_operator import PythonOperator

from airflow.operators.dummy\_operator import DummyOperator

from airflow.utils.dates import days\_ago

from airflow.utils.email import send\_email

from airflow.hooks.base\_hook import BaseHook

import os

from slack\_sdk import WebClient

from slack\_sdk.errors import SlackApiError

def send\_slack\_message(message, channel='#general'):

    client = WebClient(token=os.environ['SLACK\_API\_TOKEN'])

    try:

        response = client.chat\_postMessage(

            channel=channel,

            text=message

        )

    except SlackApiError as e:

        print(f"Error sending message: {e.response['error']}")

def task\_success(context):

    send\_slack\_message("DAG succeeded!", "#your-channel")

def task\_failure(context):

    send\_slack\_message("DAG failed!", "#your-channel")

default\_args = {

    'owner': 'airflow',

    'start\_date': days\_ago(1),

    'email\_on\_failure': False,

    'email\_on\_retry': False,

    'on\_success\_callback': task\_success,

    'on\_failure\_callback': task\_failure,

}

with DAG(

    'dag\_with\_slack\_notifications',

    default\_args=default\_args,

    description='A simple DAG with Slack notifications',

    schedule\_interval='@daily',

) as dag:

    start\_task = DummyOperator(

        task\_id='start'

    )

    end\_task = DummyOperator(

        task\_id='end'

    )

    start\_task >> end\_task

# Conclusion

Analytics engineers will find immense satisfaction in the successful execution of this project for several reasons:

The Netflix Data Pipeline project has been successfully executed, showcasing a robust data processing framework built within a Docker container. The pipeline integrates several powerful tools: Apache Airflow for orchestration, DBT (Data Build Tool) for data transformation, Snowflake for data storage, and Slack for error handling and notifications.

## Key Components

1. **Docker Containerization:**
   * The entire pipeline is encapsulated within a Docker container, ensuring consistency and reproducibility across different environments.
2. **Airflow for Orchestration:**
   * Apache Airflow orchestrates the entire workflow, managing the scheduling and execution of tasks seamlessly.
   * Tasks are defined within Airflow DAGs, enabling clear and maintainable workflows.
3. **DBT for Data Transformation:**
   * DBT is utilized to transform raw data into a structured format, ready for analysis.
   * Transformations are version-controlled and documented, ensuring transparency and reproducibility.
4. **Snowflake for Data Storage:**
   * Transformed data is stored in Snowflake, a scalable and efficient cloud-based data warehousing solution.
   * Snowflake's capabilities ensure fast query performance and secure data storage.
5. **Slack for Error Handling:**
   * Integration with Slack enables real-time notifications for both successes and failures within the pipeline.
   * This ensures prompt attention to issues, maintaining the reliability of the data pipeline.

## Utilization of Data

1. **Data Analysis Team:**
   * The Data Analysis team will leverage the processed data stored in Snowflake to create comprehensive BI dashboards.
   * These dashboards will provide valuable insights into various aspects of Netflix's operations, supporting data-driven decision-making.
2. **Support Teams:**
   * Support teams will use the Slack error notifications to monitor and address any issues that arise within the data pipeline.
   * This proactive approach ensures minimal downtime and quick resolution of problems, maintaining the integrity and availability of data.