**SSMS to Snowflake Real-World Data Pipeline**

**📄 Project Title**

**Building a Real-World Data Pipeline: SQL Server (SSMS) to Snowflake + Airflow + DBT Cloud**

**🔍 Project Overview**

In this project, we build a real-world data pipeline that:

* Extracts data from **SQL Server (SSMS)** using **Python**.
* Loads data into **Snowflake** (RAW layer).
* Orchestrates ETL using **Apache Airflow** (Dockerized).
* Triggers transformations using **DBT Cloud** automatically from Airflow.
* Builds fact tables for analysis.
* Implements full audit logging and error handling.
* Follows best practices: Parameterization, Secrets Handling, Incremental Loads, Github Version Control.

**📈 Architecture Diagram**

[Static PNG and Animated GIF separately provided]

**Flow:**

SQL Server (SSMS)

⬇️

Python Script (Extract)

⬇️

Snowflake (RAW Layer)

⬇️

Airflow (Trigger DBT)

⬇️

DBT Cloud (Transformations)

⬇️

Final Fact Table

**🔗 Technologies Used**

* Microsoft SQL Server (SSMS)
* Python (pyodbc, pandas, snowflake-connector)
* Snowflake Data Warehouse
* Apache Airflow (Dockerized)
* DBT Cloud
* GitHub (Version Control)

**📁 Folder Structure**

src/

extract.py

load.py

audit.py

dbt\_trigger.py

config\_pipeline/

sqlserver\_config.json

snowflake\_config.json

dags/

pipeline\_dag.py

models/

staging/

schema.yml

stg\_players.sql

stg\_matches.sql

stg\_player\_stats.sql

facts/

fct\_performance.sql

.gitignore

dbt\_project.yml

README.md

**🔍 Project Setup Steps (Summary)**

1. **Setup SQL Server**
   * Create sample data tables.
2. **Python Scripts:**
   * extract.py: Extract data from SQL Server.
   * load.py: Load into Snowflake (with create-if-not-exists logic).
   * audit.py: Insert load audit logs.
   * dbt\_trigger.py: Trigger DBT job using API call.
3. **Create Airflow DAG:**
   * BashOperators for extract, load, audit.
   * SimpleHttpOperator to trigger DBT Cloud job.
4. **Dockerize Airflow:**
   * Set up airflow-scheduler, airflow-webserver, redis, postgres.
5. **Setup DBT Cloud Project:**
   * Create DBT models (staging, facts).
   * Add correct source configuration.
6. **Configure GitHub:**
   * Initialize repo.
   * Push code and maintain versions.

**📉 DBT Models Overview**

| **Model** | **Purpose** |
| --- | --- |
| stg\_players.sql | Clean player base data |
| stg\_matches.sql | Clean matches info |
| stg\_player\_stats.sql | Clean player performance stats |
| fct\_performance.sql | Fact table combining match and player stats |

**🏢 Airflow DAG Overview**

| **Task** | **Description** |
| --- | --- |
| extract\_data | BashOperator calling extract.py |
| load\_data | BashOperator calling load.py |
| insert\_audit\_log | BashOperator calling audit.py |
| trigger\_dbt\_job | SimpleHttpOperator calling DBT Cloud |

**📈 Best Practices Followed**

* Parameterized Connections via JSON Configs.
* Separate Configs for Airflow and Source Systems.
* Proper Folder Structure (src, dags, models, config\_pipeline).
* Airflow Secrets Handling for DBT Token.
* Audit Logging Table in Snowflake.
* GitHub Integration for Version Control.
* dbt Cloud Job Orchestration from Airflow.
* Incremental and Full Load Support.

**🚀 Conclusion**

✨ Vishwas Singh successfully built an end-to-end production-grade data pipeline with full automation, transformations, audit tracking, and CI/CD practices integrated!

💼 This project is **perfect to showcase** for real-world Data Engineer roles and demonstrate hands-on expertise!

**✨ Next Uploads:**

* GitHub Push Commands Guide
* LinkedIn Post Draft
* Challenges + Solutions Document
* Interview Questions Document

Stay tuned! 🚀