Breweries Case

VERSIONING

Versão	Data	Autor	Descrição
1.0	11/11/2024	Bhrenner Wilson Marcelino	Document Creation

Sumário

Objective

The goal of this test is consuming data from an API, transforming and persisting it into a data lake following the medallion architecture with three layers: raw data, curated data partitioned by location, and an analytical aggregated layer.

Source

The Source for this test is an api that can be consumed via the endpoint: <api.openbrewerydb.org/breweries> . In this api there is data about breweries of different types and different locations, the file is in json format and has the following schema:

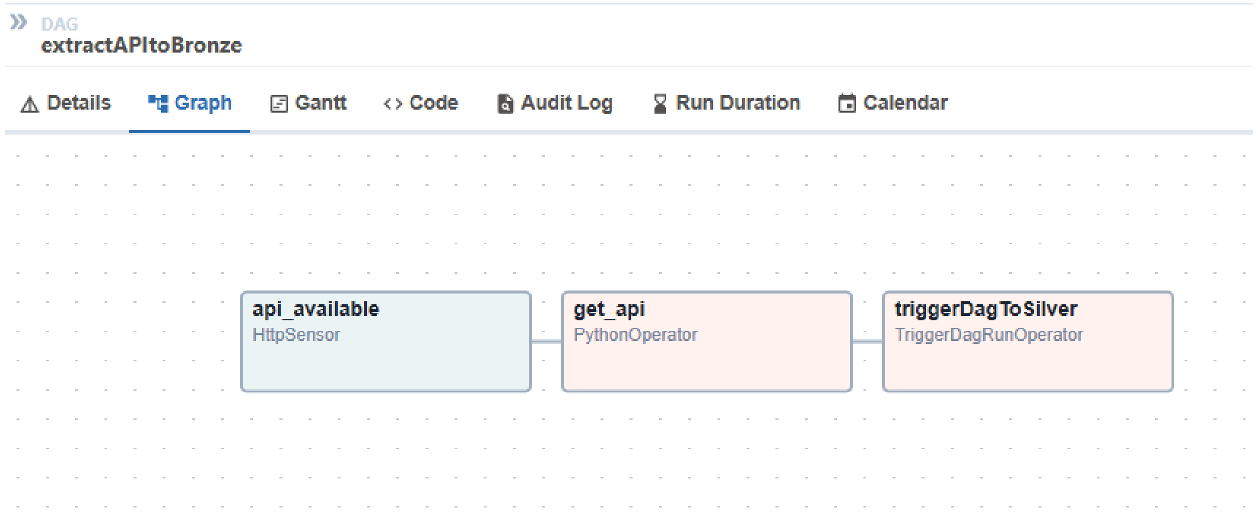
```
json_schema = {
        "id": str,
        "name": str,
        "brewery_type": str,
        "address_1": str,
        "address_2": str,
        "address_3": str,
        "city": str,
        "state_province": str,
        "postal_code": str,
        "country": str,
        "longitude": str,
        "latitude": str,
        "phone": str,
        "website_url": str,
        "state": str,
        "street": str
}
```

Solution

To complete the case, the solution was developed in 3 different steps, each step is an airflow dag.

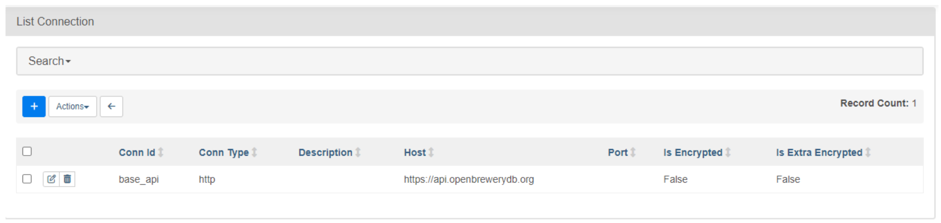
1. **ExtracttoBronze**

   This step was designed to read data from the API using a connection and variables in Airflow, utilizing the HTTP sensor to validate. It transforms this data into a DataFrame using Pandas and writes it to the "bronze" layer of the data lake, preserving the original schema and JSON format properties.



1.1. Connection

Connections in Airflow are sets of configurations used to connect with other tools in the data ecosystem. This connection, called base_api, is used to validate and connect to the API. It works together with the variable endpoint_api, which is a parameter that defines the full path to access the data.


List Connection

Search

Actions

Record Count: 1

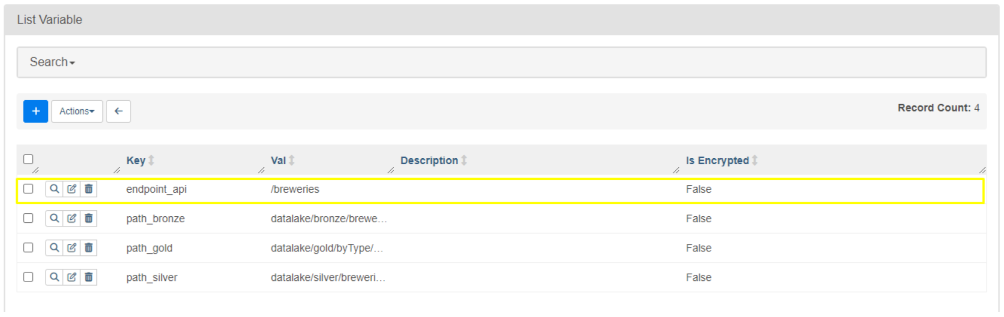
Conn Id	Conn Type	Description	Host	Port	Is Encrypted	Is Extra Encrypted
base_api	http		https://api.openbrewerydb.org		False	False

1.2. Variable

Variables are a generic way to store and retrieve arbitrary content or settings as a simple key value store within Airflow.

1.2.1. End point

In this case, the variable is used to connect to the API to consume the data.

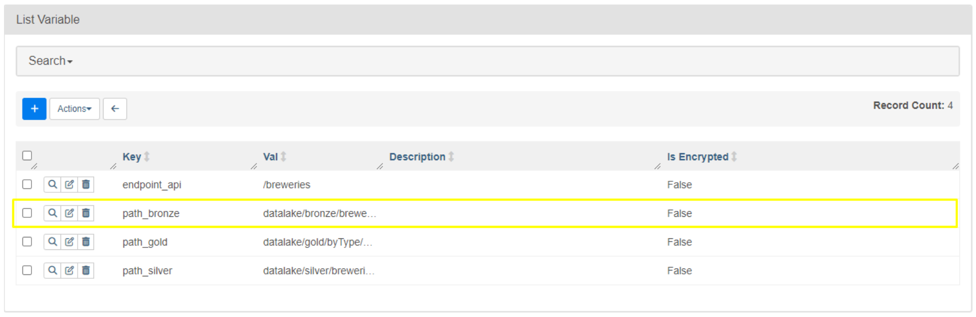

List Variable

Record Count: 4

Key	Val	Description	Is Encrypted
endpoint_api	/breweries		False
path_bronze	datalake/bronze/brewe…		False
path_gold	datalake/gold/byType/…		False
path_silver	datalake/silver/breweri…		False

1.2.2. File path layer

For the next process to consume the generated file in the lake, a variable is set in Airflow using the same file path where the data was saved. This dynamically assists the process in consuming the file on the correct date.

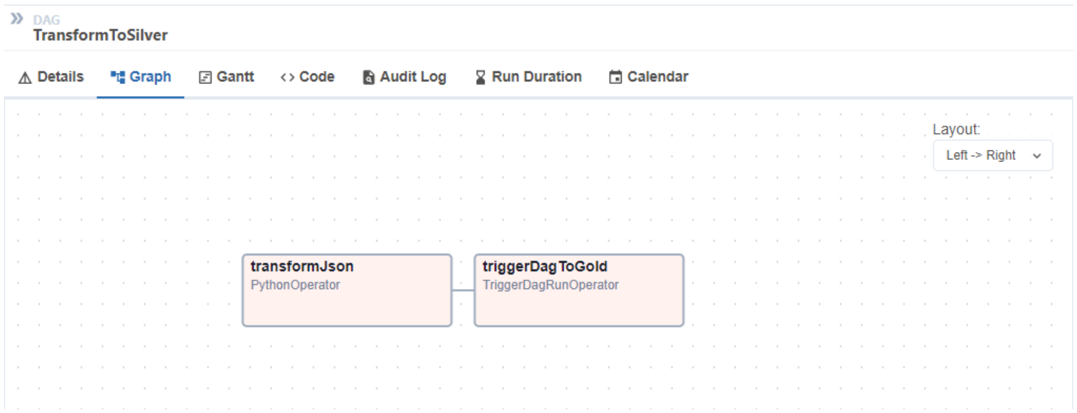

List Variable

Record Count: 4

Key	Val	Description	Is Encrypted
endpoint_api	/breweries		False
path_bronze	datalake/bronze/brewe…		False
path_gold	datalake/gold/byType/…		False
path_silver	datalake/silver/breweri…		False

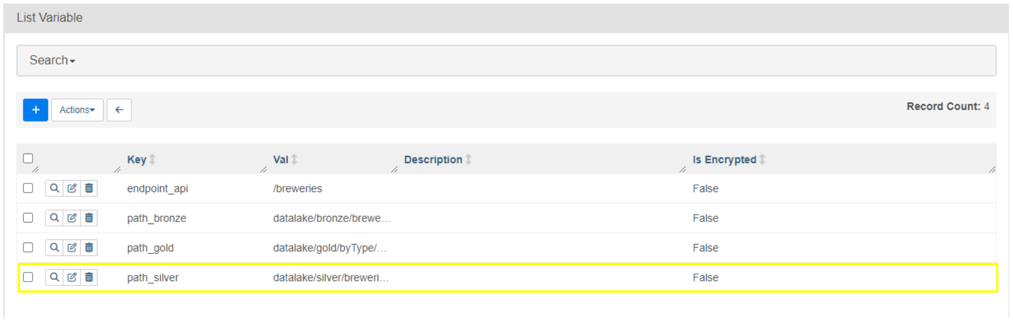
2. **TransformToSilver**

After completing the previous step, the data is read from the bronze layer transformed into a dataframe by pandas, applying a fixed schema. For partitioning issues, a column called "process_date" was added indicating the date on which the data was processed. After these transformations, the dataframe is converted to parquet type and saved in the "silver" layer, partitioned respectively by: "process_date", "country", "state" and "city".



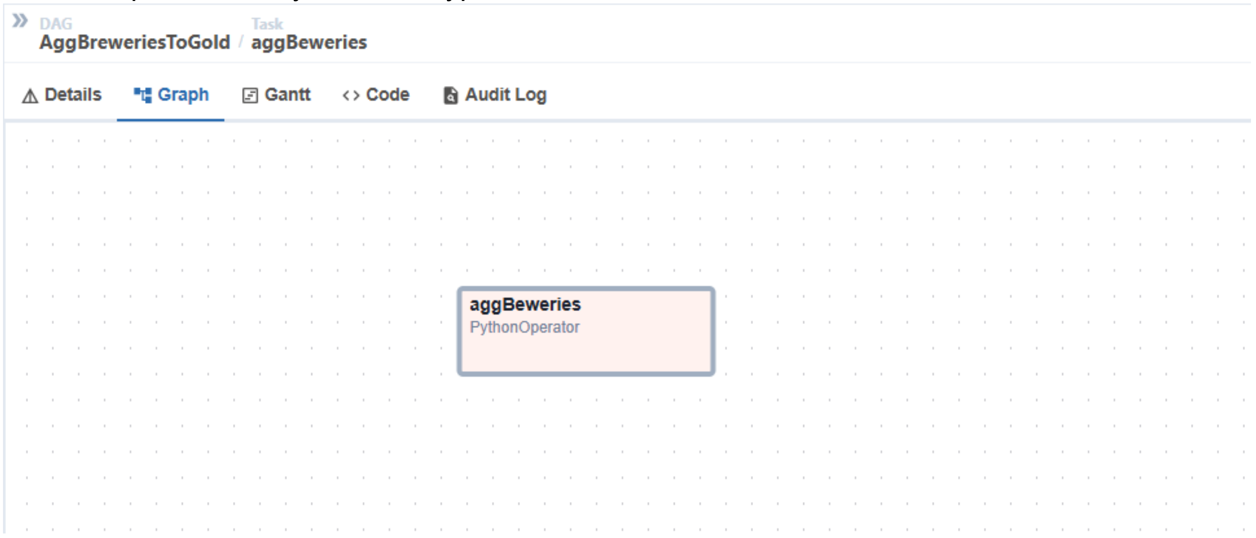
2.1. File path layer

For the next process to consume the generated file in the lake, a variable is set in Airflow using the same file path where the data was saved. This dynamically assists the process in consuming the file on the correct date.



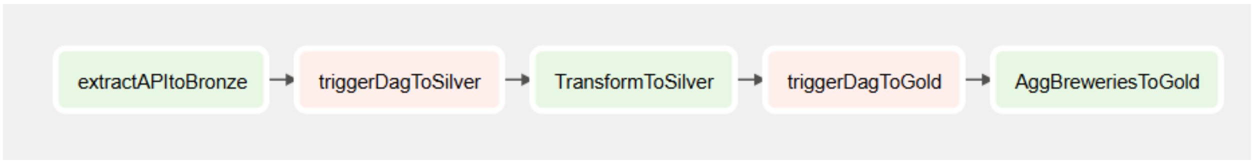
Key	Val	Description	Is Encrypted
endpoint_api	/breweries		False
path_bronze	datalake/bronze/brewe...		False
path_gold	datalake/gold/byType/...		False
path_silver	datalake/silver/breweri...		False

3. **AggBreweriesToGold**
   In this last step, the data in parquet format is read from the silver layer by pyarrow, pandas and the view construction process begins. The view called “BreweriesbyLocType” shows the number of breweries aggregated by type and location.
   After the transformations, the data is saved in parquet format in the “gold” layer of the datalake, partitioned by date and type of view.



Orchestration

The process is scheduled to run once a day. The first process starts and when completed the second process starts and so on.

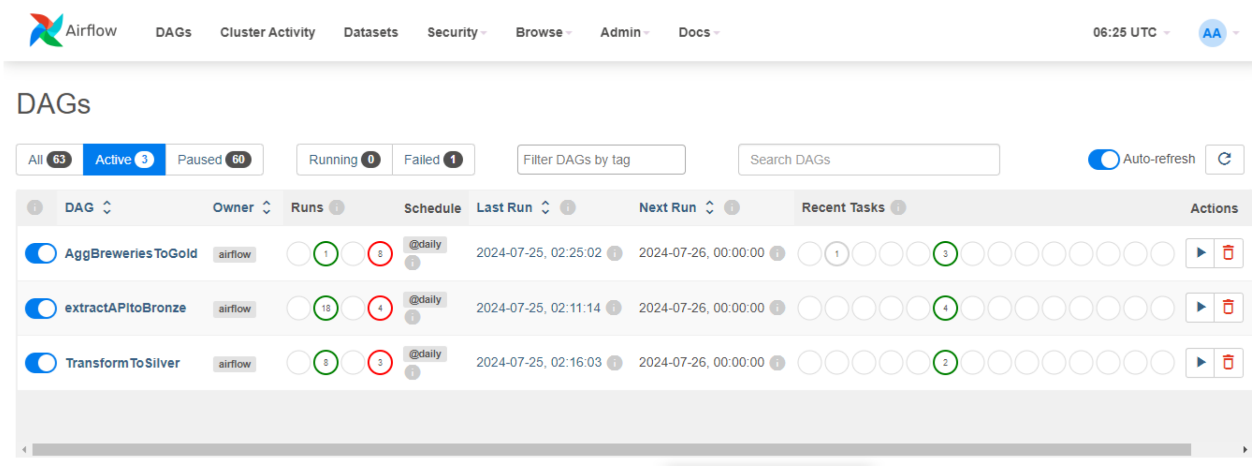


Monitoring and Alerting

In a future would be to make a report on days when there is a problem with the API and we were unable to read its data, when the day's folder in datalake has no records, checks the status of each DAG on failure and retries, it would send an email alert to the developers.

How to execute

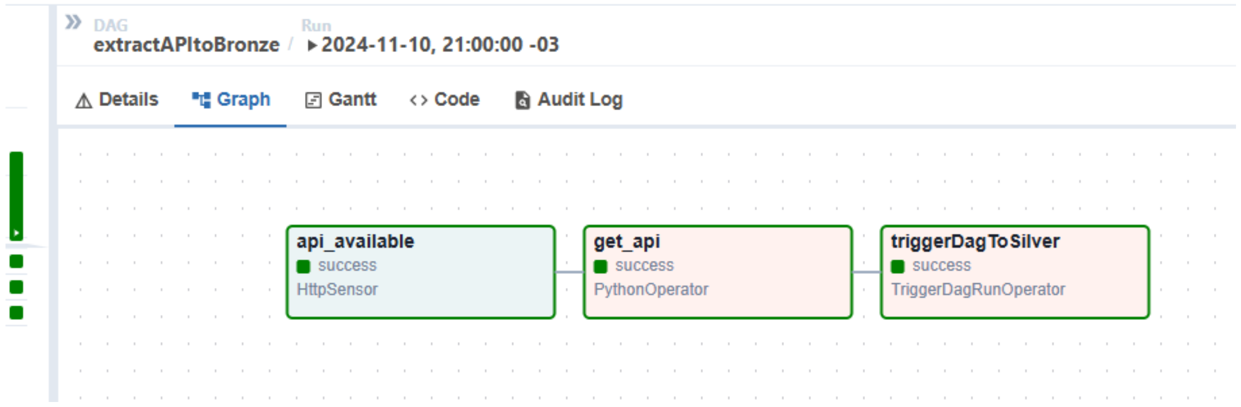
1. First of all it's necessary complete the steps in: **docs/ Setup.docx**
2. After starting the environment, there will be a panel called "Dags", you must enable the following dags: **ExtracttoBronze, TransformToSilver, AggBreweriesToGold.**
   With **just these three** dags active, your "active" panel should look like this:



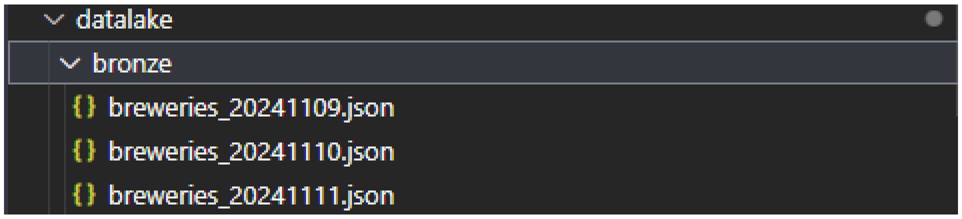
DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions
AggBreweriesToGold	airflow	1 8	@daily	2024-07-25, 02:25:02	2024-07-26, 00:00:00	1 3	
extractAPItoBronze	airflow	18 4	@daily	2024-07-25, 02:11:14	2024-07-26, 00:00:00	4	
TransformToSilver	airflow	8 3	@daily	2024-07-25, 02:16:03	2024-07-26, 00:00:00	2	

3. After activation, you need to set the connections parameters and endpoint variables to connect with the API and the you can wait for the scheduled time in airflow or run the dag manually, just click on the **ExtracttoBronze** Run icon. This dag when executed will automatically execute the next dagPartition**.**

   Clicking on **ExtracttoBronze**, you will be able to view a panel with the status of each task, as shown below:



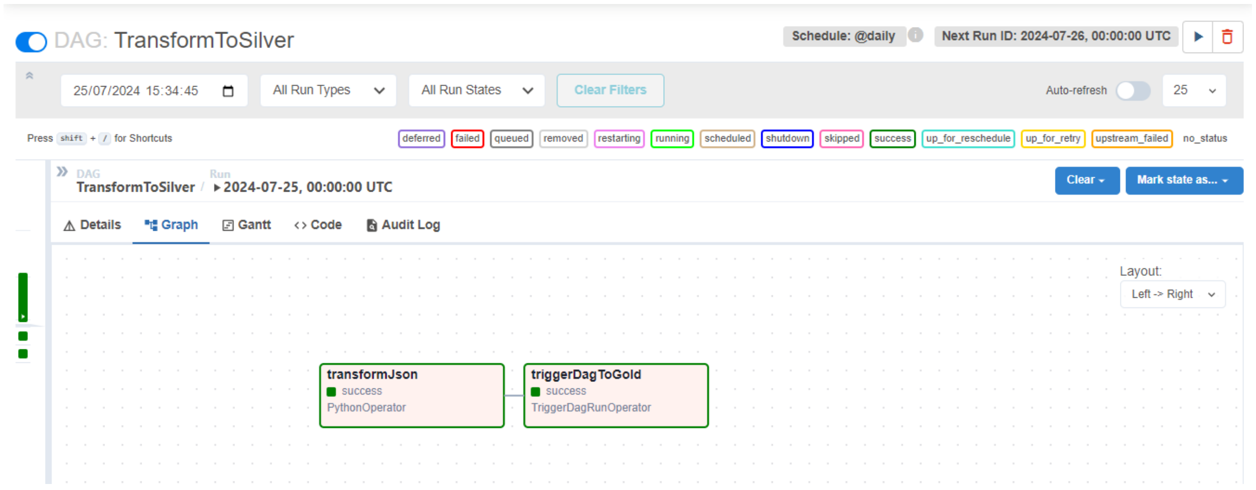
The result in the datalake should be the following in case of success:


```
datalake
  bronze
    breweries_20241109.json
    breweries_20241110.json
    breweries_20241111.json
```

4. After the first step is completed, just check if the other dags executed automatically after the first.

   The results should be as follows:

   Sucess on **TransformToSilver**

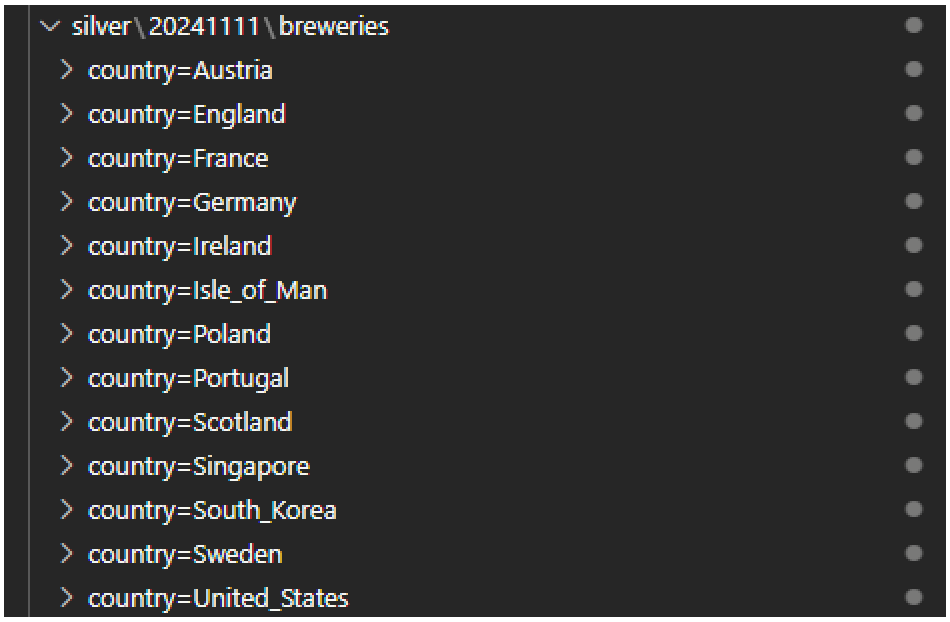

DAG: TransformToSilver

Schedule: @daily — Next Run ID: 2024-07-26, 00:00:00 UTC

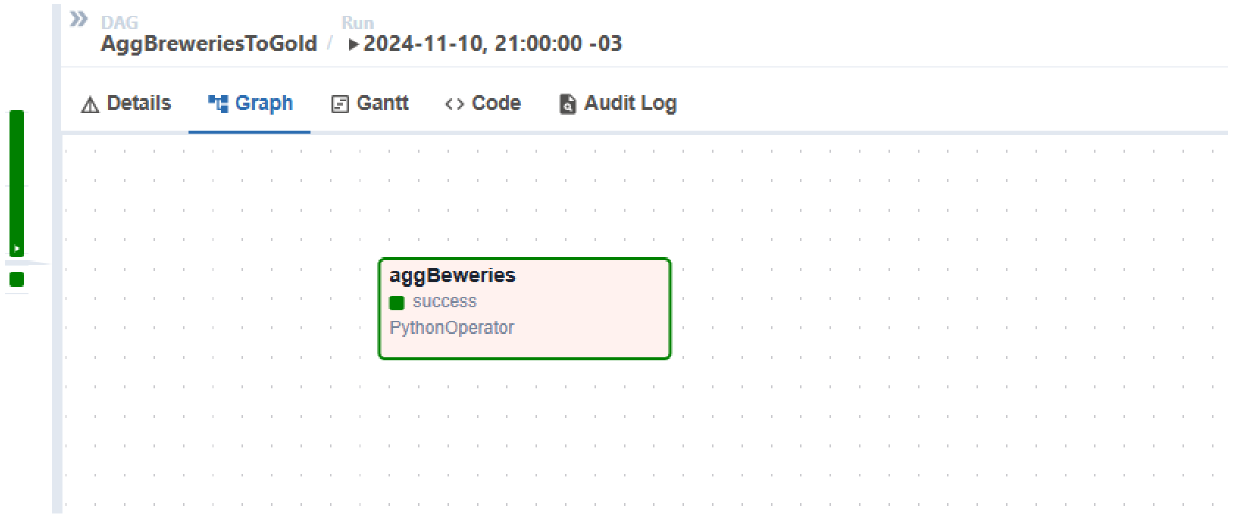
25/07/2024 15:34:45 — All Run Types — All Run States — Clear Filters — Auto-refresh — 25

DAG TransformToSilver / Run 2024-07-25, 00:00:00 UTC

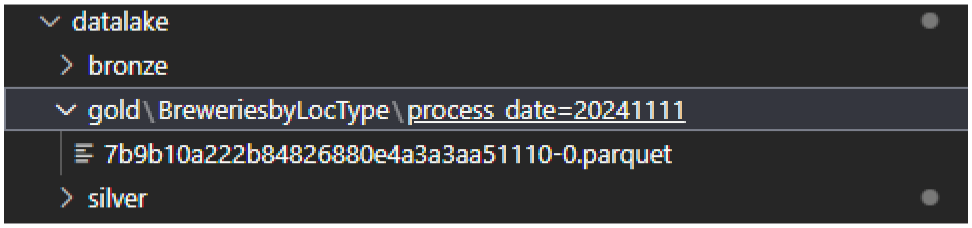
On datalake


```
silver\20241111\breweries
  country=Austria
  country=England
  country=France
  country=Germany
  country=Ireland
  country=Isle_of_Man
  country=Poland
  country=Portugal
  country=Scotland
  country=Singapore
  country=South_Korea
  country=Sweden
  country=United_States
```

Sucess on **AggBreweriesToGold:**



On datalake:



datalake
bronze
gold\BreweriesbyLocType\process_date=20241111
7b9b10a222b84826880e4a3a3aa51110-0.parquet
silver